Proactive Vulnerability Discovery and Assessment in Smart, Connected Systems Through Systematic Problem Analysis

by

Qi Chen

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2018

Doctoral Committee:

Professor Z. Morley Mao, Chair
Professor Henry X. Liu
Professor Atul Prakash
Assistant Professor Zhiyun Qian, University of California, Riverside
Professor Michael K. Reiter, University of North Carolina, Chapel Hill

Qi Chen

alfchen@umich.edu

ORCID iD: 0000-0003-0316-9285

To my parents, Professor Jianhua Chen and Mingxiu Ma,

and my love, Yu Stephanie Sun.

ACKNOWLEDGEMENTS

Getting a Ph.D. is quite a long and tough journey. Looking back at the past six years, I find myself constantly making mistakes, getting rejected, and questioning about whether I am really that talented to become a competent researcher in the end. The only reason that gets me this far is the tremendous help and support from so many people along the way.

First and foremost, I would like to thank my advisor, Professor Z. Morley Mao. She is always willing to make herself available whenever I need her guidance, no matter for research, work, or life. Her consistent passion about research and her encouragement when I stuck are no doubt the most important reason why I am able to overcome so many hurdles in my Ph.D. journey. I especially admire her endless energy and resilience in pursuing higher standards in research and career, making her a perfect role model for my upcoming faculty career. I am extremely fortunate to have her being my advisor.

Next, I want to thank my girlfriend, Yu Stephanie Sun, for accompanying me for more than 7 years since we were both undergraduate students in Nanjing University. Without her unconditional love and support, I don't think I can even have the courage to take on the challenge of pursuing a Ph.D. degree. Having her alongside me is the best thing that ever happened to me in my whole life.

I am very much grateful to my dissertation committee, Professor Atul Prakash, Professor Mike Reiter, Professor Zhiyun Qian, and Professor Henry Liu for their valuable feedback and help in refining the dissertation. I would like to especially thank Zhiyun for closely working with me on various research projects. His guidance is of tremendous help to me, especially at the early stage of my Ph.D. study.

I appreciate the collaboration opportunity with Dr. Eric Osterweil and Matt Thomas at Verisign Labs starting from my internship in May 2015. They taught me how to perform network measurement and data-driven research, which becomes a core skill set in my following research. I have been fortunate enough to have them being my mentors.

I would also like to thank Professor Henry Liu and Dr. Yiheng Feng for collaborating with me so successfully in the last two years of my Ph.D. study. They taught me so much about the transportation field, without which it's impossible to finish my last project – in my option, the most exciting one – about smart transportation security. The challenges I overcame in this highly interdisciplinary research collaboration will absolutely have a long lasting benefit in my upcoming career.

I also want to thank my colleagues and friends in Ann Arbor. I would like to first thank Dr. Yihua Guo and Haokun Luo. We three came to Michigan at the same year and became best friends ever since. I have highly enjoyable interaction with them both academically and socially throughout my Ph.D. study. We did tons and tons of funny (or silly) things together, for example the special interest group of foosball (SIGFOOS) and the player ranking system [112], and the "car-breaking" midnight chat with Yihua (please check out his dissertation for more details [210]). I also had lots of happy time with Dr. Earlence Fernandes, who started the Ph.D. program at the same year with me and also worked in the computer security research field. We were almost always roommates when travelling to security conferences and workshops, and shared nearly every piece of achievements and milestones in our Ph.D. study.

Also many thanks to my former group members, Dr. Qiang Xu, Dr. Junxian Huang, Dr. Sanae Rosen, and Mark Gordon, who helped me through tough times at the beginning of my Ph.D. study. In addition, I have enjoyed working on research projects together with Dr. Yunhan Jack Jia, Yuru Shao, Professor Ding Zhao, Professor Amir Rahmati, Professor Harsha Madhyastha, Professor J. Alex Halderman, Yucheng Yin, Yulong Cao, Shengtuo Hu, Jeremy Erickson, Chao Kong, Yikai Lin, David Ke Hong, Dr. Ashkan Nikravesh,

Jason Ott, Jie You, Shiqi Wang, Shihong Huang, Xiaochen Yu, EJ Lin, Rob Levy, and industry researchers Dr. Jie Hui, Aaron Drake, Dr. Kevin Lau, and Karthik Iyer. I also had a lot of fun with Yuanyuan Zhou, Professor Feng Qian, Professor Roya Ensafi, Dr. Mehrdad Moradi, Tom Andrews, Shichang Xu, Xiao Zhu, Lei Ruan, Deepak Kumar, Professor Michael Bailey, Dr. Yunjing Xu, Dr. Jing Zhang, Dr. Yue Liu, Dr. David Bild, Jiecao Yu, Shikai Li, Dongyao Chen, Dr. Yu-Chih Tung, Dr. Fei Li, Xianzheng Dou, Dr. Zhe Wu, Zhiqiang Sui, Professor Kassem Fawaz, Arun Ganesan, Mert Pese, Nishil Talati, Kevin Eykholt, Dr. Denis Foo Kune, Dr. David Devecsery, Dr. Michael Chow, Andrew Quinn, Dr. Sara Rampazzi, Tim Trippel, Ofir Weisse, Zhongjun Jin, Wai Wong, Xiaochi Li, Professor Eric Wustrow, Professor Zakir Durumeric, Travis Finkenauer, Ben VanderSloot, Matt Bernhard, and many others.

Finally, I want to thank my parents, Professor Jianhua Chen and Mingxiu Ma, for their guidance and support in both my life and career. Thanks again to Stephanie who has always been sources of love and support. This dissertation is dedicated to them.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

ABSTRACT

The world is increasingly connected through a series of smart, connected systems such as smartphone systems, smart home systems, and emerging smart transportation and autonomous vehicle systems. While leading to improved services, such transformation also introduces new security challenges. To address these challenges, in contrast to existing defense mechanisms that are mostly ad hoc and reactive, my dissertation research is dedicated to developing systematic problem analysis approaches that can proactively discover and assess new security problems in smart, connected systems.

To achieve this goal, my dissertation focuses on two most fundamental capabilities in any smart, connected system: network stack and smart control, and demonstrates that static/dynamic program analysis and network measurement can be used to systematically identify new code-level and network-level security challenges in smart, connected systems, and gain insights about problem severity to address design trade-offs in the defense solutions. More specifically, my research is able to leverage these techniques to discover a new attack vector (US-CERT alert TA16-144A) that is unexpectedly brought by the recent expansion in the DNS system, and new algorithm-level security vulnerabilities in the next-generation smart transportation systems. For these discoveries, systematic vulnerability cause analysis is performed subsequently to uncover the associated new network-level and code-level security challenges. On the defense side, these techniques are also used in my dissertation research to build the first automated detection tool for packet injection vulnerability, a recurring problem in network communication protocols, and define more useful attack surface to balance the design trade-off in name collision attack defenses.

CHAPTER I

Introduction

The world is entering a new era of transformational change: both ourselves as humans and our physical living environments are becoming increasingly connected by smart technologies. Such revolution starts from the popularization of smartphones, which are now connecting more than 2.5 billion individuals to the Internet and others on the move [86]. Soon after that, physical objects such as door locks, cars, and traffic lights are increasingly computerized [14, 55, 46], forming numerous Cyber-Physical Systems (CPS), and then further become ubiquitously interconnected with the recent advances of the Internet of Things (IoT) technology [102, 10, 52, 131]. The resulting systems from such transformation, which we call **smart, connected systems**, include *smart, connected end systems* such as smartphones, IoT devices, and emerging autonomous vehicles, and *smart, connected distributed systems* such as smart home and emerging smart transportation systems.

These smart, connected systems feature more ubiquitous network-based access and information sharing, and more functionality rich and usable control platforms, leading to new opportunities for innovation, improved services, and enhanced quality of life. However, such transformation also introduces new security challenges. First, the newly-introduced system capabilities, e.g., network connectivity and smart control, inevitably increase the attack surface and also the problem complexity, making security analysis and defense design more challenging than before. Second, to fully utilize the new system capabilities, new sys-

tem operation models (e.g., machine learning based perception and control in autonomous driving [138, 49, 60]) and new user interaction methods (e.g., voice control [4, 53, 44]) are typically created. This inherently introduces new security requirements, with which a secure and robust design can be fundamentally challenging. Third, even if these challenges at the design level are fully solved, challenges still remain at the implementation level. As repeatedly discovered in my dissertation research (e.g., for network protocols in §III and smart control algorithms in §VI), the actual implementation of security-critical features may not always conform to the design due to various reasons ranging from development mistakes to deployment-time constraints.

In this dissertation, my research focuses on the manifestation of these three security challenges in two most basic features in any smart, connected system:

(1) Network stack. Network connectivity is the most basic capability in smart, connected systems to enable more advanced sensing, actuation, and control. However, adding network-based access into previously isolated physical objects largely increases the attack surface, making remote compromise possible. For example, the widespread security weaknesses in today's IoT devices have already be remotely exploited and caused a series of massive Distributed Denial of Service (DDoS) attacks [81]. Wireless communication services in modern vehicles, e.g., Bluetooth and cellular connections, were found to have various vulnerabilities that allow remote attackers to have long distance vehicle control and location tracking [171]. To solve these problems, not only the design of the network protocols but also their implementations need to be secure, which are both challenging research problems today [250, 234, 152, 121, 266]. In addition, even with effective security features carefully designed and correctly implemented, it also requires the users of the protocols, e.g., developers at higher layers, to avoid misuses of these features, which is also shown to a common source of security vulnerabilities in practice [184, 196].

(2) Smart control. In smart, connected systems, the system controller is the key enabler of the advanced and intelligent services by making more optimized use of the network

connectivity and system capabilities, e.g., enabling autonomous driving based on a variety of machine learning algorithms [138, 49, 60], and enabling more intelligently traffic light control based on more optimized traffic control algorithms [82]. Like discussed earlier, a secure and robust design for these new system control features can be fundamentally challenging and in many cases even requires multidisciplinary research efforts to address. For example, it is found that the machine learning models used in smart, connected systems such as smart homes and autonomous vehicles are generally vulnerable to adversarial input and can be deliberately tricked into making wrong control decisions in autonomous driving [167, 247], voice assistants [165, 283], and face recognitions [263]. At this point, how to generate a sufficiently robust machine learning model is still an open problem that requires joint research efforts from both machine learning and security communities to solve [166, 233]. In my research, we find similar problems in smart traffic signal control algorithms, which we believe also requires joint research efforts from both transportation and security communities to solve (detailed later in §VI).

To address these security challenges, existing defense mechanisms are mostly ad hoc and reactive, creating case-by-case solutions to fix exposed vulnerabilities, many times even *after* they have been actively exploited in practice, such as the recent Mirai botnet and WannaCry ransom attacks [81, 137]. These solutions can neither systematically address the exposed problems in existing smart, connected systems, nor be applied to future smart, connected systems to prevent similar problems. With the transformation to smart, connected systems becoming increasingly faster and pervasive, this situation only deteriorates and becomes more favorable to attackers.

Research goal. To win this arms race, my research aims at developing systematic program analysis approaches that can *proactively* discovery and assess new security challenges in existing and future smart, connected systems. More specifically, these approaches *systematically* look for vulnerabilities using rigorous techniques such as static/dynamic program analysis for code-level vulnerabilities and network measurement for network-level

vulnerabilities, and thus lead to design improvements that are more systematic than before. Thus, even for future smart, connected systems in different application domains, most of the methodology and solution design can be adapted and evolved.

To achieve this goal, my research leverages three types of analysis techniques to achieve high rigorousness:

- *Static program analysis.* Static program analysis is capable of automatically analyzing the behaviors of computer programs based on their source code or binaries without executing them. Before the analysis, the targeted program behaviors usually need to be precisely defined at the code level. Thus, static program analysis is most useful in detecting and analyzing known classes of vulnerabilities. With the main benefit in automation, it has been applied to address various security problems including detecting vulnerabilities such as buffer overflow [182] and cross-site scripting [275], detecting privacy leakage [203, 158], detecting and analyzing malware [268], etc. Due to the lack of run-time information, tools built upon static program analysis techniques tend to overestimate a program's vulnerability status and thus have high false positive rates [262, 158].

- *Dynamic program analysis.* Dynamic program analysis analyzes a computer program by generating test inputs to trigger the behaviors of interest. Compared to static program analysis, dynamic program analysis techniques do not require access to the source code or binaries of the target programs, and thus can more conveniently handle situations where the binaries are obfuscated [278], or running remotely, e.g., on a remote server [170]. Also, since it executes the program, it can provide run-time information and handle dynamic program language features, and thus does not suffer from false positives. In addition, such analysis can have generic analysis metrics, e.g., code coverage in fuzz testing [5], instead of pre-defined vulnerability patterns, and thus may help discover new classes of vulnerabilities. However, it is very difficult to efficiently generate test inputs that can ensure a high code coverage, and thus

usually have high false negative rates [115]. Thus, it is complementary to static analysis techniques, and sometimes they are used in combination to balance the trade-off between efficiency and effectiveness [160, 289].

- *Network measurement.* Since static and dynamic program analysis techniques target computer program behaviors, they are applicable to vulnerabilities at the code level but not those at the network level, e.g., those caused by configuration, policies, or inter-dependencies in networked systems. For the latter, network measurement is an effective method to systematically discover new network-level vulnerabilities [193], analyze vulnerability status [192, 244], analyze vulnerability causes [172], monitor vulnerability patching status [252], etc. In such network measurement based analysis, it needs to first define the network traffic patterns of interest, e.g., a vulnerability related traffic signature, which is similar to the program behavior used in static and dynamic program analysis. Since connectivity is the most basic capability in smart, connected systems, network measurement is a necessary vulnerability analysis technique when studying the network-level security problems in these systems.

These three analysis techniques are thus generally applicable for systematically discovering and characterizing security problems that can manifest as patterns in source code, run-time system behavior, or network traffic. When applying these techniques, it's usually necessary to address various design challenges due to the need for balancing different properties of the solution systems, e.g., efficiency, effectiveness, scalability, etc. For example, one program analysis technique, symbolic execution, is capable of creating inputs to all execution paths in a program, but suffers from severe scalability limitations and can hardly be applied to important real-world code bases such as the Linux kernel [254]. Thus, care must be exercised to choose appropriate analysis techniques and creatively define analysis targets in the solution system design to solve a problem in practice. In this dissertation, one common strategy my research uses is to first understand the different levels of problem severity and then design the solution system to target the most severe subset of the problem. For

example, my work in §III is able to identify a subset of packet injection vulnerabilities that has the highest exploitability in practice and then design the vulnerability detection system to prioritize them, which is found to effectively reduce false alarms without compromising vulnerability detection effectiveness.

In this dissertation, my research demonstrates that **static/dynamic program analysis and network measurement can be used to systematically identify new code-level and network-level security challenges in smart, connected systems, and gain insights about problem severity to address design trade-offs in the defense solutions.** More specifically, my dissertation research is able to leverage these analysis techniques to (1) build the first system to automatically detect packet injection vulnerability, a recurring problem in network communication protocols; (2) discover a new attack vector (US-CERT alert TA16-144A [127]) that is unexpectedly brought by the recent expansion in the DNS system, and perform subsequent systematic analysis at both network and software levels for its defense, and (3) perform the first security analysis of the next-generation Connected Vehicle (CV) smart transportation system, which discovers new security vulnerabilities at the level of the traffic control algorithm.

Research impact. My dissertation research has impact in both academia and industry in the form of research papers in top-tier security conferences, media coverage in The Register, SC Magazine, Security Week, Naked Security, Bleeping Computer, etc. [56, 125, 143, 140, 111, 88, 117, 126, 142], vulnerability disclosures such as a US Department of Homeland Security (DHS) US-CERT Alert [127], and industry discussions and responses [136, 100].

1.1 *Network Stack*: Systematic Detection of Packet Injection Vulnerabilities in Network Communication

In the network stack of smart, connected systems, off-path packet injection attacks remain a serious threat to communication integrity, causing attacks such as phishing and malicious script injection. Current solution is to apply case-by-case patches, but due to the complex nature of the problem, new variants of packet injection vulnerabilities are still emerging in recent years, targeting critical protocols such as TCP. We argue that such recurring problems need a systematic solution. In my dissertation research, we design and implement PacketGuardian, a precise static taint analysis tool that comprehensively checks the packet handling logic of various network protocol implementations [173]. The analysis operates in two steps. First, it identifies the critical paths and constraints that lead to accepting an incoming packet. If paths with weak constraints exist, a vulnerability may be revealed immediately. Otherwise, based on "secret" protocol states in the constraints, a subsequent analysis is performed to check whether such states can be leaked to an attacker.

In the second step, observing that all previously reported leaks are through implicit flows, our tool supports implicit flow tainting, which is a commonly excluded feature due to high volumes of false alarms caused by it. To address this challenge, we propose the concept of attacker-controlled implicit information leaks, and prioritize our tool to detect them, which effectively reduces false alarms without compromising tool effectiveness. We use PacketGuardian on 6 popular protocol implementations of TCP, SCTP, DCCP, and RTP, and uncover new vulnerabilities in Linux kernel TCP as well as 2 out of 3 RTP implementations. We validate these vulnerabilities and confirm that they are indeed highly exploitable. Detailed results are summarized on our project website [87].

1.2 *Network Stack*: Discovery and Systematic Analysis of Name Collision Vulnerabilities in Network Service Discovery

Besides network communication, another important network function in smart, connected systems is service discovery, which helps an end system automatically configure network services. We find that Web Proxy Auto-Discovery (WPAD), a popular service discovery protocol based on domain name system (DNS), has significant DNS query leakage problem and can be exploited to launch Man in the Middle (MitM) attacks from anywhere in the Internet. We call this newly-exposed MitM attack vector *WPAD name collision attack*, and perform the first systematic vulnerability study [172]. We first characterize the query leakage problem to understand the fundamental problem cause, and then use attack surface definition and quantification to systematically study the vulnerability status in the wild. Our results show that 10% of highly-vulnerable domains have already been registered and their exploitation can start at any time, showing real threat to Internet users. Based on our analysis, we propose and empirically evaluate a set of remediation strategies. Due to the significant impact on corporate and end users, the U.S. Department of Homeland Security (DHS) released a US-CERT alert based on our work Various DNS operators contacted us to obtain the list of highly-vulnerable domains [127]. Domain name company Verisign also acknowledged the attack severity [136].

While we have shown that the name collision problem is a real threat today, our understanding of its impact on the internal services is limited to the WPAD service. In fact, over 600 services are registered to support DNS-based service discovery, and thus the name collision problem may have much broader impact than the WPAD service alone. Thus, we then generalize the WPAD exploit to a new class of attacks on DNS-based service discovery, and perform a systematic study of the affected services under name collisions [175]. We find that the name collision problem broadly breaks common security assumptions made in todays service software, and nearly all of the affected services expose vulnerabil-

ities in popular clients. To demonstrate the severity, we construct exploits and find many new name collision attacks including another MitM attack vector, document leakage, malicious library injection, and credential theft. Leveraging the insights from our analysis, we propose multiple service software level solutions. We have performed responsible disclosure and got email acknowledgement from Apple, Microsoft and Comcast on the identified vulnerabilities.

1.3 *Smart Control*: Systematic Discovery and Analysis of Algorithm-level Vulnerabilities in Next-generation Smart Transportation

As discussed earlier, smart control is another fundamental capability in smart, connected systems besides network connectivity. Thus, I also perform vulnerability analysis on system controllers, by focusing on the emerging Connected vehicle (CV) based smart transportation systems. In such systems, vehicles and the transportation infrastructure are connected through wireless communication, which is an ongoing effort of the U.S. Department of Transportation (USDoT) to dramatically improve the transportation systems in mobility, safety, environmental impact, and public agency operations [129]. Having demonstrated the potential to greatly improve transportation mobility efficiency, such dramatically increased connectivity also opens a new door for cyber attacks. In this work, we perform the first detailed security analysis of the next-generation CV-based transportation systems [176]. As a first step, we target the USDOT (U.S. Department of Transportation) sponsored CV-based traffic control system, which has been tested and shown high effectiveness in real road intersections. In the analysis, we target a realistic threat, namely CV data spoofing from one single attack vehicle, with the attack goal of creating traffic congestion.

We first analyze the system design and identify data spoofing strategies that can potentially influence the traffic control. Based on the strategies, we perform vulnerability analysis by exhaustively trying all the data spoofing options for these strategies to understand

the upper bound of the attack effectiveness. For the highly effective cases, we analyze the causes and find that the current signal control algorithm design and configuration choices are highly vulnerable to data spoofing attacks from even a single attack vehicle. These vulnerabilities can be exploited to completely reverse the benefit of the CV-based signal control system by causing the traffic mobility to be 23.4% worse than that without adopting such system. We then construct practical exploits and evaluate them under real-world intersection settings. The evaluation results are consistent with our vulnerability analysis, and we find that the attacks can even cause a blocking effect to jam an entire approach. In the jamming period, 22% of the vehicles need to spend over 7 minutes for an original half-minute trip, which is 14 times higher. We also discuss promising defense directions leveraging the insights from our analysis.

1.4 Dissertation Organization

This dissertation is structured as follows. Chapter II describes background and related work for the problem domains studied in my dissertation research. In Chapter III, we describe our design and implementation of the first automated detection tool for packet injection vulnerability, a recurring problem in network communication protocols. In Chapter IV, we describe our discovery of a new attack vector, WPAD name collision attack, that was unexpectedly brought by the recent expansion in DNS, and our subsequent systematic cause analysis and vulnerability assessment. Chapter V then describes our generalization of the WPAD name collision attack and the first systematic analysis of the vulnerability status at the service level under this generalized class of attacks. In Chapter VI, we describe the first security analysis of next-generation smart transportation and the discovery of new algorithm-level security vulnerabilities. In Chapter VII, we conclude this dissertation and discuss potential future directions.

CHAPTER II

Background and Related Work

In this chapter, we describe the background and related work for the three problem domains studied in my dissertation research: network communication protocol security, DNS system security in the new gTLD era, and software security in next-generation smart transportation.

2.1 Problem Domain 1: Network Communication Protocol Security

2.1.1 Related Work

Network protocol analysis. To detect protocol design vulnerabilities, prior work has used formal methods such as model checking and specification languages to perform rigorous protocol specification testing [163, 162]. However, these cannot prevent vulnerabilities due to weak implementations. For implementation-level vulnerabilities, static analysis has been applied to identify system DoS vulnerabilities [169], on-path protocol manipulation attacks [225], and protocol interoperability problems [248]. However, none focused on the off-path packet injection vulnerability studied in this paper. In addition, due to the unique vulnerability pattern (detailed in§3.3.2), our work needs to support implicit data flow analysis and address the associated challenge of high false positive rates, which is not handled by previous work.

Side-channel attack and detection. Recently years witness a rise in the discovery of new side channels. For **storage channels**, it has been found that proc file systems can be abused as side channels to infer keystrokes [285], webpage [215], and system state [174]. In particular, Qian et al. [250, 251] used proc file packet counters to infer TCP sequence number. Another popular channel is **timing channel**, including code path [222], data [155], and cache-access timing channel [282, 209]. In network protocol attacks, some header fields are also found to be useful for inferring sequence number [204]. In comparison, our goal is not to report new side channels but focuses on designing an automated tool to systematically detect side channels.

In contrast to new side channel discovery, automated detection of side channels has been less explored. Dynamic analysis such as black-box testing has been used to find side channels in web application [170], and timing side channels in SSL/TLS implementation [237]. To overcome the limitation of dynamic analysis in the analysis completeness, static analysis tools are also developed to detect web application and cache side channels [284, 188]. In comparison, our work focuses on storage side channels for network protocol states, which is not covered by existing tools. In addition, our work identifies and prioritizes the detection of a new category of highly-exploitable implicit information leaks called *attacker-controlled implicit information leaks* (detailed in§3.3.2), which is also not discussed in previous work.

Static analysis for taint-style vulnerability. For taint-style vulnerabilities, static analysis tools have been designed to detect buffer overflow [182], format string vulnerabilities [261], and SQL injection and XSS [216, 220, 271]. Recently, Yamaguchi et al. [281] propose to use code property graph to effectively mine such vulnerabilities in large amounts of code. Different from them, our analysis targets packet injection instead of code injection, which requires handling much more and also diverse checks due to header field semantics. Moreover, we have a follow-up leakage analysis which is not included in previous tools.

Static taint analysis are also used to detect information leakage vulnerabilities in recent

years, especially for privacy leakage in Android system [158, 203, 208]. However, these tools exclude implicit flow tainting due to its low-entropy in leakage and the problem of high FPs [221]. In comparison, our tool taints implicit flows as required by our analysis goal, and proposes to target attacker-controlled implicit information leaks to mitigate the FP problem while maintaining high accuracy.

Quantitative information flow for side channel assessment. Quantitative information flow (QIF) is an approach to estimate the capacity, e.g., the entropy, of information leaks [230, 181]. Previous work used QIF to quantify the information leaks from cache side channels [189] and from network traffic of web applications [284, 170, 177]. In comparison, our work focuses on research challenges in detecting information leaks in network protocols, instead of the challenges in quantify them. Following up our work, Zhou et al. design a framework for more scalable and flexible side-channel leakage assessment in software, which is able to quantify the storage side channels studied in this paper [292].

2.2 Problem Domain 2: DNS System Security in the New gTLD Era

2.2.1 Background: DNS Ecosystem

DNS (Domain Name System) [238] is a distributed system which translates domain names to network service identifiers (such as IP addresses for computers in the Internet or a private network). Domain names are a set of labels separated by dots, for example `www.example.com`, and are organized in hierarchical subdomains of the DNS root domain. The first level of domain name labels under the root domain are the TLDs [123], including gTLDs such as `.com`, and country code Top-Level Domains such as `.us`. Directly below TLDs are Second-Level Domains (SLD) [104], e.g., `example` in `www.example.com`. In this dissertation, the term *domain* is defined to be any DNS name, and TLDs and SLDs are specific types of domains.

Domain name management and delegation. In DNS, a DNS zone is defined as the

set of DNS domain names that are contiguous in the DNS tree hierarchy, and which are administered by the same authority. The DNS root zone is the canonical top of the DNS tree. It is the authoritative zone for all of DNS' TLDs. The structure and contents of the DNS root zone are determined by an organizational role called the Internet Assigned Numbers Authority (IANA), which is performed by the Internet Corporation for Assigned Names and Numbers (ICANN). The DNS root zone's actual operational and authoritative maintainer is a role called the Root Zone Maintainer (RZM), which is currently performed by Verisign. ICANN delegates the management of its subdomains, the TLDs, to TLD registry operators. Under TLDs, SLDs are registered in the process of domain name registration.

Domain name registration. A domain name registration is the delegation of the administration of an SLD and its subdomains under a TLD, which usually involves 3 parties: TLD registry operators, registrars, and registrants [246]. At a high level, registry operators manage TLDs, registrars conduct the daily business of transacting with clients for SLDs, and registrants pay to receive administrative authority to run SLDs. Once a domain is registered by a registrant, the registrar submits certain information to the corresponding TLD registry operators, and the WHOIS database [141] then maps the registered domain name to the registrant details.

Domain name resolution. In the domain name resolution process, end hosts rely on recursive DNS resolvers, usually configured by network providers, e.g., corporate network administrators and home network providers. Using the cached results whenever possible, the resolvers query the name servers following the DNS domain label hierarchy, getting either the corresponding IP address, or an NXDomain response (`rcode 3` in RFC1035 [239], NXD for short), indicating that no such domain name exists.

The New gTLD Program. In the history of DNS, the set of TLDs has remained relatively small and stable, with only 66 new TLDs added in 14 years before 2013 [245]. In 2011, with the goal of enhancing competition and consumer choice, ICANN approved the launch of the New gTLD Program [146], which in less than 2 years has added over 700

new gTLDs as of 2015/08/25. To differentiate these new gTLDs from the legacy ones such as `.com`, in this dissertation they are also referred to as nTLDs. This enormous wave of new gTLD delegation raised name collision concern in the domain name industry [245], and in this dissertation, we perform the first systematic study of one of the consequences of this problem in the wild.

2.2.2 Background: Internal DNS Namespace and iTLD Usage

The DNS ecosystem described above is the public DNS namespace for domain names visible to the Internet. Similarly, a local area network, e.g., a corporate network, can also set up an internal DNS namespace with private domain names. This helps control the access to internal confidential information, and can operate despite any external network connectivity disruption, making it a common practice for companies.

To create an internal DNS namespace, internal name servers are used to serve the zone files for a customized internal domain, and the resolvers are configured to query these servers instead of the DNS servers in public namespace. To make the internal domain name easy to reference and also to prevent confusion between internal and public namespaces, some administrators in the past used TLD strings that have not been delegated (in the public DNS namespace) as iTLDs.

The use of iTLDs implicitly assumes that these TLD strings will not be delegated in the public namespace; however, with the launching of the New gTLD Program, many of the popular iTLD strings have already been delegated today and are open for public registeration [151]. This breaks the implied assumption that previously undelegated iTLDs will never be delegated. As a side effect, the leaked internal queries to these iTLD strings that were previously benign now expose issuers to the MitM attacks studied in this dissertation.

2.2.3 Background: DNS-based Service Discovery

The WPAD proxy configuration belongs to a general class of DNS-based service discovery processes that utilize named and structured DNS records to facilitate service discovery in a discovery domain. The traditional approach for the discovery issues A or AAAA DNS queries with the service name prepended to the discovery domain. A more advanced approach is to use SRV records [1]. To discover service `svc` over transport protocol `prot` (e.g., TCP or UDP) in domain `comp.ntld`, the SRV query format is `_svc._prot.comp.ntld`. From the response, the client obtains the server's domain name and the port number. Subsequent A or AAAA queries are then issued to obtain the server's IP address.

This DNS-based discovery process is formally defined in RFC 6763 [40], named the DNS-based Service Discovery or DNS-SD. In the discovery process, a DNS PTR query is first issued to retrieve a list of available service instance names. For each instance name, an SRV query is then issued to locate the server name and port. Like the traditional SRV-based discovery process, the PTR and SRV queries in DNS-SD all use the format `_svc._prot.comp.ntld`. DNS-SD is compatible with both unicast DNS and multicast DNS (mDNS) [179]. When used with mDNS, DNS-SD can provide Zeroconf [144], which can discover services on nearby devices in local link without setting up unicast DNS servers. A popular Zeroconf implementation is the Apple Bonjour [22], which is built-in with the latest macOS [23].

My dissertation research considers the general concept of DNS-based service discovery including both the standard DNS-SD procedure and the traditional approach. For the query format, we refer to the queries in the form of `_svc._prot.comp.ntld` as *standard* queries and others as *non-standard* queries. To standardize the discovery process, the official use of certain service names are registered in the Internet Assigned Numbers Authority (IANA) service name registry [65]. In this dissertation, we refer to the service names in the IANA registry *registered* names and others as *non-registered* names.

2.2.4 Background: Server Authentication Mechanisms

To prevent connecting to an unintended server, the service client can perform server authentication to validate the server identity before performing the designed service functionality. When TLS is used, the client can use the server's TLS certificate to certify the server's ownership of the requested name subject, e.g., the domain name. In the validation process, the certificate chain is inspected to check if the certificate is issued by a trusted certification authority (CA). For the public Internet, a set of trusted third-party CAs are preinstalled in popular OSes or browsers. For an internal network, the network administrators typically use self-signed local CAs [24], which are installed into the end user systems beforehand.

Another popular authentication approach is to use a PSK distributed to the client and the server. PSK-based authentication methods can be used for client authentication only, for example by sending the key in plain text or hashed format to the server. Some methods can provide both client and server authentications called mutual authentication, e.g., Kerberos [73] and DIGEST-MD5 [134]. In my dissertation research, we perform a systematic vulnerability analysis to understand whether the clients with server authentication support are robust enough under the name collision threat model.

2.2.5 Related Work

DNS spoofing attacks. Like the WPAD name collision attack studied in §IV in this dissertation, some previous DNS spoofing attacks also try to deceive victims using malicious DNS response. One attack category assumes that the attacker is MitM and thus replies forged response when observing a query. This can be achieved through attacking the network configurations of the victim devices. For example, prior work [269, 59] show that scripts on web pages can change home routers' DNS configurations and point the client resolver IP to attacker's servers. Another category of attacks assumes that the attacker is off path. One such example is DNS cache poisoning attack [267, 240], which corrupts the

resolver's cache with spoofed DNS responses, causing all downstream devices to be redirected to the attacker's IP addresses. These previous attacks exist since the victim cannot determine whether the received DNS responses are legitimate or manipulated, which can be solved by DNSSEC protocol [157, 156]. Compared to them, the attacker in the WPAD name collision attack is actually authoritative for the request domains. This means that she can legitimately give malicious response and launch MitM attack without the need of spoofing, making it exploitable even if DNSSEC is used.

Attack on DNS and DNS-based service discovery. To attack end systems using malicious DNS responses, previous attacks require the attacker to be either on the resolving path [190], or off the path but physically inside the targeted network [267]. Compared to these attacks with tight attack placement and timing requirements, the name collision attack studied in this dissertation only needs a domain registration to exploit users from all over the world, which are thus easier to launch and also of larger scale. Recently, Lever et al. proposed the concept of residual domain trust abuse in the public DNS namespace, and used DNS traffic to characterize such abuse [227]. In comparison, residual trust exploitation is for the same domain in a single namespace triggered by domain re-registration, but the trust exploitation in this work is for domains across namespaces triggered by the name collision problem. Besides, our work also performs vulnerability analysis at the service software design level, which is not discussed in previous work.

Besides DNS systems, there has also been work on studying the security problems in using DNS-SD. Könings et al. analyzed the mDNS traffic in a university network to study the privacy leakage [223]. Xing et al. studied the major Zeroconf frameworks, and found popular apps such as AirDrop are vulnerable to MitM attacks [159]. Compared to these local network attacks, our work considers the name collision attack threat model, which are more powerful, of larger scale, and easier to launch (discussed in §5.2.1). Due to such threat model difference, our analysis covers not only the local-link discovery usage scenario targeted in these previous work, but the unicast DNS domain discovery usage scenario as

well.

New TLD delegation study. The addition of new gTLDs into the DNS root zone usually requires considerable debate about the extent to which new TLDs will actually serve a real need. Before the New gTLD Program, the growth of gTLD set maintained a very slow and steady rate. Some previous work studied the impact of certain early gTLD delegation, e.g., for `.biz` [213] and `.xxx` [212], and recently Halvorson et al. perform the first study targeting the New gTLD Program [211]. These studies mostly focus on characterizing the registration intent; in comparison, our work targets newly-discovered security problems specifically introduced by the new gTLD delegation.

Name collision from new gTLD delegation. Before our work, concerns from the domain name industry have already been raised about potential name collision problem from new gTLD delegation [243]. Several studies have measured the leaked DNS queries to the DNS root servers and shown the potential risks of information leakage, denial of service, and MitM attack [245, 265]. The discussions resulted in a name collision management framework from ICANN in 2013 [68], which allows the majority of new gTLD strings to be delegated by following an Alternate Path to Delegation (APD). In APD, the new gTLD registries are required to block large numbers of high-risk SLDs according to measurement of DITL (Day in the Life of the Internet) dataset. Later on in 2014 a new framework allows releasing these blocked names after a 90-day period called "controlled interruption" for testing and resolving name collision problem [67]. However, previous studies have shown that the block list is ineffective due to the statistical limitation of DITL dataset [270]. In addition, the controlled interruption period is unlikely to change anything for problems similar to the WPAD name collision attack, since the victim machines automatically perform the vulnerable operations even without user awareness [265]. This indicates the lack of a systematic approach to understand and find effective solutions for the newly-exposed name collision problem. Our work uses in-depth cause analysis and attack surface quantification to fill this critical research gap.

Vulnerability in server authentication usage. Previous work uncovered a series of security problems in server authentication in TLS, e.g., certificate validation vulnerabilities due to incorrect use of TLS APIs [196, 200]. In comparison, my dissertation research uncovers additional usage that is not incorrect or weak by itself, but only becomes vulnerable under the name collision attack threat model. For PSK-based authentications, some methods are known to be weak due to the lack of server authentication, e.g., Basic and NTLM [61, 30]. Instead of finding new security problems in these authentication mechanism, our work focuses on characterizing the *vulnerable use* of these weak methods in the network service clients, which is found to be a common vulnerable design choice under name collision attacks.

2.3 Problem Domain 3: Software Security in Next-generation Smart Transportation

2.3.1 Background: CV Technology and Recent Advances

Connected vehicle (CV) technology uses wireless communications to connect vehicles and the infrastructure with the goal of dramatically improving the transportation systems in mobility, safety, environmental impact, and public agency operations [129]. Due to the high data transmission requirement in the transportation scenario, the DSRC (Dedicated Short Range Communications) protocol is specifically designed for the CV communication scenarios with dedicated band allocated by the Federal Communications Commission (FCC) [43].

The communication in the CV environment has two categories: vehicle-to-vehicle (V2V) communication, and vehicle-to-infrastructure (V2I) communication. To support them, both the vehicle and the infrastructure sides need to install DSRC devices, which are called On-Board Units (OBUs) and Road-side Units (RSUs) respectively. In such CV environment, vehicles use OBUs to periodically broadcast Basic Safety Messages (BSM)

including its real-time trajectory data, e.g., location and speed, to the surrounding vehicles and infrastructure. This enables a series of safety functions on the vehicle side, e.g., blind spot and lane change warnings, and also enables the traffic infrastructure to leverage the real-time traffic data to improve traffic control performance.

Recent advances in the CV deployment. With the DSRC standard becoming mature [219], OBUs and RSUs products are already on market today [28]. USDOT estimates that equipping the OBUs would cost around $341 to $350 per vehicle in 2020 [130]. This makes the CV technology a very cost-effective option to increase transportation system performance in practice, and the USDOT has already proposed to mandate all new light-duty vehicles to equip OBUs [128]. The market penetration rate will gradually increase after such mandate [135], and in our analysis we call the vehicles with and without OBUs *equipped vehicles* and *unequipped vehicles* respectively.

To foster the development of CV-based transportation systems, in 2010 the USDOT launched the Dynamic Mobility Applications (DMA) research program and developed nearly 70 such systems, or CV applications [35]. To encourage service providers, researchers, and application developers to participate, these applications are open sourced and are available free to the public [89]. Built on the success of the DMA program, on September 1, 2016, the USDOT awarded $45 million to start small-scale deployment of these systems, called the CV Pilot Deployment Program, in three sites including New York [129]. In my dissertation research, we perform the first security analysis of such CV-based transportation systems as a timely study to understand the potential security problems and challenges at the design level before large-scale deployment.

Security and Credential Management System (SCMS). As one of the most importatnt infrastructure, the transpiration systems are highly security and safety critical. Thus, to enhance the communication security in the CV environment, the USDOT will deploy the Security and Credential Management System (SCMS) on both the vehicle and infrastructure sides [133]. It is a Public-Key Infrastructure (PKI) system that requires every BSM

messages to be signed by the sender's digital certificates issued beforehand, and thus the receivers can verify the signature before acting on it [133, 276].

2.3.2 Related Work

Data spoofing attack in the CV environment. Similar to our work, previous work also identifies data spoofing as a realistic attack vector in the CV environment. Amoozadeh et al. studied the V2V-based automated vehicle platoon system, and found that spoofed attacks can cause rear-end collision or significant instability [153]. A more recent work summarizes a comprehensive list of data spoofing attack sources including not only DSRC but also other sensors such as GPS [187]. While these work focus on data spoofing attacks on the vehicle side through V2V, our work is the first study that exposes concrete data spoofing attacks on the transportation infrastructure side through V2I. In addition, compared to V2V attacks that can at most affect one lane of vehicles at a time, V2I attacks can affect all vehicles in an intersection as concretely shown in our evaluation, and thus are able to cause much wider impact on the transportation system.

Critical infrastructure security. Several studies have investigated the security of critical infrastructure and facilities, e.g., smart grid [154, 229]. These studies highlight the security challenges and the severe consequences brought by introducing connectivity into these previously isolated critical systems, which is also concretely shown in this work for the next-generation CV-based transportation. Closer to our work, Ghena et al. performed the first publicly available security analysis of a deployed traffic infrastructure system [202]. Their work found that the traffic controllers uses weak credentials and can be remotely controlled by the attacker. In comparison, our work targets the next-generation CV-based traffic control instead of the traditional one. In addition, the weak credential problem they discovered is a known problem across many embedded network devices [183], and can be fixed using state-of-the-art authentication mechanisms [202]. In comparison, our study assumes that such problem has already been solves, and targets new security problems at the

traffic control algorithm level.

Traffic control algorithm security. Prior to our study, very few studies explored the security problems in the traffic control algorithms. Laszka et al. performed a theoretical analysis to estimate the potential congestion an attacker can create assuming that she can arbitrarily compromise multiple signal controllers [226]. A follow-up study was then performed for the same attack goal but with a weak assumption, in which the attacker can only compromise the sensors that collects traffic flow information [201]. In comparison, neither of these work analyzes the CV-based signal control scenario targeted in our work. In addition, compared to their thread model that assumes the ability of compromising arbitrary numbers of infrastructure-side devices, our threat model, data spoofing from one signal attack vehicle, is much more realistic (§6.3).

CHAPTER III

Systematic Detection of Packet Injection Vulnerabilities

3.1 Introduction

The encryption coverage on today's Internet is unfortunately still poor: only 50% in 2017 [58]. Thus, off-path packet injection attacks remain a serious threat to network security. Recently a number of such attacks and their variants have been reported including off-path TCP packet injection [250, 251, 204, 205] and DNS cache poisoning attacks [267, 240]. These attacks jeopardize the integrity of network communication, and lead to serious damage where personal data from unsuspecting users can be leaked when visiting a web site. Despite application-layer encryption support (e.g., SSL and TLS), network connections are still vulnerable. For instance, for HTTPS connections, the initial request sent by the browser may still be an unencrypted HTTP request, and the server subsequently redirects the client to the HTTPS site. As shown in a recent study [250], an off-path attacker can inject a legitimate response to the very first HTTP request. Furthermore, such packet injection attacks can result in DoS, e.g., by injecting a reset (RST) packet with an inferred TCP sequence number.

To combat such threats, the network stacks typically implement stringent checks on various fields to verify if an incoming packet is valid. In fact, a number of RFCs like RFC 5961 [253] are dedicated to this purpose. However, two problems remain. First, the design of an RFC may not be formally verified to be secure. Second, even if the design

is secure, the actual implementation may not always conform to the design. In fact, the implementation is generally much more complex and difficult to get right. For instance, it has been shown that TCP implementations on Linux and FreeBSD are significantly weaker than what the RFC recommends regarding the mitigation against off-path attacks [251]. This calls for a systematic approach to verify protocol implementations.

In this work, we fulfill this very need by developing an effective and scalable static program analysis tool, *PacketGuardian*, which can systematically evaluate the robustness (e.g., the level of security strength) of a network protocol implementation against off-path packet injection attacks. To ensure effectiveness and accuracy, our tool uses a precise context-, flow-, and field-sensitive static taint analysis with pointer analysis support. To address the scalability challenge caused by such high analysis sensitivity, we choose a data flow analysis of summary-based approach, which is known to be more scalable compared to other frameworks [258], and is demonstrated to scale to very large code base like the Linux kernel [280].

At a high level, the tool operates by performing analysis in two steps: (1) Find all paths leading to the program execution point of accepting an incoming packet. This helps identify the critical checks that a protocol implementation relies on to prevent packet injection, and may directly reveal a packet injection vulnerability if any check is weak. (2) Motivated by the observation that strong checks typically rely on certain hard-to-guess or "secret" communication protocol state, e.g., TCP sequence numbers, or RTP source IDs, we perform a subsequent analysis to check whether such secret states can be leaked to an attacker through side channels.

In network protocol implementations, these "secret" protocol states are unlikely to be leaked directly through explicit flows, and all previously reported leakage has been through implicit flows [250, 251, 204]. Therefore, PacketGuardian supports implicit flow tainting, which is known to be of much less value compared to explicit flow tracking (implicit flow usually leaks at most 1 bit of information) and at the same time cause large numbers of

false positives [221]. It is thus a commonly excluded feature in nearly all taint analysis tools [158, 203, 208, 216]. To address the false positive challenge without compromising tool effectiveness, we leverage a key insight that the previously-discovered practical leaks are all *attacker-controlled implicit information leaks*, meaning that an attacker can influence which bit to leak. By prioritizing this special type of leak, we effectively reduce the false positive number and make the tool more useful for finding practical vulnerabilities.

Our analysis requires access to source code, which is a realistic assumption for many key network protocols. The tool we have developed is fully functional and is able to analyze arbitrary portions of the Linux kernel source code. By applying our tool to the Linux kernel TCP, SCTP, DCCP, and variants of open source RTP protocol implementations, we are able to identify a set of new vulnerabilities not previously reported. For example, for the 3 RTP implementations, two can be compromised by injecting less than 51 packets. For the Linux kernel TCP implementation, our tool identifies 17 high-entropy protocol state leakage, with 11 of them successfully validated in a realistic test bed. This illustrates that the Linux kernel TCP stack is still vulnerable even after the recent patches for the previous known leakage [253, 191], indicating the complex nature of the problem.

The contributions of this work are as follows:

- We formulate the problem to systematically analyze the security properties of network protocol implementations against off-path packet injection attacks, and develop an effective and scalable static program analysis tool to address it using a precise context-, flow-, and field-sensitive taint analysis with pointer analysis.
- To enable the detection of practical information leaks due to implicit flows while ensuring low false positives, we propose the concept of attacker-controlled implicit information leaks and prioritize our tool to detect them. To the best of our knowledge, we are the first to design a taint analysis tool for detecting attacker-controlled implicit information leaks.
- We implement and apply our tool on 6 real implementations for 4 network proto-

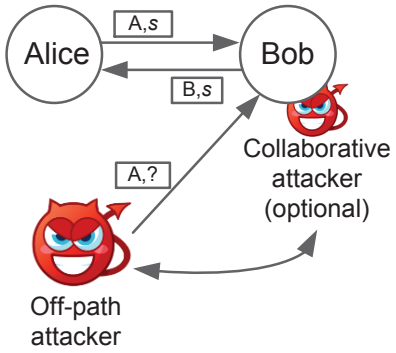

Figure 3.1: Packet injection attack threat model in §III.

cols. From the result, we are able to discover new and realistic vulnerabilities confirmed by proof-of-concept attacks for Linux kernel TCP and 2 out of 3 RTP implementations.

3.2 Attack Threat Model

Fig. 3.1 depicts the threat model for the off-path packet injection attack considered in this work. As shown, an existing communication channel (e.g., a TCP connection, a UDP session, or RTP session) is established between Alice and Bob. The attacker's goal is to inject a packet into the channel targeting Bob, pretending to be a packet from Alice. The attack goal can be to inject payload, e.g., to launch attack such as phishing, or to trigger the termination of the channel, resulting in denial-of-service (DoS). The attacker in this threat model is off-path, i.e., much weaker and more realistic than a man-in-the-middle attacker. To ensure channel integrity, Alice and Bob usually share several secret protocol states, denoted as s in the figure, and include it in the packet. These states are unknown to the off-path attacker and should be hard to guess.

To incorporate recently-discovered packet injection vulnerabilities [250, 251, 204, 205], our threat model also *optionally* considers a collaborative attacker sharing the same system as Bob. This collaborative attacker can be an unprivileged malware program [250, 251], or a script in the browser [204, 205]. This collaborative attacker is tasked to provide feedback

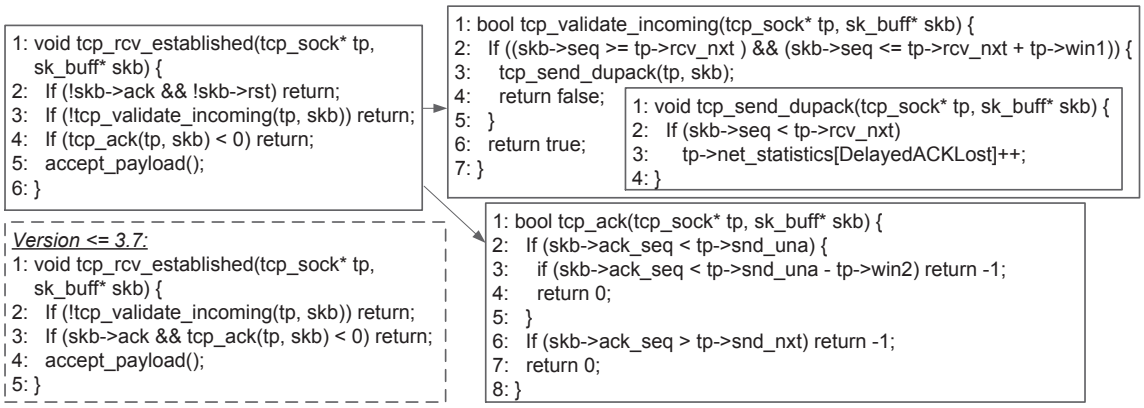

Figure 3.2: An illustrative code example of a simplified implementation for handling an incoming TCP packet in Linux kernel 3.15.

about any packet injection attempt of the off-path attacker, facilitating the inference of the secret protocol state for a successful injection.

3.3 Illustrative Example

3.3.1 Packet Injection Attack for TCP

To illustrate how static analysis can help detect packet injection attacks for TCP, Fig. 3.2 shows a significantly simplified implementation example for handling an incoming TCP packet, which is the entry for an injection packet from an off-path attacker. This implementation is mostly based on Linux kernel 3.15, from which we only include the important logic, i.e., sequence and acknowledgment number checks. In this figure, `tcp_rcv_established()` is the main entry function, parameter `tp` is the socket status maintained by the system, and parameter `skb` is the data structure for the incoming packet. Function `accept_payload()` copies the packet data into the application layer, indicating the acceptance of the incoming packet for this TCP connection, i.e., a successful injection.

To evaluate the robustness of this implementation against off-path packet injection, the key question is **what strong checks exist to prevent an off-path injected packet**

from reaching `accept_payload()`. As we can see in `tcp_rcv_established()`, 3 checks on line 2, 3, and 4 exist. The check on line 2 requires the incoming packet to have either ACK or RST bit set, which is easy to bypass by an attacker. The checks on line 3 and 4 call into `tcp_validate_incoming()` and `tcp_ack()`, and can be passed only if the former returns true, and the later returns a non-negative value. In `tcp_validate_incoming()`, to return true, the `seq` field of the incoming packet needs to fall into the receive window [`tp->rcv_nxt`, `tp->rcv_nxt + tp->win1`], and the size of this window is usually between 2^{14} to 2^{20}. `tp->rcv_nxt` is a protocol state unknown to an off-path attacker, thus it takes up to 2^{18} guesses to pass the check. In addition, for `tcp_ack()` to return a non-negative value, `ack_seq` needs to fall into [`tp->snd_una - tp->win2`, `tp->snd_nxt`]. Like `rcv_nxt`, `snd_una` and `snd_nxt` are also protocol states unknown to the attacker, making this check also hard to pass. Combined with the check in `tcp_validate_incoming()`, it takes up to $2^{36} = 68,719,476,736$ guesses for a single packet to be accepted, making it practically unexploitable. Therefore, these are important checks to prevent off-path attackers. In this work, we use the number of packets needed for one injection as the metric for evaluating *off-path packet injection robustness* of a protocol implementation, denoted by N_{pkt}.

We note that the robustness strongly depends on the implementation details. As shown in the bottom-left rectangle of Fig. 3.2, before Linux 3.7, the ACK bit check was much weaker. In this case, off-path attacker can simply set the ACK bit to 0 to avoid the checks in `tcp_ack()`, resulting in a large reduction in N_{pkt} from 2^{36} to 2^{18}. This turns out to be a missing implementation of a check required by the protocol specification [191]. Thus, *even for a well-designed protocol, the corresponding implementation of it may not be robust against off-path packet injection attacks*.

If strong checks do exist, which usually depend on secret protocol states unknown to the attacker, a further question is whether with the help of a collaborative attacker, **these protocol states can be leaked**. This is of concern since previous work [251, 204] has shown that

`rcv_nxt` and `snd_nxt` can have leakage through storage channels such as proc files. The threat demonstrated by Qian et al. [251] is especially realistic as `rcv_nxt` and `snd_nxt` can be inferred under only a second. The upper-right rectangle in Fig. 3.2 illustrates this reported leakage for `rcv_nxt`. Since it is very unlikely to pass the check on line 1 in `tcp_validate_incoming()`, the attack packet reaches `tcp_send_dupack()`, and if `seq` set by the attacker is smaller than `rcv_nxt`, it changes a counter *DelayedACK-Lost* in proc file, otherwise not. If we inspect this counter closely, each comparison leaks 1 bit of information, and thus at most 32 guesses/packets are needed to infer the exact value of `rcv_nxt`. Note that at the time of Qian et al. [251], the check in `tcp_ack()` is easy to bypass. In the current version, the check in `tcp_ack()` is strengthened as shown in the bottom-right rectangle in Fig. 3.2, and even if `rcv_nxt` is guessed, the code still does not have exploitable vulnerabilities for packet injection. However, from our automated vulnerability detection shown later in §3.7, we discover 14 new highly-exploitable leaks for `snd_nxt/snd_una` even after the fix. Thus, *even for well-implemented protocols with strong checks, the protocol states of these checks can still be leaked through attacker-accessible channels, rendering the checks ineffective*.

To systematically discover such vulnerabilities, we argue that automated analysis is required to ensure correctness and coverage, given that the implementations are rather complex — 64 different paths with more than 300 direct and 600 indirect checks are found before accepting an incoming packet in Linux kernel 3.15.8.

3.3.2 Attacker-controlled Implicit Information Leaks

In the above example, the leakage of protocol state `rcv_nxt` is one case of implicit information leaks as the secret is leaked through control dependency (predicates on line 2 in `tcp_send_dupack()`). Compared to classic implicit information leaks, this instance is quite special in that it involves attacker-controlled data in the predicates (`skb->seq` in the example), giving an attacker the ability to influence the control flow. We name this special

Implicit information leak category	Exploit-ability	Example of exploits and related work		
		Exploit case	Attack	Detection/defense
Classic	Low	N/A	N/A	N/A
Strict control dependency (SCD) based [161]	High	Crypto key extraction	[209, 287, 282, 228]	[188, 288, 274, 255]
		Side-channel leaks in web/Android apps	[178, 174, 291]	[284, 170, 232]
Attacker-controlled	High	Packet injection attack	[250, 251, 204]	***This work***

Table 3.1: Categorization of implicit information leaks and position of the work in §III.

type of leaks *attacker-controlled implicit information leaks*, a new concept proposed in this work. As shown in the illustrative example, since attacker-controlled data is involved, an attacker can use different input to actively trigger leaks from the same predicate multiple times and thus extract the secret bit by bit, making it highly-exploitable in practice.

Table 3.1 shows a categorization of implicit information leaks to help illustrate the position of this new concept and this work. Classic implicit information leaks is from a secret information related predicate (e.g., `if (secret > 100)`) to an information sink (e.g., a public value), which usually just leaks 1 bit of information (e.g., whether `secret` is above 100 or not). Since the leakage volume is extremely low compared to explicit information leaks, and tracking it causes large numbers of false positives [221, 161], detecting classic implicit information leaks is a commonly excluded feature in nearly all taint analysis tools [158, 203, 208, 216].

To enable detection of severe information leaks from implicit flows without causing high volumes of false positives, Bao et al. propose to limit implicit flow tracking to a special type of control dependency called strict control dependency (SCD) [161]. SCD denotes the correlation between an equivalency predicate (e.g., `if (secret == 100)`) and an information sink, thus when the information sink is changed, it directly reveals all bits of the secret, making it much more severe than the classic implicit information leaks. Cryptographic key extraction through cache side channels [209, 287, 282, 228] is one real-world exploit example of SCD-based leaks, which leverages the SCD in bitwise equivalence testing of the secret key in certain cryptographic system implementations such as RSA implementation of GnuPG [287, 282]. Another exploit example is side-channel

leaks in web and Android applications [178, 174, 291], in which network traffic pattern is SCD on user choices in web or Android applications. As shown in Table 3.1, both examples are studied extensively on both attack and defense sides.

Attacker-controlled implicit information leaks is a newly-identified category of highly-exploitable implicit information leaks, and similar to Bao et al. [161], we propose to prioritize this special type of leaks in order to balance the vulnerability detection effectiveness and false positives. This concept is orthogonal to SCD in that attacker-controlled data is involved in the control dependency. The target of this work is to identify exploitable cases of such leaks focusing on off-path packet injection attacks [250, 251, 204], and we are the first to design a taint analysis tool for detecting this type of leaks (detailed in §3.6).

3.4 PacketGuardian Overview

In this section, we first describe the analysis required for detecting packet injection vulnerabilities, and then present a design overview of PacketGuardian which supports this analysis.

3.4.1 Analysis Steps

Following the discussion in §3.3, we break the analysis into two steps: accept path analysis and protocol state leakage analysis.

Step 1: Accept path analysis. For a packet injection, the goal is to pass all checks and reach the program point where the packet is accepted, e.g., `accept_payload()` in Fig. 3.2. In this chapter, we refer to these paths as *accept paths*. For a particular protocol implementation, the off-path packet injection robustness depends on the weakest accept path. Thus, the first analysis step is to find the weak accept paths in the implementation. The output needs to highlight the checks related to attacker-controlled information, e.g., header fields, to help analyze the accept path strength.

Step 2: Protocol state leakage analysis. If all accept paths are all well-protected by

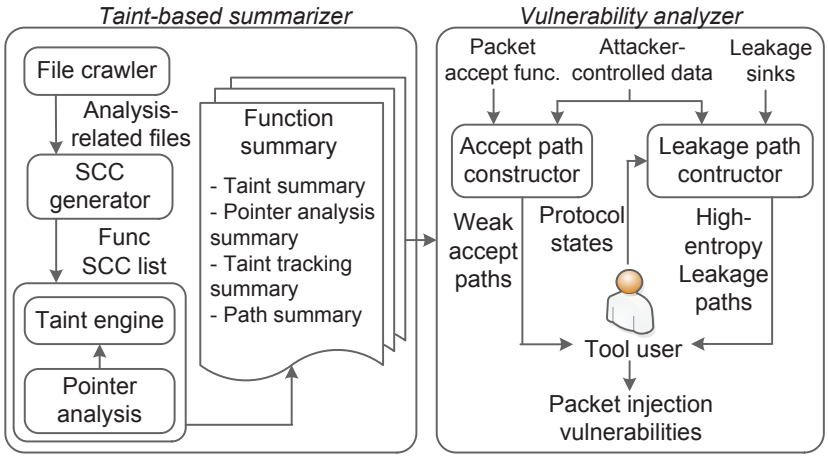

Figure 3.3: PacketGuardian design overview.

"secret" protocol states unknown to the attacker, the implementation can still be vulnerable if these protocol states are vulnerable to information leakage as illustrated in §3.3. Thus, after accept path analysis, we follow up with an information leakage analysis for important protocol states.

The first step is to analyze the strength of the checks related to attacker input on the program path reaching a pre-defined analysis sink, which is similar to the traditional code injection analysis, and thus it can be modeled as a static taint analysis problem with attacker-controlled data as taint source like in previous work [216, 290, 281]. The second step is an information leakage problem and again can be solved by static taint analysis.

Note that symbolic execution is alternative choice, but since it tracks finer-grained information for each variable than taint analysis, it comes with much higher computation overhead, which is unlikely to be efficient and scalable enough in practice, especially in our case high analysis sensitivity are necessary (shown in §3.7.1). Thus, we choose taint analysis in the current design.

3.4.2 PacketGuardian Design

To support the analysis in §3.4.1, PacketGuardian has 2 major components: taint-based summarizer, and vulnerability analyzer, as shown in Fig. 3.3. In this section, we briefly introduce the design of each component, and details are provided in §3.5 and §3.6.

Pre-processing. To support taint analysis, the source code needs to be first pre-processed to the format required by a certain static analysis tool. We choose CIL [242] for our analysis, so for its input requirement, `.c` files are pre-processed to `.i` files in this step.

Taint-based summarizer. With pre-processed source code, given an entry function, taint-based summarizer performs a precise static taint analysis with flow, field, and context sensitivity with pointer analysis. In §3.7.1, we show that such analysis strength is required to discover real vulnerabilities with minimum false positives (FPs). Further, we employ implicit flow tracking (with separate taints from explicit flows), as the protocol logic checks commonly induce leakage through control dependence (see §3.3). Note that implicit tainting is known to generate a large number of FPs [221], and nearly all existing taint analysis tools choose to ignore implicit flows [158, 203, 208, 216]. We show that after prioritizing attacker-controlled implicit information leaks, PacketGuardian does not suffer from the excessive FP problem.

To achieve context sensitivity, our static taint analysis needs to be performed in an inter-procedural data flow analysis framework, with two major choices: IFDS/IDE framework [257, 259], and summary-based (or functional) approach [273]. IFDS/IDE framework performs analysis from function caller to callee, and in the worst case, the analysis complexity is proportional to the number of call graph edges. In contrast, summary-based approach first generates strongly-connected components (SCC) of the call graph and computes function summary from callee to caller. In this approach, each function only needs to be analyzed once and thus has lower complexity and significant performance gains [258]. Its disadvantage is that it needs storage for function summaries, and the callee-to-caller order makes taint path construction unnatural. To support high sensitivity and implicit

flow tracking, our analysis faces a significant scalability challenge if applied to a large code base like the Linux kernel. Fortunately, as demonstrated in previous studies [280], summary-based approach can scale to very large programs.

Following these design choices, as shown in Fig. 3.3, all related source files are first crawled in a breath-first search framework starting from the entry function. After merging these files, function SCCs are computed and serve as input to the taint analysis engine. Taint analysis are then performed in the order of callee to caller, and output function summaries.

Vulnerability analyzer. In vulnerability analyzer, our tool uses the function summaries from the taint-based summarizer to construct paths for accept path analysis and protocol state leakage analysis in §3.4.1. Taking attacker-controlled data as taint source and packet accept functions as sink, *accept path constructor* constructs accept paths with the attacker-controlled data related predicates labeled. The output is further analyzed, with the result being either an obvious packet injection vulnerability, or a set of protocol states that the implementation relies on to prevent injection.

If the accept paths are well-protected by a set of protocol states, *leakage path constructor* performs the second step to find possible leakage of these important states. In this analysis, we also use the function summaries, but the taint sources and sinks become the protocol states and public side channels accessible to the attacker. These channels can be storage side channels [251, 215, 174, 291], public events like sending packets [204], timing, power, etc. Besides detecting leaks, we also construct the leakage paths to help tool users understand and analyze these leaks. In this step, we prioritize attacker-controlled implicit information leaks, as all previously reported highly-exploitable leaks are of this special type [250, 251, 204].

With the choice of summary-based approach, even though the taint sources and sinks are different in the two steps, our tool only needs to perform taint analysis, the most time-consuming part, once instead of multiple times for each source and sink pair. While identifying sources and sinks is a problem for taint analysis in general [256], PacketGuardian

users can conveniently try different sinks in the analysis without re-running the taint analysis.

Manual effort in analysis. In our design, the manual effort mainly lies in identifying protocol states, and the amount of it depends on the number of output paths and predicates. As detailed in §3.6, our design mitigates this problem using path pruning and taint information annotations, which is shown to be effective in §3.7, e.g., our pruning reduces 42.6% paths on average.

3.5 Taint-based Summarizer

In this section, we detail the two core designs of the taint-based summarizer, the taint analysis engine and function summary.

3.5.1 Taint Analysis Engine

In this section, we detail the design of taint environment, propagation logic, and how we support flow, context, and field sensitivity with pointer analysis.

Taint environment. To specify the tainting relationship, each program variable v is associated with a taint environment $\gamma : v \rightarrow T$, where T is a set of taint values $\{t_i | i = 1, ..., k\}$. Each taint value t_i is associated with a variable v_i, meaning that v is tainted by variable v_i. In our design, variables in γ include local, global, formal, and function return variables. Each v is specified by a tuple with its identification information such as variable name and type.

Taint label of explicit and implicit flows. As discussed in §3.4.2, it is a design requirement to include implicit flows, which is known to cause excessive FPs [221]. At the same time, the importance of explicit leaks is much higher than implicit leaks since the former directly leaks the entire data. Thus, to distinguish leaks of different importance and be able to support policies on limiting implicit flow tainting [218], we label each taint value with 2 boolean values d and c, for taint values coming from explicit flows ($d = true$) or implicit

Statement/expression	Taint operation
$Const, Sizeof(typ/str)$	$T_{exp} = \varnothing$
v	$T_{exp} = L_d(\gamma(v))$
$Sizeof(exp_1)$	$T_{exp} = L_d(T_{exp_1})$
$Cast(exp_1)$	$T_{exp} = L_d(T_{exp_1})$
$unop(exp_1)$	$T_{exp} = L_d(T_{exp_1})$
$biop(exp_1, exp_2)$	$T_{exp} = L_d(T_{exp_1}) \cup^l L_d(T_{exp_2})$
$exp : exp_1?exp_2 : exp_3$	$T_{exp} = L_c(T_{exp_1}) \cup^l L_d(T_{exp_2}) \cup^l L_d(T_{exp_3})$
$v = exp$	$\gamma(v) = L_d(T_{exp}) \cup^l L_c(\cup^l\{T_{ct_k}\|ct_k \in CT\})$
$Asm(\{exp_{in_i}\|i\}, \{v_{out_j}\|j\})$	$\gamma(v_{out_j}) = L_d(\cup^l\{T_{exp_{in_i}}\|i)$ $\cup^l L_c(\cup^l\{T_{ct_k}\|ct_k \in CT\})$

Table 3.2: Taint value calculation and propagation logic for intra-procedure propagation. CT includes the constraints that the current statement is control dependent on.

flows ($c = true$). This is a unique design in PacketGuardian and not supported in most existing taint analysis tool [158, 203, 208, 216].

Taint propagation. The tainting process is to propagate taint values by updating $\gamma(.)$ after processing each statement. Table 3.2 shows the taint propagation logic in the statement and expression format defined by CIL [242]. This table only has intra-procedure propagation logic, and inter-procedure logic will be covered later.

In the table, we introduce 3 new operations for taint label management, L_d, L_c, and $\cup^l$. L_d and L_c modify the labels of all taint values in a set with explicit flow and implicit flow label respectively, and $\cup^l$ is simply the set union operation but with label merging, for example if both sets have v but with different labels d_i, c_i and d_j, c_j, the merged taint value label is $(d_i||d_j)$ and $(c_i||c_j)$.

Flow-sensitive tainting with both explicit and implicit flows. Our taint propagation is performed in a data flow analysis framework, where each $stmt_i$ has a taint environment $\gamma_i(.)$, and after tainting according to the rules in Table 3.2, $\gamma_i(.)$ is updated and passed to the egress statements in CFG. Our data flow analysis is a may-taint analysis to tradeoff potentially higher FPs for lower FNs (we have other mechanisms to lower FPs later on). To increase the analysis efficiency, we use topology order to visit CFG nodes.

To support implicit flow tainting, we maintain a constraint path, CT, during the data flow analysis. CT describes the list of conditional branch statements such as `if` exp and

`Switch` exp, which we call constraints (denoted by ct), that the current statement is control dependent on. Each ct is described by a tuple $\{exp, T_{exp}\}$, and adds a new ct after processing a conditional branch statements with exp. We compute the control dependence relationship with a postdominator analysis [273], and delete the ct from CT if the current statement is not control dependent on it. With this constraint list, we compute the implicit flow taint value set by merging T_{exp} of all ct in CT. As shown in Table 3.2, this implicit flow taint is added in taint propagation after applying $L_c(.)$.

Context sensitivity. To support context sensitivity, function call statements need to be correctly handled for inter-procedure taint propagation. According to our design, the taint modifications after calling a callee function can also be described in a function taint environment $\gamma^f(.)$, by merging the return statement taint environments in the callee function using $\cup^l$ operation.

Before being applied, $\gamma^f(.)$ needs to be transformed to the caller function context since $\gamma^f(.)$ is computed in the callee context. This transformation is done in an instantiate function $Inst : v_{callee} \rightarrow v_{caller}$, which replaces the formal parameter variables in callee function with the caller actual parameter variables in the call site of caller function. $Inst(.)$ also handles the side effect in the process for the callee function variables, i.e., caused by changing the values of de-referenced pointer formal or global parameter variables, using a context-sensitive pointer analysis explained later.

Field sensitivity. As shown in the example in §3.3, the header fields related to protocol states in a network protocol are usually implemented as a few fields of a composite type variable. Thus, it will cause large numbers of FPs if we don't distinguish same variable with different fields and taint the whole variable like in some previous tools [203, 114]. We support field sensitivity with the standard technique of expanding each variable with an offset element in the variable tuple. After adding this feature, both the intra- and inter-procedure taint propagation logic need to be updated accordingly.

Adding offset element in variable tuple can also cause $\gamma^f(.)$ to keep increasing with

same variable having different offset due to recursive fields (e.g., `next` in linked list data structure) in a loop. To solve this problem, we add an iteration limit of loops, which is a common practice in field-sensitive data flow analysis.

Taint with pointer analysis. As shown in §3.3, network protocol implementations use pointer extensively, and in our example, the leakage sink is changed with de-referencing a pointer, making pointer analysis a must. In our design, we choose pointer analysis to support referencing and de-referencing pointers when needed during taint propagation. To better work with our taint analysis, our pointer analysis is also flow-, field-, and context-sensitive based on the traditional flow-sensitive pointer analysis framework [214].

With this feature, our analysis has another environment, *pointer environment*, $Ptr : v \rightarrow \{v_i | i = 1, ..., k\}$, meaning that v points to a set of variables $\{v_i | i = 1, ..., k\}$. Like taint environment, we associate each statement $stmt_i$ with a pointer environment for flow-sensitive analysis. In inter-procedure case, the pointer relationship in a callee function is summarized to a a function pointer environment Ptr^{callee}, and $Inst(.)$ is also needed to transform the variables to caller function context accordingly.

Note that parameter aliasing is a classic problem in summarizing points-to relationship, which is typically solved by partial transfer functions (PTF) [277]. In network protocol implementations, pointer parameters typically are used for semantically different purposes, e.g., `tp` for socket status and `skb` for the incoming packet in the illustrative example (§3.3), thus we assume no parameter aliasing in the current implementation. This may introduce inaccuracies, and we plan to implement PTF for improvement in future work.

3.5.2 Function Summary

After taint analysis, the function summary are generated with 4 parts : taint summary, pointer summary, taint tracking summary, and path summary.

Taint and pointer summaries. Taint and pointer summaries are the function taint environment $\gamma^f(.)$ and function pointer environment $Ptr^f(.)$ respectively (detailed in §3.5.1).

After generated, they are fed back into the taint analysis engine for subsequent analysis to support inter-procedure taint and pointer analysis.

Taint tracking summary. As mentioned in §3.4.2, since we choose summary-based approach over IFDS/IDE for scalability, tracking taint propagation becomes unnatural. However, we do need this tracking since it benefits our vulnerability analysis by making the taint result explainable. Thus, we design taint tracking summary to fulfill this goal in PacketGuardian. Note that this summary has another important benefit for our analysis as it can help us locate the indirect constraints of implicit flow taint to obtain a complete accept path and leakage path (detailed later in §3.6).

Like function taint and pointer environments, this summary is specified by a tracking environment $Track : \langle v, t \rangle \rightarrow TR$, where $t = \gamma^f(v)$ and TR is the set of track values. Each track value describes one source for a taint value, which can come from intra-procedure explicit flow, intra-procedure implicit flow, or inter-procedure explicit or implicit flow from a callee function. Since explicit flow is relatively easy to understand, to lower the tracking overhead we only record the source file line numbers of the program point passing the taint. For implicit flows, we create a track value for each ct in CT to make it precise.

For inter-procedure taint tracking, we don't let the track value propagate from callee to caller function like in taint and pointer summaries. Otherwise, the track value set will increase accumulatively at each time of inter-procedure propagation, making the analysis hard to scale. More importantly, in that case each taint tracking summary will have complete taint history for each variable and taint value pair, which is unnecessary since only a few important variables need tracking. Thus, in our design, during function call we only store a "function pointer" in the TR, and delay the actual inter-procedure tracking computation till the vulnerability analysis phase when needed. This "function pointer" is designed to have complete context information to load the callee taint tracking summary and reconstruct the inter-procedure tainting path later.

Path summary. To meet the goal of outputting the accept and leakage paths for explaining

the packet injection vulnerability, during the taint analysis we also summarize the important paths. Like taint tracking summary, recording the inter-procedure program paths is not necessary, and we only record the intra-procedure program paths, and keep a "function pointer".

To satisfy the analysis requirements, the path we record has 2 parts, a constraint path and a path end point. The constraint path is the same as CT mentioned earlier, and here the list of ct is those ones that the path end point is control dependent on. To help explain the path and also enable further tracking of the expression taint, we expand $ct = \{exp, T_{exp}\}$ with 3 elements: variable taint value set $\{\langle v_i, t_i, Track(\langle v_i, t_i\rangle)\rangle | i = 1...k\}$, branch br, and line number, where v_i is a variable used in exp. Variable taint value set gives fine-grained information about the taint values and track values for each variables used in ct, which helps the path pruning and prioritizing detailed later in §3.6. Branch br records whether this path takes the true branch of ct or the false branch of it.

The path end point can be in two forms: a function, or a sink-related statement. The path end point of a function is designed to serve for the role of "function pointer" mentioned earlier, and it can also serve for the vulnerability analysis with a function sink, e.g., `accept_payload()` in Fig. 3.2. The path end point of a sink-related statement is designed to mainly serve for protocol state leakage analysis when this statement is related to a channel accessible to an off-path attacker. For example, this statement can be modifying a public value in storage channels [250, 174], or related to a special instruction in data timing channels (e.g., SSE instructions discussed by Andrysco et. al. [155]), etc. In our current implementation, we focus on storage channels and record the statement changing a global variable, or the de-referenced value of a formal or global parameters since they may point to a global variable depending on the caller context.

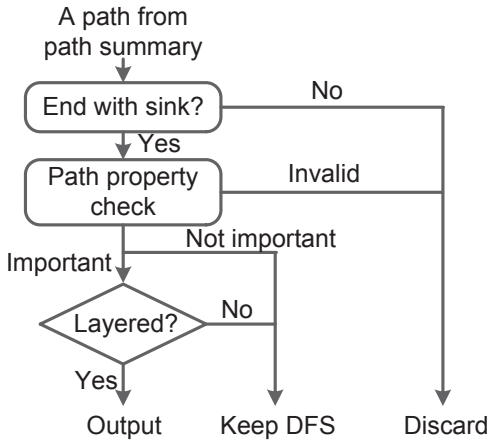

Figure 3.4: Path analysis process in DFS path construction and analysis framework.

3.6 Path Construction and Vulnerability Analysis

In this section, we first introduce a path construction and analysis framework, and then detail accept path analysis and protocol state leakage analysis.

3.6.1 DFS Path Construction and Analysis Framework

The difference between an accept path and a leakage path merely lies in the analysis sink definition and the constraint analyzing and filtering rules that can be applied to reduce FPs. Thus, both analysis can be supported by a general path construction framework following a DFS (depth-first search) paradigm based on the path summary. As mentioned in §3.5.2, each path in a path summary has a constraint path part and a path end point part. Starting from an entry function, the DFS path construction process analyzes the paths in the summary, passes the paths to the callee functions if the path end point is a "function pointer" and continues the DFS process. The process ends when it reaches the analysis sink defined by an analysis task, and output concatenated inter-procedure paths. Like the inter-procedure propagation in taint analysis engine, here we need to use the calling context stored in the "function pointer" and $Inst(.)$ to change the variable context.

Path analysis with implicit flow tracking. In the path construction process, we analyze

each path in the path summary following the procedure shown in Fig. 3.4. We first check whether the path end point is the analysis sink or whether it is a "function pointer" that can call into the analysis sink. If not, this path is unrelated to the analysis task and we discard this path. After that, the property of the path is checked according to the purpose of the analysis task. If its property is considered valid for the analysis, it will be further judged on its importance; otherwise it is discarded. If its property is considered important and the layered analysis mode is on, the path result will be output. Otherwise, the DFS process continues to its callee function. The layered analysis mode will be described later in this section. When reaching the analysis sink, we only output the path if it is considered important.

The path property is determined by analyzing the variables and variable taints of the constraints in the constraint path. These constraints are directly related to the analysis, which we call *direct constraints*. However, besides direct constraints there are also other important constraints that the analysis sink depends on. For example, in Fig. 3.2, the sequence number check on line 2 in `tcp_validate_incoming()` is one of the most important checks preventing off-path packet injection, but it is not the constraint that `accept_payload()` is control dependent on. This dependence is passed through the return value of `tcp_validate_incoming()` to the direct constraint on line 3 in `tcp_rcv_established()`. In order to find these indirect constraints, we use $Track(\langle v, t \rangle)$ in variable taint value set stored in the path summary, and if t includes implicit flow taints, we track its taint path to the indirect constraint that passes these taint values. Based on our taint tracking design, these indirect constraints can be found in an inter-procedure fashion.

Layered path construction. To ensure minimum FNs, the path pruning rules in our accept path and leakage path analysis prefer to be conservative. However, this conservativeness may lead to more FPs, causing heavy analysis overhead. This problem can be quite serious for us since our output is program paths and nested constraint can exponentially increase the

path number. To mitigate this problem, PacketGuardian supports a layered analysis mode, which is included at the bottom of Fig. 3.4. In this mode, when the path is important, we stop the DFS process and output the partial results. With these partial results, tool users can filter out the paths that are not of interest as early as possible, and feed the rest back to the tool to continue the DFS. As shown in our evaluation later in §3.7, this can largely reduce both the number of unimportant output paths and the analysis time. To reduce manual effort, PacketGuardian only stops when the path is considered important as this indicates that some constraints on the path are tightly related to the analysis but it is hard to automatically tell whether they are of interest.

3.6.2 Accept Path Analysis

In accept path analysis, the path is constructed and analyzed with attacker-controlled data and accept functions as input. Attacker-controlled data is usually the function parameters related the incoming packet (e.g., `skb` in Fig. 3.2), and accept functions are functions that indicate the acceptance of the incoming packet, for example copying data to upper layers, or terminating the channel. If it is hard to find such functions, PacketGuardian also supports adding pseudo accept functions to label the analysis sink of interest.

Analysis sink check. In this analysis the analysis sink is a function, so we only consider the paths with end points of functions in path summary. Also, we only care about end point functions that are or may call into the accept functions. Thus, before the analysis, we first create a list of such functions by a DFS crawling process, and then in the analysis sink check discard the paths without an end point function in the list.

Constraint path property check. In the path analysis, each constraint is determined with a property of *protocol state check*, *weak check*, and *strong check*. For a constraint ct, we first check whether it is tainted by attacker-controlled data by looking at T_{exp}, and if not, it is a comparison related to a protocol state and thus labeled as channel state check. If it is tainted through explicit flows, we find out which variable v is attacker-controlled using

the variable taint set in ct, and use exp to understand the comparison this constraint does for v. If it is tainted through implicit flows, the important comparison is done in a indirect constraint and we use the tracking described in §3.6.1 to find it out. We only consider this constraint to be weak check if (1) except v, all other variables are constants, or (2) this constraint requires v to be non-equal to non-constant variables. For the former, an attacker can easily spoof the corresponding packet fields to pass the check, and for the latter, it is very likely that a random value can pass the check. For all other cases, we conservatively label the constraint as strong check to avoid FNs.

In the path construction framework, if the path has a strong check constraint, it is considered important, and otherwise unimportant. A path is considered invalid if it has conflict constraints, e.g., one constraint requires v to be larger than a value while another one requires it to be smaller. In our tool, we use a simple approach to detect this conflict by checking whether two constraints are exactly the same but one has $br = true$ and another has $br = false$.

Weak path candidate output. After the DFS path construction, all the output paths are valid accept paths. To reduce analysis effort, by default the path output consists of only protocol state check and strong check constraints. We include protocol state checks as it can help understand the channel conditions for an accept path. Note that we filter out the weak check constraints only in the last step so that the user can also configure the tool to show all constraints.

Since the goal is to identify the weakest accept path, we also apply path filtering to filter out stronger paths before the final output. If the constraints of one path is a subset of that of another path, the latter is stronger and will be filtered out.

3.6.3 Leakage Path Analysis

In this analysis, the information sources are the protocol states the strong accept path checks depend on, and the sinks are the channels accessible to an off-path attacker. Based

on our path summary design, our sinks can be a function, a statement, or the paths reaching an important program point. This can support storage channels related to a statement that changes a global value [215, 250, 174], timing channels related to a statement or program path lengths [155, 222], or public events related to a function such as sending a packet [204].

Leakage detection. The taint summary for the entry function is a summarized variable tainting relationship, and we can directly tell whether there is possible storage channel leakage by checking if the storage channel sink variables are tainted through explicit or implicit flows. This is a convenient way to quickly tell the leakage status, but lacks detailed information for understanding the leakage, especially for implicit information leaks. Also, it cannot cover channels except storage channels. Thus, we also use the DFS path construction framework to construct leakage paths in this analysis.

Explicit information leaks are relatively easy to understand and PacketGuardian user can just use the taint propagation line numbers in the tracking summary to analyze the leakage. The user can also use the DFS path construction framework to construct the paths just like the accept path construction in §3.6.2 with a change of the analysis sink. However, in a protocol implementation the protocol states are usually not directly leaked through explicit flows to a storage channel – more common leakage is implicit information leaks as shown in recent vulnerability reports [250, 204].

For implicit information leaks, as discussed in §3.3.2, even though classic ones are generally considered of less value and commonly excluded in taint analysis tool design [158, 203, 208, 216], *attacker-controlled implicit information leaks* proposed by this work are highly-exploitable according to existing vulnerability reports for practical protocol state leakage [250, 204]. Thus, our leakage path analysis targets this special type of leaks, and a very important benefit of this is that this can largely reduce FPs, which is a critical problem for implicit flow analysis [221].

In this following part of this section, we describe how to use the DFS path construction

framework to find leakage paths through attacker-controlled implicit flows.

Analysis sink check. In this analysis, we filter out the paths which cannot reach the leakage sinks we defined. For storage channels, we can use the taint summary to check this, and for function sinks and other statement sinks, a DFS process like in the accept path analysis can be used to label function callees of interest, and discard the path of no interest in the DFS path construction.

Constraint property check. In the path analysis, each constraint is determined with a property of *unrelated*, *valid low entropy*, *invalid low entropy*, and *high entropy*. Since we target attacker-controlled implicit information leaks, the constraint is of interest only if it is tainted by both attacker-controlled data and the information sources. If not, it is labeled as unrelated constraint. If related, we find out the variables tainted by attacker-controlled data v^a and those tainted by the information source v^s respectively in direct and indirect constraints. With exp in the constraint we can figure out the comparison it does, and label the constraint as invalid low entropy if the constraint requires v^a to be equal to v^s, and as valid low entropy if the requirement for v^a is to be non-equal to v^s. For both cases the constraint has low entropy, but for the former it is unlikely to pass this check while for the latter it is very likely. For all other cases, we label the constraint as high entropy.

If the path has an invalid low entropy constraint, it is considered invalid and will be discarded. Otherwise, if it has a high entropy constraint, it is considered important. For all other cases it is considered unimportant. Like in accept path analysis, we also check the constraint conflicts and discard the paths with conflicts. For this analysis, layered analysis mode can be very helpful since it is usually hard to judge the entropy automatically. For example, in the illustrative example the receive and send window ranges are depending on dynamic protocol states and protocol design, making automatic judgement difficult. As shown in our evaluation in §3.7, with tool users filtering out paths with invalid low entropy constraints which are labeled conservatively as high entropy ones, finding practical vulnerabilities can be much more efficient.

Leakage path candidate output. Each leak is categorized by the high entropy constraints and the leakage sink, and by default PacketGuardian does not present unrelated and valid low entropy constraints to the user. PacketGuardian users can also configure the tool to output all constraints for more details. For an output path to break the non-inference property [206] and cause leakage, the same sink cannot be triggered for both true and false branches of a high-entropy constraint under all conditions. To check this, for a leakage path p_1 we first find all paths, say p_2, sharing the same sink with p_1 but takes the opposite branch in the high-entropy constraint ct^{high} in p_1. Then, we check whether all constraints in p_2 excluding ct^{high} are a subset of all other constraints in p_1. If so, p_1 is considered invalid and won't be included in the output.

3.7 Evaluation

Following the design, we implemented the taint-based summarizer and vulnerability analyzer in OCaml with roughly 15K and 2.8K lines of code respectively. In this section, we evaluate the tool's effectiveness, accuracy, efficiency by applying it to 6 real network protocol implementations, covering 4 different network protocols. All experiments are run on a desktop computer with a 2.60GHz 8-core Intel Xeon CPU and 128 GB memory.

Code bases. The first code base we target is TCP in Linux kernel version 3.15.8, and we denote it as *TCP-Kernel*. Different from previous work which reported vulnerabilities in TCP code base by manual inspection [251], our tool performs automated analysis, and outputs not only all existing ones but also 11 new highly-exploitable ones. Besides TCP, we also choose two other famous protocols in the Linux kernel, SCTP and DCCP, denoted as *SCTP-Kernel* and *DCCP-Kernel*. Both of them are transport layer protocols providing reliable message delivery like TCP but having distinct features to support other communication requirements.

Besides transport protocols, we also analyze an application layer protocol, RTP, which is one of the most popular protocol for delivering audio and video over IP networks. We

Code base	Analysis entry function	Func #
TCP-Kernel	tcp_rcv_established()	1730
RTP-oRTP	rtp_process_incoming_packet()	141
RTP-PJSIP	on_rx_rtp()	67
RTP-VLC	rtp_queue()	22
SCTP-Kernel	sctp_sf_eat_data_6_2()	290
	sctp_sf_do_9_1_abort()	277
DCCP-Kernel	dccp_rcv_established()	359

Table 3.3: Statistics for the 6 code bases in our evaluation.

Tool w/o features	TP #	FP #	FN #	Low-entropy #
w/o field	4	501	0	27
w/o implicit flow	0	N/A	4	N/A
w/o pointer analysis	0	N/A	4	N/A
w/o layered	4	0	0	1336
w/ all above	4	0	0 (base line)	14

Table 3.4: Evaluation of accumulative improvement using *rcv_nxt* leakage in TCP-Kernel.

pick 3 different popular libraries, oRTP 0.24.1, PJSIP 2.4, and VLC 2.2.0, all of which implement RTP. In the following sections, we denote them as *RTP-oRTP*, *RTP-PJSIP*, and *RTP-VLC*.

For all 6 cases, the analysis chooses the function handling incoming packets as the entry, which are listed in Table 3.3. The last column shows the number of functions reachable from the entry point, showing the complexity of the code bases.

3.7.1 Tool Effectiveness and Accuracy

Table 3.5 summarizes the vulnerability and accuracy result for all 6 code bases. Column 2 describes the type of accept path defined in the analysis task, which in our experiments we consider 2 types: data and close, which means the analysis sink is to feed data to upper layers and to close the channel respectively. We call them *inject-payload* and *close-channel* accept paths in this section. Column 4–6 show the number of output paths, true positive (TP) number and false positive (FP) number. Here the ground truth is the feasible paths among all accept paths before pruning, and since our design is conservative in path pruning and filtering, we do not have any false negative (FN) cases for all 6 code bases. Column

Code base	Type	Weak path output				Pkt # needed for injection	Pkt # needed for injection w/ protocol state leakage	Protocol states that the strong checks rely on
		Path # w/o prune	Path #	TP #	FP #			
TCP-Kernel	Data	64	9	9	0	$(\frac{2^{32}}{win1} \times \frac{2^{32}}{win2})$*	$(32+32)$	rcv_nxt, snd_nxt, snd_una
	Close	40	1	1	0	2^{32}	32	rcv_nxt
RTP-oRTP	Data	21	15	10	5	51 *	N/A	N/A
RTP-PJSIP	Data	1	1	1	0	3	N/A	N/A
RTP-VLC	Data	32	8	4	4	2^{32} *	2^{32}	ssrc
SCTP-Kernel	Data	12	5	4	1	$2^{32} \times \frac{2^{32}}{rem\_win}$	$2^{32} + \frac{2^{32}}{rem\_win}$	my_vtag, base_tsn, cumulative_ tsn_ack_point
	Close	5	2	2	0	2^{31}	2^{31}	my_vtag, peer_vtag
DCCP-Kernel	Data, Close	2	1	1	0	$\frac{2^{48}}{seqno\_win}$	$\frac{2^{48}}{seqno\_win}$	dccps_gsr, dccps_swh, dccps_swl

Table 3.5: Summary of vulnerability analysis results. Number labeled with "*" indicates that it can be smaller under special channel conditions. $win1$ and $win2$ is usually between 2^{14} to 2^{20}, rem_win is less than 4096 by default, and $seqno\_win$ is 100 during default initialization.

3 shows the path number without the path pruning described in §3.6.2. As shown, our pruning reduces 42.6% output paths on average without introducing FNs. Since this output will be analyzed by an analyst, this pruning greatly reduces human efforts.

Column 7 shows the worst case number of packets needed for one injection after the accept path analysis, which is N_{pkt} defined earlier in §3.3. As shown, the N_{pkt} for 3 Linux kernel code bases is at least 10^7 for either inject-packet or close-channel cases, which are unlikely to be exploitable in practice. Their protections solely rely on a few "secret" protocol states unknown to the off-path attacker, which are listed in the last column.

In contrast, the 3 code bases for RTP protocol show diverse results. RTP-oRTP and RTP-PJSIP only need 51 and 3 packets to achieve injection, which are both easy to exploit in practice. But for RTP-VLC 2^{32} packets are needed, which is rather robust. All 3 code bases claim to follow RTP RFC 3550, but our result indicates that even following the same design, their packet injection robustness can be very different due to implementation

differences.

For the code bases that do not have practical vulnerabilities in accept path analysis, we proceed to the second analysis step — protocol state leakage analysis. The N_{pkt} after leveraging leakage are shown in the column 8. For TCP, both protocol state `rcv_nxt` and `snd_nxt`/`snd_una` have high-entropy leakage, and largely degrade the N_{pkt} to only 64 and 32 for inject-payload and close-channel cases respectively. Leakage for `snd_nxt` and `snd_nxt`/`snd_una` have been reported previously [251] by manual discovery, and it is noteworthy that the `snd_nxt`/`snd_una` leakage has already been strengthened after Linux kernel version 3.8 and thus the vulnerability no longer exists. However, using our tool, we automatically find 4 high-entropy leakage for `rcv_nxt`, including the one reported before and 3 new ones. We validated all of them through experiments and confirm that they are indeed exploitable. For `snd_nxt`/`snd_una`, even after the fix, our tool successfully reports 13 new ones and 7 of them are validated.

For inject-payload case in SCTP-Kernel, a low-entropy leakage of `my_vtag` exists and also greatly reduces N_{pkt} from $2^{32} \times \frac{2^{32}}{rem\_win}$ to $2^{32} + \frac{2^{32}}{rem\_win}$. However, it is still a large number and not exploitable in practice. For RTP-VLC, DCCP-Kernel and close-channel case in SCTP-Kernel, no high-entropy leakage is output and thus their N_{pkt} with leakage remains the same. In §3.7.3, we provide more details on these results.

We further conduct an experiment to understand the effects of our static analysis enhancement. As shown in Table 3.4, we breakdown the accuracy improvement with each analysis enhancement using the `rcv_nxt` leakage analysis in TCP-kernel. The evaluation includes TP, FP, FN, and low-entropy leakage, and due to the difficulty of determining ground truth, we use the result of the tool with all features as baseline to evaluate FN for other cases. The results show that all static analysis enhancements, especially implicit flow tainting tracking, are necessary and play an important role.

3.7.2 Tool Efficiency

Before the taint analysis, the code pre-process is a one-time effort which takes around 8.7 hours for the entire Linux kernel, and only less than a minute for oRTP, PJSIP, and VLC.

For taint-based summarizer, since summarizing the entire Linux kernel is infeasible, we limit the scope of TCP-Kernel, SCTP-Kernel, and DCCP-Kernel to the `net` folder under the self-contained Linux kernel networking subsystem. TCP-kernel takes the longest time of 7.8 hours, which we believe is acceptable considering that the computed summary can be reused later for further analysis. In addition, the time can further improved by analyzing functions in parallel as shown in Saturn [280], which is another advantage of our choice of summary-based approach.

With the function summaries, the accept path and protocol state leakage path analysis are very efficient, and perform these analysis on all code bases is less than 10 seconds. Note that this efficiency also benefits a lot from our layered analysis mode, for example, for `rcv_nxt` leakage analysis in TCP-Kernel, it takes 984.5 seconds in total if not using layered analysis mode.

3.7.3 Result analysis

In this section, we detail the vulnerability analysis results summarized in Table 3.5. Due to the space limit we cannot provide code-level details for all results, and for more details about the experiment setup and vulnerability results, please visit our result website **http://tinyurl.com/PacketInjectionVulnerability** [7].

3.7.3.1 TCP-Kernel

Accept path analysis. Our tool outputs 9 inject-payload accept paths which are all TPs. 6 out of them are in TCP fast path processing. The conditions for entering fast path is shown in Fig. 3.5. On line 1, to match the prediction flag it requires the receiver's exact

Code base	Protocol state	Output #	Validated #	Hard to trigger #	FP #	Low-entropy #
TCP-Kernel	rcv_nxt	18	4	0	0	14
	snd_nxt/una	65	7	6	9	43
SCTP-Kernel	base_tsn, cumulative _tsn_ack_point	3	N/A	N/A	0	3
DCCP-Kernel	dccps_gsr/swh/swl	5	N/A	N/A	1	4

Table 3.6: Protocol state leakage analysis result. Ssrc for RTP-VLC and my/peer_vtag for SCTP-Kernel is not included since our tool does not output any high-entropy leakage for them.

send window size, which is possible to achieve in some cases, e.g., when TCP connection is idle. The hard requirement of falling into fast path is that the sequence number, `seq`, needs to equal to the protocol state `rcv_nxt` on line 2. The other 3 output paths are on the slow path, which correctly implements the latest standard specified in RFC 5961 to defend against off-path attacks. In short, they all require the `seq` to fall in the receive window, and `ack` to fall into another window like shown in Fig. 3.2. Thus, their N_{pkt} is roughly $\frac{2^{32}}{win1} \times \frac{2^{32}}{win2}$.

For channel-close case, our tool outputs 1 path due to the effectiveness of our pruning and it is a TP. This path resets the TCP connection in `tcp_validata_incoming()`, and requires `seq` to be equal to `rcv_nxt`. Thus, its N_{pkt} is 2^{32}. Note this an update as specified in RFC 5961 from the previous TCP implementation where a TCP RST is accepted as long as the `seq` falls in the receive window. This change significantly increases the blind in-window RST attacks.

Protocol state leakage. Both inject-payload and close-channel accept paths are protected by protocol state `rcv_nxt`, so we first use this as the leakage source in our leakage path analysis. In our experiments, we use network statistics output in `netstat`, `snmp` and `sockstat` in `/proc/net/` as storage channel leakage sinks. To find the variables that are output to these sinks for taint analysis, we perform static analysis in file `net/ipv4/proc.c` starting from `proc_create()`, locate the proc file function operation registration and find the target variables in the output function, e.g.,

```
1: If ((tcp_flag_word(th) & TCP_HP_HITS) == tp->pred_flags &&
2:     TCP_SKB_CB(skb)->seq == tp->rcv_nxt &&
3:     !after(TCP_SKB_CB(skb)->ack_seq, tp->snd_nxt) {
     ...
4: }
```

Figure 3.5: Code snippet for conditions of entering TCP fast path.

`netstat_seq_show()` for `netstat`. With these sink variables, we first check taint summary for the entry function, and find that these variables are tainted only by `rcv_nxt` through implicit flow. Then we use these tainted variables as leakage sinks in the leakage path analysis.

The leakage results are summarized in Table 3.6. For `rcv_nxt` our tool outputs 18 leakage candidates, and 4 of them are high-entropy TPs. 14 of them are low-entropy leakage, which are mostly pruned out by layered analysis. Note that since falling into fast path requires `seq` equaling to `rcv_nxt`, all fast path related leakage are filtered out automatically as invalid low-entropy leakage. Among the 4 TPs, one of them is reported by previous work [251] by manual discovery. The other 3 are all new discovery, and one uses the same high-entropy constraint in `tcp_send_dupack()` as the one reported, but has a newly-discovered sink *TCPDSACKOldSent* in `netstat`. For the other 2, the attacker packet also makes the code calling into `tcp_send_dupack()` but with different calling context by deliberately failing the PAWS check, e.g., by using an old time stamp, before the `seq` check (line 2 in Fig. 3.2).

After knowing `rcv_nxt`, the attacker can successfully reset the connection and causing DoS. However, to injection payload, the attacker still lacks the knowledge of `snd_nxt` or `snd_una` to pass the `ack` check. We then run another leakage path analysis with these two values together as leakage sources. Like `rcv_nxt`, the sink variables are only tainted by implicit flow. In this setting, we assume that the attacker already knows the correct `rcv_nxt` using the leaks discovered above. For leakage sinks, we use the same ones as

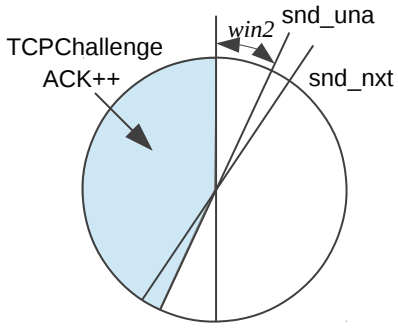

Figure 3.6: Leakage of snd_nxt through sink `TCPChallengeACK`.

those in the `rcv_nxt` analysis.

Since at this time the exact `rcv_nxt` is known, the attacker's packet can exploit leakage vulnerabilities in more program paths including both fast path and slow path area. Our tool outputs 66 leakage candidates for `snd_una`, and 43 of them are low-entropy leakage which are filtered during layered analysis. Among the 9 FPs, 3 cases are caused by requiring packet length to be smaller than data offset field or having an incorrect checksum value, but actually such packets are dropped in `tcp_v4_rcv()` before entering our entry function `tcp_rcv_established()`. Other 4 FP cases requires a fast path protocol state `tcp_header_len` to be greater than 4, but in the implementation it can only be 0 or 4. The last 2 cases are caused by conflicting constraints across procedures, which can be solved by applying more advanced constraint solver such as a SMT solver [185], which we leave as future improvement. The 13 TPs are all new discovery, and 8 are in fast path while 5 are in slow path. All the 8 fast path ones use the comparison between `snd_nxt` and `ack` one line 3 in Fig. 3.5, and after this comparison, there are 8 different sinks in `tcp_rcv_established()`, `tcp_send_ack()`, etc.

The 5 leaks in slow path both goes into `tcp_ack()`, and the high-entropy constraint they use is on line 1 and 2 in `tcp_ack()` of Fig. 3.2. As shown in Fig. 3.6, probability of reaching the `return` on line 3 is $\frac{2^{31}-win2}{2^{32}}$, which leaks around 1 bit of information under the assumption that $win2$ is usually smaller than 2^{20}. In the code base, right before the

`return` on line 3 there is a `tcp_send_challenge_ack()`, in which sink *TCPChallengeACK* is triggered when the challenge ACKs that are already sent is under a threshold set in `/proc/sys/net/ipv4/tcp_challenge_ack_limit`, which is usually around 100.

Validation. We setup a TCP connection between desktop computer A and B, and have another attack computer using raw socket to send attack packets to B to validate these leaks. Computer B is installed with Linux kernel 3.15.8 and added debug information along the program path to validate whether the leakage path is triggered exactly as our tool output, and at the same time monitor the corresponding leakage sinks in A's proc file system. For `rcv_nxt`, we validate all 4 high-entropy leakage. For `snd_nxt`/`snd_una`, 7 out of the 13 cases are validated. The other 6 cases are relatively hard to trigger, for example, 5 of them requires kernel configuration `CONFIG_NET_DMA`, which is only available for processors of certain architecture, e.g., Intel Xscale I/O processors 32/33x.

3.7.3.2 RTP

Since the 3 RTP code bases flow the same network protocol and thus similar to each other in most of the core logic, we cover their results all together in this section. RTP usually doesn't have the option to close the channel with an incoming packet, so our accept path analysis are all inject-payload accept path analysis.

RTP-oRTP. The output for RTP-oRTP has 15 paths, among which 10 are TPs and 5 are FPs. The 5 FPs are all caused by two channel variables having the same meaning, one indicating whether `ssrc` is set, and another indicating whether the first packet is delivered. The changing of two variables is usually correlated and thus they have equivalent values, but our analysis treats them separately, resulting in FP paths with semantically-conflicting constraints. Among the 10 TP cases, 3 requires guessing the correct 32-bit protocol state `ssrc` value, thus N_{pkt} is 2^{32}. However, another 3 TPs indicate that in its logic by default after 50 packets with a new `ssrc` and consecutive sequence numbers, RTP-oRTP will

```
1: matched_ssrc = NULL;
2: for (i=0; i<n; i++) {
3:    if (pkt->ssrc == ssrc) {
4:       matched_ssrc = ssrc;
5:       break;
6:    }
7: }
8: if (matched_ssrc != NULL) {
9: ...
```

Figure 3.7: False positive causes for RTP-VLC accept path analysis.

change the `ssrc` to the new one, making the N_{pkt} reducing to 51. The other 6 TPs are all under very special channel conditions, for example N_{pkt} can be as low as 1 if the attacker precisely captures the moment when `ssrc` is not set yet.

RTP-PJSIP. For RTP-PJSIP, the output only has 1 path and it is a TP. In this path, unlike RTP-oRTP, it changes its protocol state `ssrc` right away if it sees a new one, and relies its robustness solely on the sequence number. According to its logic output by our tool, 2 packets with consecutive sequence numbers will trigger a channel restart, and the 3rd packet's payload will be accepted. Thus, N_{pkt} for RTP-PJSIP is 3.

RTP-VLC. The output for RTP-VLC has 8 paths and 4 of them are TPs. The causes of the 4 FPs are shown in Fig. 3.7. In these paths, it takes both the false branch of `i<n` on line 2 and the true branch on line 8, which is actually not feasible. This is mainly because we construct paths in a flow-sensitive framework and merge the paths from the `break` on line 5 and `i<n` on line 2 when reaching line 8. This can be solved by path-sensitive analysis which has higher precision but also much higher overhead. For the 4 TPs, 2 of them requires the correct `ssrc`, thus their N_{pkt} is 2^{32}. Like RTP-PJSIP, the other 2 TPs change `ssrc` right away, and since RTP-VLC maintains sequence number state separately for each `ssrc`, the N_{pkt} is actually 1. However, changing `ssrc` in RTP-VLC is only when the channel is configured to support more than one `ssrc`, and by default RTP-VLC only supports one. Thus, in normal cases the N_{pkt} is 2^{32} for RTP-VLC.

RTP-VLC protocol state leakage. Among these 3 RTP code bases, only RTP-VLC is hard to inject in default setting due to the protection from the protocol state `ssrc`. In the taint summary of the entry function, 14 variables are tainted by `ssrc`, all through implicit flows. To check the leakage possibility, we set all these 14 variables as leakage sinks in the leakage path analysis but no high-entropy leak is found.

Validation. We build oRTP 0.24.1 and PJSIP in pjproject 2.4, establish audio communication between computer A and B, and read payload in B from application layer APIs. Since proc file `netstat` only shows the local IP address and UDP port for the RTP channel, the attacker computer sends attack RTP packets to B with correct destination IP address and port but different source IP address and port from A's. In the audio data we sent, we include packet number so that we know which packet's payload gets in to the upper layer. We successfully validate that the payload of the 51-st packet for oRTP, and the 3rd packet for PJSIP gets accepted. We also confirm that for VLC without correct `ssrc` the injection cannot succeed.

3.7.3.3 SCTP-Kernel

Accept path analysis. Our tool outputs 5 paths for inject-payload accept path analysis, and 4 are TPs. One SCTP packet can have multiple chunks, and the 1 FP case is because it requires previous chunks from the same packet to have ready been accepted, which is an implementation semantic information that is not known by our tool. One of the TPs has no special channel condition dependence, and it requires (1) it has the correct 32-bit protocol state `my_vtag`, and (2) the sequence number `tsn` falls into a window win starting from a protocol state `base_tsn`, and by default this win is 4096. At the same time, `tsn` also needs to be larger than the previously-received `tsn`, stored in a third protocol state, `cumulative_tsn_ack_point`. We denote the valid tsn range as rem_win, which is win excluding the parts before `cumulative_tsn_ack_point`. Thus, the N_{pkt} is $2^{32} \times \frac{2^{32}}{rem\_win}$. The other 3 TPs all depend on special channel conditions and their N_{pkt} is not

smaller.

For close-channel case, our tool outputs 2 results and both are TPs. One path handles error cause code in the incoming packet, and the other handles packets without error cause code. In both cases, the packet needs to have correct `my_vtag` or `peer_vtag`, which are both 32 bits. Considering the probability that `my_vtag` equals `peer_vtag`, the N_{pkt} is 2^{31}.

Protocol state leakage. The accept paths are protected by `my_vtag`, `base_tsn`, and `cumulative_tsn_ack_point`, so we use them as leak sources. For sinks, we also use storage channel like in TCP-Kernel, and for SCTP we use SNMP statistics in proc file `/proc/net/sctp/snmp`. To get the variables in these sinks, we perform the same static analysis described in §3.7.3.1.

We run the leakage path analysis and find no high-entropy leaks. From the analysis log we find that all leakage paths start with `my_vtag` check, and thus are low-entropy leaks. One of them can be used to tell whether the attack packet has the correct `my_vtag` by looking at sink *SctpInPktDiscards*. This needs 2^{32} packets in the worst case, but it can still be helpful to lower the N_{pkt} from $2^{32} \times \frac{2^{32}}{rem\_win}$ to $2^{32} + \frac{2^{32}}{rem\_win}$.

With the knowledge of `my_vtag`, we still needs to have a `tsn` that can fall into the window specified by `base_tsn` and `cumulative_tsn_ack_point`. We use them as leak sources and find 3 leaks but all are low-entropy ones.

For the close-channel accept paths, the protocol states they rely on are `my_vtag` and `peer_vtag`. In the taint summary, the sinks are also only tainted by implicit flows, but our tool outputs no high-entropy leaks for both of the sources.

3.7.3.4 DCCP-Kernel

Accept path analysis. In DCCP, the checks for copying payload and resetting connection are the same. In this analysis, our tool outputs 1 path and it is a TP. In this path, the DCCP sequence number `seqno` needs to fall into a sequence window $seqno\_win$ around

a protocol state `dccps_gsr` as long as 48 bits, and the higher and lower bounds of this window are another two protocol states `dccps_swh` and `dccps_swl`. Thus, the N_{pkt} is $\frac{2^{48}}{seqno\_win}$. Note that the initial size of this seq_win is only 100, making it impractical to inject. In normal cases there should be another check for the DCCP acknowledge sequence number `ackno`, but as shown in our analysis output, attacker can send a DATA type DCCP message without acknowledge sequence number to avoid that check.

Protocol state leakage. We use all 3 protocol states as sources in this analysis. For sinks, currently DCCP does not create a proc file to store global statistics yet, but it does have a structure for SNMP statistics like TCP-Kernel and SCTP-Kernel, which has same leakage potential if enabled in the future. Thus, we use these variables as leakage sinks. Our tool outputs 5 leaks and 4 of them are TPs. The 1 FP path requires (1) the attack packet is a `SYNC` or `SYNCACK` packet having the right `ackno`, (2) `seqno` is larger than `dccps_swl`, and (3) it fails the $seqno\_win$ check. However, when (1) and (2) happen, `dccps_gsr` is updated with `seqno` and it won't fail the $seqno\_win$ check. In our analysis, we can know that `dccps_gsr` is updated, but cannot be sure that `seqno` can pass the $seqno\_win$ check. The 4 TPs all require `seqno` to fall into $seqno\_win$, and thus are all low-entropy leaks.

3.8 Limitation and Future Work

Possible FNs due to implementation simplification. We design and implement a high precision data flow analysis with implicit flow tainting and pointer analysis to avoid FNs as much as possible. However, there may still be cases causing FNs due to simplified implementation. For example, as described in §3.5.1, we add an iteration limit of loops to avoid adding recursive fields and this may lead to FN cases if the leakage sinks have recursive fields.

Failure to identify semantically-conflicting and low-entropy constraints. As discussed in §3.7.3, the majority of the FPs are caused by conflicting constraints that are tricky to identify. In the future, we plan to use a SMT solver [185] commonly employed by symbolic

execution as tool improvement.

Limited scope of storage channel. As described in §3.6.3, our tool is designed with the capability to cover a range of leakage channels such as storage channels, data timing channels, and public events like sending packets. However, in our experiments we only use storage channels in proc file system as leakage sinks, and may miss practical vulnerabilities leaked through other channels. In the future, we plan to incorporate other sinks in the leakage path analysis.

3.9 Summary

In this chapter, we design and implement an effective and scalable static program analysis tool, PacketGuardian to systematically analyze the security properties of network protocol implementations against off-path packet injection attacks. PacketGuardian uses a context-, flow-, and field-sensitive taint analysis with pointer analysis to achieve high precision, and also targets attacker-controlled implicit information leaks. The solution significantly eases the classic problem of false positives of implicit flow tracking while still yields high detection accuracy of practical exploits. By applying our tool on 6 real network protocol implementations, we are able to discover new and realistic vulnerabilities confirmed by proof-of-concept attacks for both Linux kernel TCP and 2 out of 3 RTP implementations.

CHAPTER IV

Discovery and Systematic Analysis of WPAD Name Collision Attack

4.1 Introduction

Recently, Man in the Middle (MitM) attacks on web browsing have become easier than they have ever been before — the attacker only needs to register one of certain domain names, and web traffic of Internet users from all over the world can be automatically redirected to the attacker's MitM proxy. The underlying vulnerability comes from a problem called "Name Collision" [245]. Name collisions occur when administrators configure their internal systems to use names from local/internal namespaces that are also used in other namespaces (such as the global Domain Name System, DNS), and a collision happens when a query for a name is resolved in an unexpected namespace.

The MitM attack focused upon in this chapter is a name collision based attack that arises from leakage of internal namespace Web Proxy Auto-Discovery (WPAD) queries. These WPAD queries are designed to automatically configure proxies for end systems only from within an administrative domain such as a corporate internal DNS namespace, but only in two of 13 DNS root servers, roughly 20 million such queries are observed to be leaking to the public DNS namespace every day. This has been a known problem for years but remains understudied, mainly because these queries typically use undelegated TLDs as internal Top-

Level Domains (iTLDs) [194, 147, 84], and thus were not exploitable previously. However, in the recently-launched New gTLD (generic Top-Level Domains) Program [146], many of these popular iTLD strings have begun to be delegated and are open for public domain name registration, allowing attackers to exploit these leaked WPAD queries by setting up MitM proxies from anywhere on the Internet with only a domain name registration. Note that this is not a limitation or weakness of new gTLDs per se, but instead a manifestation of a name configuration problem leading to name collisions which we argue should be fully mitigated.

To characterize the magnitude of this newly-exposed MitM threat, we perform the first systematic study of the underlying problem causes and the vulnerability status in the wild. First, we investigate the fundamental underlying cause of WPAD query leaks from internal networks. Using a local testbed and traffic analysis, we find that a major cause that accounts for a significant proportion of the leakage traffic is actually a result of settings on the end user devices. More specifically, we find that under common settings, devices can mistakenly generate internal queries when used outside an internal network (e.g., used at home). From this finding, we identify a set of highly-vulnerable Autonomous Systems (ASes) with both high volume of leaked WPAD queries and high diversity of vulnerable query domain names, which is found to be dominated by home access network ASes.

Second, for these highly-vulnerable ASes, we perform a systematic assessment of the vulnerability status in the wild. Leveraging the insights that most domain names in leaked WPAD queries are transient and low-volume, we propose that a more useful characterization of attack surface should focus on domain names that persistently expose many victims. We call such domain names *highly-vulnerable domains* (HVDs), because an adversary could gain more value from operating them. From this definition, we then design an attack surface quantification method which systematically balances the trade-off between query persistence and high query volume. This allows us to focus on the most exploitable domain names. For example, for the delegated new gTLD `.network`, only 4% of the

domain names in the leaked WPAD queries match the HVD definition.

By applying our attack surface quantification method to the victim ASes, we find that almost all of the leaked queries are for new gTLD domain names defined to have high vulnerability, which indirectly validates our attack surface definition. If these domain names are registered by an attacker, she becomes authoritative to answer all the vulnerable queries, and actual exploits can start at any time. Fortunately, as of September 2015, the registration of these HVDs just started, and our registration status analysis (detailed in §4.6.2) does not find statistical evidence showing that these domains are being maliciously targeted for registration. Nevertheless, we did find seemingly naïve attack registration patterns in the wild, showing potential attack attempts. These results illustrate real MitM threat for Internet users in the wild, and provide a strong and urgent message to deploy proactive protection.

To effectively defend against this attack, remediation strategies can be deployed at the new gTLD registry level to scrutinize the registration of HVDs, and also at the AS level and end user level to prevent the vulnerable queries from being leaked to the public DNS namespace. Based on the insights from the problem cause and vulnerability characterization, we discuss feasible defense methods for each of these three levels, and use empirical data analysis to estimate and compare their effectiveness and deployment difficulties.

We summarize the key contributions as follows:

- Targeting the new MitM attack vector exposed by name collisions, we perform a characterization of the problem and its severity, and an in-depth analysis on the fundamental internal namespace WPAD query leakage problem. From the analysis, we are able to uncover the major leak sources and the underlying device-side causes using both local testbed and DNS root server traffic analysis.

- We present a candidate definition and quantification method for the attack surface of this MitM threat, and use it to systematically study the vulnerability status in the wild. With this, we are able to find a set of highly-vulnerable domains (HVDs) which persistently expose many victims in the wild. We find that over 97% of the leaked WPAD queries are

for these HVDs, and at this point, the HVDs for 10% of the new gTLDs have already been fully registered. These results show a real threat for Internet users in the wild.

- To prevent users from being exploited by this newly-exposed attack vector, based on the insights in our cause analysis and vulnerability quantification, we discuss a set of remediation strategies at the new gTLD registry, AS, and end user levels, and use empirical data analysis to evaluate their effectiveness and deployment challenges.

4.2 The WPAD Service Discovery Protocol

WPAD (Web Proxy Auto-Discovery) is a protocol designed for browsers or operating systems (OSes) to automatically locate a web proxy configuration file. It is primarily used in internal networks where clients are restricted from communicating to the public HTTP network, e.g., in some corporate networks. The proxy configuration file is by default named `wpad.dat`, which is written in proxy auto-config (PAC) format, and specifies the proxy IP and port using code `PROXY` ⟨`IP`⟩`:`⟨`port`⟩.

To find the proxy configuration file, WPAD supports two methods: DHCP WPAD and DNS WPAD. In the implementation, usually DHCP WPAD is attempted first by issuing a `DHCPINFORM` message to the local DHCP server. If the local infrastructure supports this proxy configuration, the PAC file location is included in option 252 in the response.

If no such configuration is found in DHCP, DNS WPAD is performed. Without an explicit configuration like that in DHCP WPAD, DNS WPAD infers the location of the proxy file based on the device domain name. For example, in a company's internal network, a corporate device can be configured with internal domain `company.ntld` in the OS. In DNS WPAD proxy discovery, the proxy file location is inferred from this name and fetched using HTTP request `http://wpad.company.ntld/wpad.dat`, involving a DNS request for `wpad.company.ntld`. To serve this proxy discovery, a company can simply set up a web server with `wpad.dat` under its root directory, and point a DNS record for `wpad.company.ntld` in its local DNS zone file to this server. In this process,

Supported OSes and browsers		Verified versions for DNS WPAD	Enabled by default
Browser	Internet Explorer	6–11	Yes
	Chrome	43	No
	Firefox	12, 33	No
	Safari	8	No
OS	Windows OS	XP, Vista, 7, 8, 8.1, 10	Yes
	Ubuntu	12.04, 14.04	No
	Mac OS X	10.10	No

Table 4.1: Popular OSes and browsers that support WPAD.

all the WPAD DNS queries should be served only by the local DNS resolvers, but as we show later, millions of such queries are leaked to the public DNS namespace every day, causing the name collision problem.

Browser and OS support. WPAD service discovery can be supported in both OS and browser levels. The configuration is typically named "Automatically detect setting" in the LAN proxy setting [145]. Table 4.1 summarizes the popular browsers and OSes supporting WPAD, along with their versions which we have verified using a local testbed. As shown, DNS WPAD is supported by all popular browsers and OSes, and some of them even use it by default, e.g., Windows OSes and Internet Explorer (IE) browsers. Note that for the browsers and OSes that do not enable it by default, the local network administrator, e.g., the IT department in a company, may enable it during the device setup process so that end devices can use its convenient proxy discovery feature. For the browsers tested in our experiments, the discovery process starts right after the browser is launched. With a valid PAC file fetched, all subsequent web traffic is redirected to the configured proxy.

4.3 Threat Model and Attack Surface

In this section, we describe the threat model and attack surface definition of the newly-exposed MitM attack vector, which we call *WPAD name collision attack*.

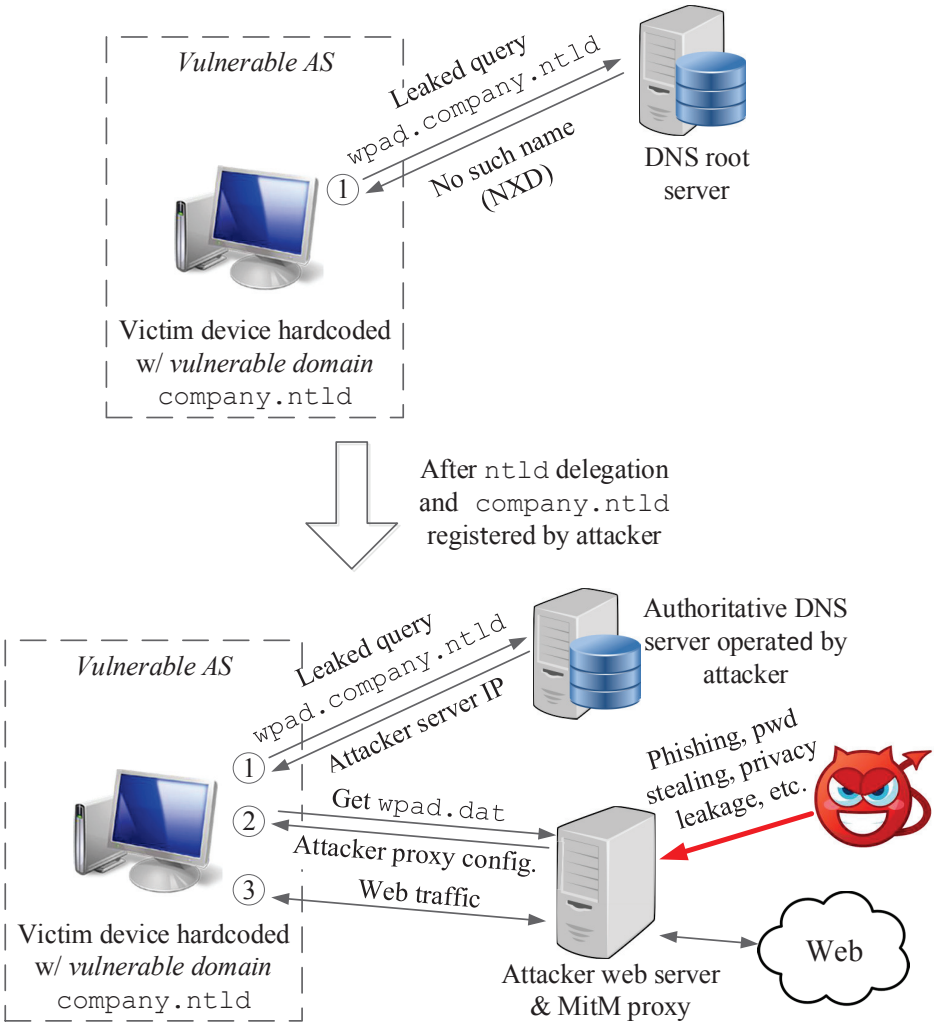

Figure 4.1: Illustration of the WPAD name collision attack. If an internal namespace TLD is delegated as a new gTLD, internal namespace WPAD query leaks can be easily exploited using MitM attack from anywhere on the Internet.

4.3.1 Threat Model

As introduced in the previous section, the WPAD protocol is designed to only configure proxies for end systems from within an administrative domain such as a corporate internal DNS namespace. Ideally, for a device belonging to a corporate domain, it performs discovery to configure a WPAD proxy only inside that domain. While these queries may have always been vulnerable to DNS spoofing attacks, the adversaries would need to be

on-path or be able to spoof DNS responses in a narrow attack window. The intended local scope of queries, the on-path requirement, and the narrow attack window have kept WPAD deceptively safe.

However, because internal queries leak to the DNS root servers and internal namespaces now collide with new gTLD domains, which are both happening in large scale today as characterized later in §4.4.1, the inherent security weaknesses in WPAD are significantly easier to exploit. Fig. 4.1 illustrates the WPAD name collision attack, in which a malicious domain registrant can exploit name collisions of leaked WPAD queries, and launch MitM attacks from anywhere on the Internet. In this attack, victim devices are assumed to be configured to use DNS WPAD for automatic proxy discovery by issuing WPAD queries in an internal DNS namespace, e.g., `company.ntld`. Here, `.ntld` is assumed to be used as iTLD but also delegated in the public DNS namespace. Under some common settings (uncovered in §4.4), such queries are mistakenly leaked out. This allows an attacker to create name collisions for these queries by registering the domain name `company.ntld` in new gTLD `.ntld`. Thus, the leaked WPAD queries from affected systems, which may be anywhere on the Internet are sent to the attacker's authoritative name server and get resolved to fetch the attacker's proxy configuration file. This causes all the subsequent web traffic in the browser or traffic from the entire OS to be redirected to the proxy controlled by the attacker. The victim user may not even recognize the attack, since the WPAD proxy discovery is fully automated at the browser launch time, and some OSes and browsers enable it by default without explicit consent from users (shown in Table 4.1). The attacker can leverage this MitM position to not only eavesdrop sensitive data such as confidential documents and user credentials, but also manipulate the traffic to inject malicious code, launch phishing attacks, or other malicious impacts to vulnerable systems.

In this attack, the adversaries only need to register new gTLD domains to direct potentially vulnerable WPAD queries to them. This means that if a potentially colliding internal domain is registered, the attacker can detect and respond authoritatively to WPAD

queries without the need of spoofing. This frees the on-path requirement and eliminates the narrow attack window drawback of previous WPAD attacks. More importantly, the authoritative nature of the malicious responses makes this attack exploitable despite DNSSEC [157, 156].

This attack is also very stealthy, since once the domain name is registered, due to privacy protection it is difficult for both new gTLD registries and third parties to examine its subdomains for attack attempts. Note that we do not assume that the attacker is fully aware of the set of the vulnerable domains (i.e., domains with leaked queries), and thus deliberately exploits them. The attackers can be sophisticated registrants who know some vulnerable domains based on their own analysis, e.g., by sniffing local network queries or accessing DNS traffic collected by organizations such as DNS-OARC [120]. Meanwhile, the registrants can also be innocent at the domain registration time, but realize and start exploitation after observing a large number of misdirected WPAD queries. Another possibility is that the registrant is completely honest but the DNS servers are compromised by an attacker to exploit these vulnerable queries.

4.3.2 Attack Surface

In order to characterize the magnitude of this newly-exposed MitM threat, we propose a candidate methodology to quantify the WPAD attack surface exposed by registrations of new domain names under new gTLDs. With that, we describe a measure of how exposed (or open) the total attack surface is based on registration status.

Our threat model focuses on the fact that MitM attacks can be launched against any client who issues a WPAD query to a domain name that is controlled by an attacker. Thus, all domain names with leaked queries to the public namespace are vulnerable. However, we find that most of the domains in the leaked query traffic appears infrequently with low query volume, implying that they may not be easily exploited in practice. For example, we find that for the delegated new gTLD `.network`, 42.3% of the domains with leaked queries

(e.g., `company.ntld` in Fig. 4.1) to two of 13 DNS root servers appeared in less than 14 days within a one-year period. Furthermore, less than 4% of these domains account for more than 98% of all leaked WPAD traffic observed at the two DNS root servers. Thus, using all the domains with leaked queries as the attack surface is an overestimate of the actual vulnerability status in practice. Therefore, we define a notion of "highly-vulnerable domains" based on a more accurate and useful attack surface characterization method described as follows.

Attack surface: highly-vulnerable domains (HVDs). In this work, we define highly-vulnerable domains for a new gTLD to be those WPAD query domains persistently exposing a large number of victims. We denote these domains as the attack surface for this new gTLD. These attack surface domains or HVDs need to have two properties: (1) high persistence, meaning that their queries are leaked to the public namespace frequently over a long time period, e.g., every day or days with regular periodicity, and (2) high query volume, indicating that once registered, many victims can be continuously exploited. From this definition, these domains are quantifiably attractive targets for adversaries, and are likely to keep exposing such vulnerability after the delegation of their TLD strings.

This methodology defines a measurably stable set of highly-vulnerable domain names. To quantify the attack surface based on this definition, we first concretely define the level of persistence using period length p and persistence duration n. We then balance the trade-off between persistence and high query volume by systematically exploring p and n, detailed later in §4.5.1. This quantification method allows us to estimate the size and composition of domains that, when registered, constitute the bulk of the WPAD name collision vulnerabilities.

4.3.3 Dataset

We describe the datasets used in our study as follows.

New gTLD list. We obtain the new gTLD list along with their delegation dates directly

from ICANN website [151]. In this work, we consider the new gTLDs delegated before 2015/08/25, consisting of 738 new gTLDs in total.

Root NXD WPAD. Due to the usage of non-delegated iTLDs, the leaked internal namespace queries are captured and replied with NXD by the DNS root servers. Thus, our vulnerability characterization and attack quantification mainly rely on NXD traffic collected at 2 of the 13 root servers — A root and J root, both managed by Verisign. Both root servers utilize IP anycasted services from a globally diverse set of locations [101], which should reduce any significant geographical biases in the data collection. The leaked queries become unobservable in this dataset after the delegation of their TLD strings. Thus, in the analysis of each new gTLD, we only use the data collected before its delegation date.

This dataset was collected internally by Verisign for around 2 years, spanning from September 2013 to July 2015. Since the first new gTLD delegation in the New gTLD Program occurred in October 2013, this dataset covers leaked query traffic for all the new gTLDs delegated so far. To study leaked WPAD queries, we extract the query traffic with query names in the form of `wpad.`⟨`domain name`⟩. Considering that single label domains, e.g., `wpad.ntld` are more easily defended at the new gTLD registries, in this dataset we only include WPAD queries with at least 2 labels in ⟨`domain name`⟩, e.g., `wpad.sld.ntld`, `wpad.3ld.sld.ntld`, etc.

New gTLD zone files and WHOIS data. Once a domain is registered, it appears in the corresponding new gTLD's zone files. Meanwhile, mapping from registered domains to the domain registrants are included in the new gTLD's WHOIS data. To study the registration status and registration pattern of HVDs in our attack surface, we use new gTLDs' zone files from ICANN Centralized Zone Data Service (CZDS) [149] and WHOIS data from BestWhois service [20], which are both pulled daily from 2014/02 to 2015/09.

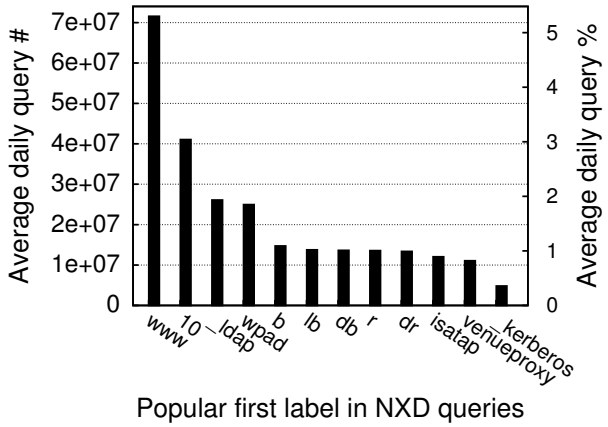

Figure 4.2: The most popular first labels in root NXD traffic.

4.4 WPAD Query Leakage Characterization

The WPAD name collision attack stems from the unintentional leakage of internal WPAD DNS queries into the public DNS namespace. This problem emerged soon after the popularization of the WPAD protocol [245, 265], however remains understudied since it was not easily exploitable until the expansion of the new gTLDs.

To systematically characterize this newly-exposed threat and help find effective solutions, we need to first have an in-depth understanding of this fundamental leakage problem. In this section, we first characterize its severity by quantitative measurements of leaked WPAD query traffic seen in the DNS root servers, and then elucidate the underlying causes of these leaks using query traffic analysis and controlled local testbed experiments.

4.4.1 Quantification of Leaked Queries

Fig. 4.2 shows the popular first labels ranked by their average daily query numbers in NXD traffic at DNS root server A and J from January to July in 2015. In DNS-based protocols, usually the protocol name is the first label. Thus, in the figure many labels belong to popular protocols such as WPAD, ISATAP, etc. The first label query number distribution exhibits a very long tail. As shown, WPAD protocol is ranked top 4 with more than 20 million leaked queries every day, showing high severity in terms of the query

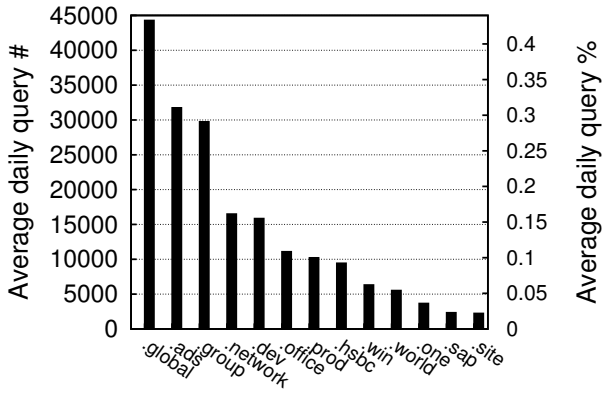

Figure 4.3: The most popular delegated new gTLDs observed in root NXD WPAD queries.

leakage problem. Using the number of distinct IP address and WPAD query domain pairs in our 2-year root NXD WPAD dataset, these queries are estimated to have at least 6.6 million potential victim users in the wild.

For these leaked WPAD queries to be exploitable in our attack, their TLD domains need to be delegated so that the attacker can register the SLD and create name collisions. We study the 738 new gTLDs that have already been delegated before 2015/08/25, and find that 65.7% (485) of them exhibited leaked WPAD queries to the 2 DNS root servers in our dataset before their delegation, revealing a significant attack surface. In §4.5, we use a more systematic approach to quantify the attack surface for these delegated new gTLDs based on the definition in §4.3.2.

To understand the vulnerability exposed by the new gTLDs that have already been delegated today, we measure the daily query percentage of these delegated new gTLD strings in the leaked queries using 1 month of root NXD WPAD data immediately prior to the delegation of the first new gTLD in the New gTLD Program on 2013/10/23. Fig. 4.3 shows the daily query volume and the overall query percentage in root NXD WPAD dataset for delegated new gTLD strings with leaked queries. As shown, even though the query percentage is not high, some top ones such as `.global` already have over 30,000 leaked WPAD queries every day. In total, 2.3% of the daily leaked WPAD queries, which are over

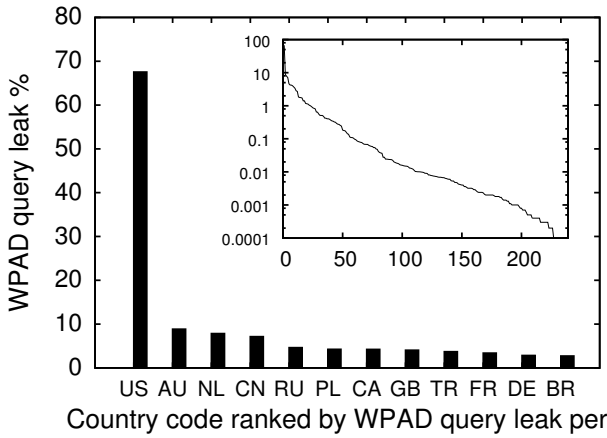

Figure 4.4: Countries ranked by WPAD query leak percentage. The figure inset shows the complete probability distribution, illustrating the long tail.

238,000 queries per day on average from only 2 DNS root servers, belong to the delegated new gTLD set. According to our threat model, these queries are already exploitable today. Note that these are query volumes from just 2 of the 13 DNS root servers. Furthermore, the number will only increase as more new gTLD strings continue to be delegated (as of 2016/03/20, 27.2% (201) more new gTLDs have been delegated since this study was conducted).

4.4.2 Leak Cause Analysis

4.4.2.1 Major Leak Source ASes

To identify the cause, we start by measuring where the leaks originate. We first break down the leaked WPAD traffic into country level according to their query IP addresses. Fig. 4.4 shows the country codes ranked by their average daily leak percentage in our root NXD WPAD dataset from January to July 2015. As shown, U.S. (United States) dominates the leaked traffic with nearly 70% worldwide, and its share is over $6\times$ more than that of the country ranked the second. In the following analysis, our focus is mainly on the leaked query traffic from the U.S.

Within the U.S., we further characterize the query traffic according to ASes. Fig. 4.5

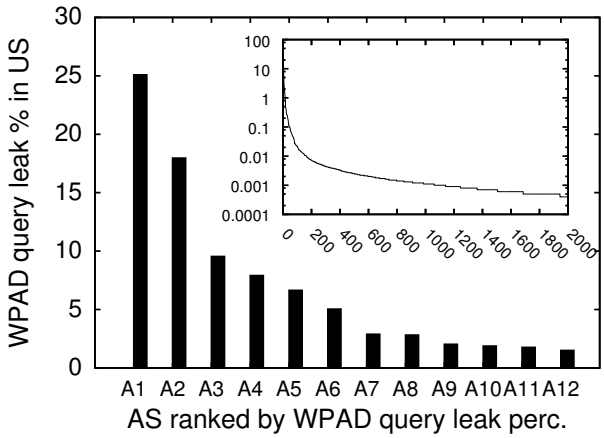

Figure 4.5: ASes ranked by WPAD query leak percentage in US.

shows the ASes with top average daily WPAD query leaks from January to July, 2015. As shown, the overall distribution exhibits a long tail, in which nearly 2000 ASes have leaked queries, but the majority of these queries come from only a few top ASes. The top 12 ASes account for 85% of all the leaks, and their names are listed in Table 4.2. In the table, we denote these ASes A1 to A12 to obfuscate the actual AS in our data. As shown, 10 out of the 12 ASes are home access network ASes. The remaining two ASes both operate open (publicly accessible) DNS resolvers, and we find that the queries come predominantly from source IP addresses within the IP address ranges listed as open DNS resolver servers on their websites. Thus, both ASes are associated with open resolver usage, which is also commonly configured by home access network users. These results suggest the major cause of WPAD query leaks is user behavior at home instead of in corporate networks.

4.4.2.2 Leak Domain Suffixes

To investigate why WPAD queries are leaked from home, we closely examine the domains of leaked WPAD queries in these home access network ASes. Surprisingly, instead of being dominated by a few popular home device domain names as we expected, we found that the leaked queries have on average more than 10,000 different domain suffixes in these 12 ASes. For example, home access network AS A1 originated WPAD queries with more

AS code name	Home access network related
A1	Yes
A2	Yes
A3	Yes
A4	Likely
A5	Yes
A6	Yes
A7	Likely
A8	Yes
A9	Yes
A10	Yes
A11	Yes
A12	Yes

Table 4.2: AS code names (used in Fig. 4.5 and Fig. 4.6) of the top 12 WPAD query leak ASes in the U.S., accounting for 85% of total leak queries. We anonymize the AS names for privacy consideration.

Domain suffix string	Query %	Home network related	Corporate network related
⟨defense contractor⟩ .master.	0.28	No	Unclear
corp.local.	0.26	No	Yes
⟨marketing⟩.local.	0.22	No	Yes
root.local.	0.21	Unclear	Unclear
⟨manufacture⟩.inc.	0.15	No	Yes
⟨town name⟩.local.	0.14	No	Yes
prod.dca.	0.13	No	Yes
⟨consulting⟩.local.	0.13	No	Yes
us.local.	0.13	Unclear	Unclear
⟨real estate⟩.local.	0.12	No	Yes
⟨computer⟩.lan.	0.11	No	Yes
⟨bank⟩.ubc.	0.11	No	Yes
datacenters.ww.	0.11	No	Yes
⟨marketing⟩.intraxa.	0.10	No	Yes
root.corp.	0.09	No	Yes

Table 4.3: Top domain suffixes of the leaked WPAD queries in home access network AS A1. For privacy consideration, we anonymize some company or institution names with their business types in brackets.

than 70,000 different domain suffixes, with the most popular one accounting only for 0.28% of all leaked queries. Moreover, we manually classify the top domain suffixes and find that they are almost all corporate internal network suffixes instead of home device domains. Table 4.3 lists the top 15 leaked query domain suffixes from A1. As before, we obfuscate the details of the domain names for discretion. As shown, none are domains for home

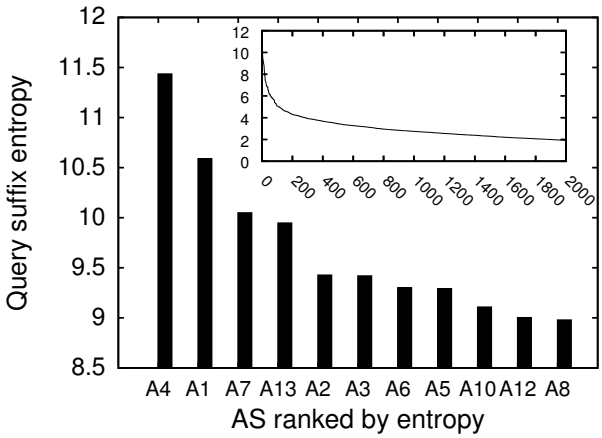

Figure 4.6: ASes ranked by their domain suffix entropy scores. Home access networks with top leak query volume (Table 4.2) are also high-entropy ASes. A13 is the only exception that did not appear in the top 12 WPAD query leak ASes.

devices such as routers. Based on the labels, e.g., "corp", "inc", 12 of them are related to corporate internal networks that are unlikely to be hosted in home networks. This suggests that a potential cause of WPAD query leaks can be attributed to individuals using corporate devices on their home networks.

To further validate this cause, we measure the average daily domain query entropy of each leak source AS. The intuition is that home access networks with end-user machines using different internal network domain suffixes should have higher entropy due to the suffix diversity. In this analysis, we measure the daily query domain suffix entropy using equation $entropy(AS_i) = -\sum_{suf \in S} p_{suf} \ln p_{suf}$, where S is the set of distinct 2-level domain suffixes (e.g., `company.ntld` in Fig. 4.1) appearing in AS AS_i in a day, and p_{suf} is the query percentage of 2-level domain suffix $suf \in S$.

Fig. 4.6 shows the leak source ASes ranked by their average daily domain suffix entropy scores from January to July, 2015. As shown, the home access network ASes with top leak query volume are also high-entropy ASes. Moreover, the top 12 high leak volume ASes are all ranked top 15 in entropy scores out of over 2000 ASes in total, which supports our hypothesis. Thus, the major cause of the WPAD leaks is very likely using devices configured with internal domain names outside of internal networks, e.g., using corporate

laptops at home.

4.4.2.3 Device-side Causes

From this cause, the major problem is on the device side: why does a corporate device still issue corporate internal WPAD queries when the device is actually not in the corporate network? In fact, with the support of DHCP, a device should be able to automatically update domain suffixes when the network changes. To find out the causes, we set up a local testbed to perform controlled experiments.

Experiment setup. We use VirtualBox to launch a virtual machine running different testing OSes on a host machine using NAT (Network Address Translation) configuration. In our experiments, we choose Mac OS X, Ubuntu 12.04, Windows XP, Vista, 7, 8, 8.1, and 10 as testing OSes.

The host machine can be connected to 3 different network environments. Two of them have 2 different domain suffixes configured in DHCP, which are automatically propagated to the host. The third environment does not have a domain suffix, which is created using the tethering feature of a smartphone on a cellular network. In our experiment, we switch the network of the host machine among the 3 environments to simulate network condition changes on the testing OSes, e.g., from corporate to home network.

Results. As summarized in Table 4.4, we find several common OS settings under which internal query leaks can happen even with automatic domain configuration from DHCP. The first case is setting the domain of a computer, which can be found in the control panel of Windows OSes. This configuration is recommended for businesses and schools, since it can remotely manage laptops they provide to their employees and students with their domain controller over VPN or Internet connection [148]. However, we find that once this is set, the OS keeps this domain name regardless of the DHCP domain configuration, and thus still issues internal WPAD queries even after the network has already changed.

The second case is about domain search list configuration, which can be accessed in the

OS configuration	Affected OSes
Set Windows PC domain	Windows XP/Vista/7/8/8.1/10
Hardcode domain search list	Windows XP/Vista/7/8/8.1/10, Mac OS X, Ubuntu 12.04
Change from a network with domain to a network without domain	Windows Vista

Table 4.4: Common OS configurations that can cause a device to mistakenly issue internal queries when the device is used outside internal network.

network setting panels of all OSes we tested. When a queried name is not considered fully-qualified [239], e.g., a dotless single label like `wpad`, the OS appends the domains in this search list one by one until obtaining a valid response. This search list can enable the OS to support both home network and corporate network by including both of their network domain suffixes. But if the corporate network domain suffixes are listed first, internal queries are tried first and thus leaked when outside the internal network. This cause has been discussed before in the web browsing context [186]; in contrast, in our experiment we study it for the WPAD proxy discovery process.

The third case is specific to Windows Vista, where we find that the domain is not unset when changing from a network with a configured domain to a network without a configured domain. This is likely a specific implementation flaw in Windows Vista, as all other OSes quickly change the domain setting to an empty string under the same condition. Due to this problem, corporate computers with Windows Vista leak internal queries when connected to a network without a configured domain, which can happen both at home and at public networks such as a café.

These results show that there exist common configurations in popular OSes that can mistakenly issue internal WPAD queries when the device is used outside corporate networks, causing internal query leaks. Note that these experiments are not intended to be exhaustive in finding all possible device-side causes, which is a rather difficult task. In fact, these identified causes might just be the tip of the iceberg, and merely patching them may only fix a small portion of the problem.

4.4.3 Result Summary and Highly-vulnerable ASes

Concluding from the characterization results above, we find that millions of vulnerable queries are leaked from internal networks every day, and the cause for the majority of the leaks is on the device side. Under common OS configurations, devices with popular OSes mistakenly keep internal domains even outside internal networks, and thus issue internal namespace WPAD queries. Once these queries are issued outside an internal network, the DNS resolvers have no idea where the local name servers are for these internal domains. Thus, they end up querying the DNS servers in the public namespace.

From our analysis above, we are also able to find 10 ASes with both highest query leak volume and query domain suffix entropy score in the U.S. as shown in Fig. 4.5 and Fig. 4.6. These ASes account for 81.2% of total WPAD query leaks in the U.S., and at the same time expose the largest variety of different victims. Thus, we consider them as the most vulnerable leak sources in our study. In the following sections, we will focus on these 10 ASes, especially the one with highest query leak volume, A1, to perform systematic assessment of the vulnerability status in the wild.

4.5 Attack Surface Quantification

Shown in the previous section, a large number of vulnerable WPAD queries are found in the public DNS namespace, many of which are already exploitable today. In this section we propose a candidate attack surface quantification method derived from the definition in §4.3.2, and evaluate its effectiveness.

4.5.1 Quantification Method

As defined in §4.3.2, the attack surface for a new gTLD is highly-vulnerable SLDs with two properties: (1) high persistence, and (2) high query volume. Because "high" query volume is a relative measure, we use query ratio, qr, as the metric for the high query

volume property. For an SLD set S under a new gTLD $ntld$, we represent query ratio as $qr_{ntld}(S) = \frac{\sum_{sld \in S} Q_{sld.ntld}}{Q_{ntld}}$, where $Q_{sld.ntld}$ and Q_{ntld} are the number of leaked queries with domain $sld\,.\,ntld$, and with new gTLD $ntld$ respectively.

To find highly-vulnerable domains, our method is to first identify domains with high persistence. This is because a domain can be exploited as long as it is queried again for WPAD proxy discovery after domain registration. To quantify the level of persistence for a domain $sld_i\,.\,ntld$, we use period length p and persistence duration n to identify domains with leaked WPAD queries to the DNS root server in every p-day period for at least n days until the delegation of $ntld$. High persistence is reflected by a small p and large n, e.g., the domain has leaked queries every day for at least 1 year before the delegation of $ntld$. We use this as evidence indicating that the leakage may likely occur with some degree of frequency even after the delegation because of high persistence.

For a new gTLD $ntld$, given a certain p and n, we can find a set of SLDs under $ntld$, $S^{p,n}$, that meet this level of persistence in root WPAD NXD dataset, with a corresponding average query ratio value $qr_{ntld}(S^{p,n}) = \frac{\sum_{i=1}^{D} qr^i_{ntld}(S^{p,n})}{D}$. Here, $D = \lfloor \frac{n}{p} \rfloor$ is the number of p-day periods during which WPAD query leaks with domains in $S^{p,n}$ are observed, which we call *persistence period*. In this equation, $qr^i_{ntld}(S^{p,n})$ is the query ratio for the i-th period.

To meet the high query ratio property, we need to find the set $S^{p,n}$ with the highest $qr_{ntld}(S^{p,n})$ under a satisfiable persistence level defined by p and n. This is non-trivial as there are trade-offs between the choices of p, n and the query ratio value. For the period length, the smaller, the more persistent, but with a small p we may lose high query ratio domains with longer appearing periods. And for the persistence duration, the larger, the more persistent, but with a large n we may lose some recent high query ratio domains.

Fig. 4.7 and Fig. 4.8 show examples of these trade-offs using 6 delegated new gTLD strings with the highest leaked query percentage (according to Fig. 4.3). As shown, when p increases, the increase of query ratio slows down, and when n increases, the decrease of

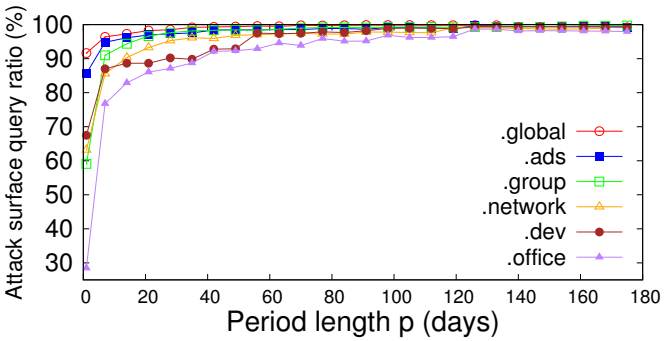

Figure 4.7: Relationship of attack surface query ratio and period length p.

query ratio starts to drop more sharply. Thus, to balance the trade off, for a period length p, we stop increasing it to avoid sacrificing the persistence level, once the increase rate of qr reaches a limit, thr_p, indicating that we have already included enough high query ratio domains. For the persistence duration n, we also set such a limit, thr_n, and stop increasing persistence level once the decrease rate of qr exceeds this limit, indicating more sacrifice in the high query ratio property.

Algorithm 1 shows the pseudocode of our quantification method. For p, our method first tries $p = 1$, and then tries multiples of 7 days considering the weekly pattern of DNS queries, i.e., $p = 7(j-1)$ where $j = 2, 3, ...$ This process stops when $\triangle qr_{ntld}(S^{p,n})$ is less than thr_p, or $\lfloor \frac{n}{p} \rfloor$ is less than 2, which reaches the point of no periodicity. For n, our method tries multiples of 91 days, i.e., $N = 91i$ where $i = 1, 2, 3, ...$, until $\triangle qr_{ntld}(S^{p,n})$ is larger than $thres_n$, or the n is so large that it exhausts our 2-year dataset. We choose 91 days because it is roughly 3 months, which is considered the least persistence duration in this work to avoid short-term domain query phenomena.

4.5.2 Evaluation

We implemented our attack surface quantification method, and applied to the 10 highly-vulnerable ASes using the root NXD WPAD dataset. In this section we use A1 as an example to show our results, because it was the top AS in both query leak volume and domain suffix entropy score, and the findings below also apply to the other 9 highly-vulnerable

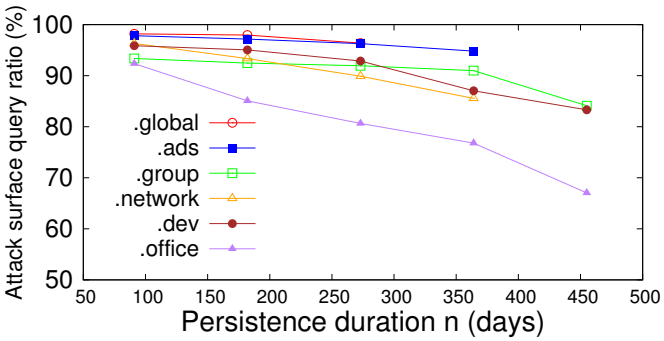

Figure 4.8: Relationship of attack surface query ratio and persistence duration n. Since the 6 new gTLDs have different delegation dates, the data range for the curves are different.

ASes.

In total, A1 presented queries in 255 out of the 738 new gTLDs delegated as of 2015/08/25. Among them, 19 new gTLDs only have leaked query data for 1 day, which are not enough to conclude their attack surface according to our definition of persistence. For the remaining 236 new gTLDs, our method is able to find attack surface domains for 204 (86.4%) of them, which are the ones accounting for 99.99% of total new gTLD WPAD query leaks in this AS.

Fig. 4.9 shows CDF of attack surface query ratio qr_{ntld}, for the 204 new gTLDs in TLD percentage and leaked WPAD query traffic percentage. As shown, for 185 (90.7%) of them, the attack surface query ratio qr output by our method are over 92.1%. These 185 new gTLDs account for 98.4% of total new gTLD WPAD query leaks in A1, showing that we are able to find domains meeting high query ratio property for new gTLDs that expose most vulnerabilities in a victim AS.

We also evaluate how well the attack surface output by our method can meet the high persistence property. As shown in Fig. 4.10, for 148 (72.5%) out of the 204 new gTLDs, which account for 98.8% of total new gTLD WPAD query leaks in this AS, their attack surface domains have periodical appearance for more than 4 periods ($D >= 4$). Thus, our method is also able to find domains meeting high persistence property for new gTLDs exposing most vulnerabilities.

Algorithm 1 Attack surface quantification method

Require: Q_{ntld} (the set of daily leaked WPAD query domains for new gTLD $ntld$ in a victim AS), thr_p, thr_n

Ensure: Attack surface domain set S for new gTLD $ntld$

$n_i = 91i$, where $i = 1, 2, 3, ...$
$p_1 = 1$
$p_j = 7(j-1)$, where $j = 2, 3, 4, ...$
for $i = 1.2, 3, ...$ **do**
    for $j = 1, 2, 3, ...$ **do**
        Find domain set S^{p_j,n_i} from Q_{ntld}
        $d^P_{qr} = qr_{ntld}(S^{p_j,n_i}) - qr_{ntld}(S^{p_{j-1},n_i})$
        if $d^P_{qr} \leqslant thr_p$ or $\lfloor \frac{n_i}{p_{j+1}} \rfloor < 2$ **then**
            break
        end if
    end for
    $q_i = qr_{ntld}(S^{p_j,n_i})$
    $d^N_{qr} = q_{i-1} - q_i$
    if $d^N_{qr} > thr_n$ or $n_{i+1} > |Q_{ntld}|$ **then**
        break
    end if
end forreturn S^{p_j,n_i}

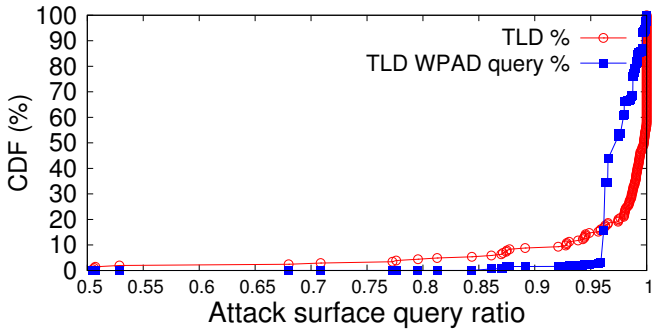

Figure 4.9: CDF of attack surface query ratio in TLD percentage and TLD leaked WPAD query traffic percentage.

4.6 Attack Surface and Exploit Status Characterization

With attack surface successfully computed, in this section we characterize their properties in the victim ASes, and also study their registration and exploit status in the wild.

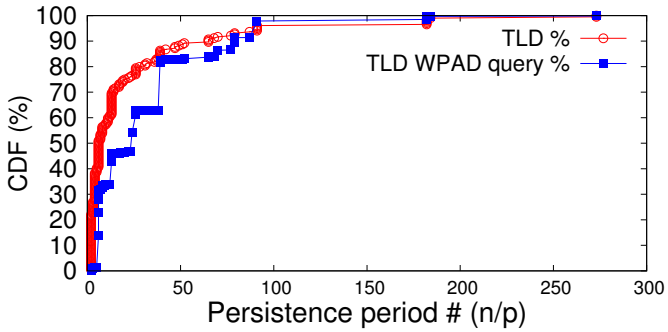

Figure 4.10: CDF of attack surface period number in TLD percentage and TLD leaked WPAD query traffic percentage.

AS code name	Attack surface domain characterization					
	Domain #	Domain query %	Distinct TLD #	# of TLDs have only 1 SLD	Distinct SLD #	# of SLD strings unique to 1 TLD
A1	1185	97.4	204	109 (53.4%)	1122	1080 (96.3%)
A2	486	97.0	122	75 (61.5%)	463	447 (96.5%)
A3	747	97.7	154	91 (59.1%)	714	694 (91.2%)
A4	3621	96.2	331	130 (39.3%)	3324	3145 (94.6%)
A5	704	96.1	146	80 (54.8%)	673	653 (97.0%)
A6	701	97.2	144	75 (52.1%)	668	646 (96.7%)
A7	1751	95.7	230	117 (50.9%)	1633	1566 (95.9%)
A8	457	97.6	113	74 (65.1%)	439	426 (97.0%)
A10	254	96.8	73	42 (57.5%)	235	224 (95.3%)
A12	255	95.5	70	44 (62.9%)	239	227 (95.0%)
Union	8918	97.0	406	92 (22.7%)	7966	7447 (93.5%)
Intersection	90	58.2	33	21 (63.6%)	80	73 (91.3%)

Table 4.5: Attack surface domain characteristics (as of 2015/09/26).

AS code name	Registration status (as of 2015/09/26)		
	Reg. #	# of TLDs w/ reg.	# of TLDs w/ full reg.
A1	129 (10.9%)	56 (27.5%)	18 (8.8%)
A2	49 (10.1%)	28 (23.0%)	10 (8.2%)
A3	68 (9.1%)	34 (22.1%)	16 (10.4%)
A4	284 (7.8%)	79 (23.9%)	8 (2.4%)
A5	67 (9.5%)	35 (24.0%)	15 (10.3%)
A6	66 (9.4%)	31 (21.5%)	9 (6.3%)
A7	123 (7.0%)	55 (23.9%)	17 (7.4%)
A8	43 (9.4%)	27 (23.9%)	12 (10.7%)
A10	28 (11.0%)	17 (23.3%)	8 (11.0%)
A12	33 (12.9%)	19 (27.1%)	14 (20.0%)
Union	589 (6.6%)	123 (30.3%)	16 (3.9%)
Intersection	14 (15.6%)	9 (27.3%)	7 (21.2%)

Table 4.6: Attack surface domain registration status (as of 2015/09/26).

4.6.1 Attack Surface Characterization

Finding 1. Among the 10 top vulnerable victim ASes, ASes operating open resolvers expose the largest attack surfaces. Column 2, 4, and 6 in Table 4.5 show the number of attack surface domains, distinct attack surface TLDs and SLDs for the 10 highly-vulnerable ASes discussed in §4.4. As shown, A4 and A7, which both run open resolvers as discussed in §4.4.2.1, have significantly more attack surface domains, TLDs and SLDs than other victim ASes, even though their leaked WPAD query traffic is much less than some home access network ASes such as A1. This is likely because these popular open resolvers are used in all kinds of network environments and the exposed suffixes are more diverse compared to a single home access network AS (also shown in Fig. 4.6). This suggests that ASes running popular open resolvers should be the first priority for deploying AS-level defense.

Finding 2. In victim ASes, large fractions of leaked WPAD queries are for new gTLD domains defined to have high vulnerability (using our attack surface definition). Column 3 of Table 4.5 lists the percentage of leaked WPAD queries for the attack surface domains during their persistence periods in the 10 highly-vulnerable victim ASes. As shown, for all of these ASes, on average 96.7% of the leaked queries are in the HVDs, showing a high ratio of exploitability in the wild if these domains are registered.

Finding 3. For most of the new gTLDs, only very few SLDs are highly vulnerable. Fig. 4.11 shows the attack surface size distribution for new gTLDs with leaked queries from A1. In the figure, even though some new gTLDs can have very large attack surface, e.g., over 250 for `.office`, 184 (90.2%) of the 204 new gTLDs have fewer than 10 domains in their attack surface. This uneven distribution also holds for other highly-vulnerable victim ASes. As shown in column 5 of Table 4.5, for 9 of the 10 ASes, more than half of the new gTLDs only have one domain in their attack surface. This indicates that for most new gTLD strings, the attack surface size is actually very small, and thus only a few domains need to be treated more carefully in registration.

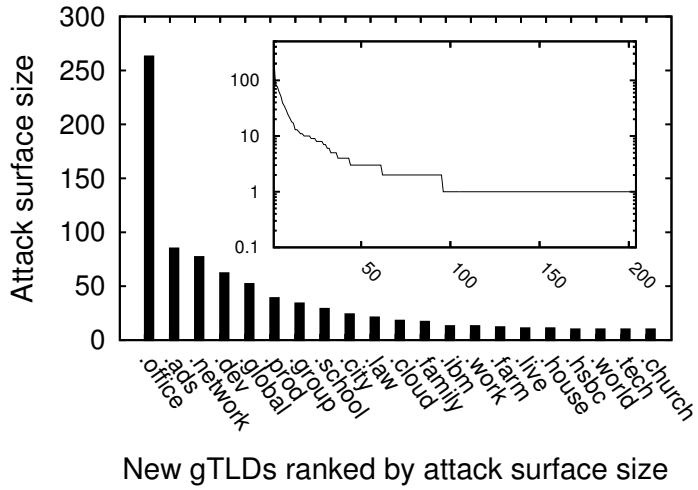

Figure 4.11: Attack surface size distribution for new gTLDs delegated as of 2015/08/25.

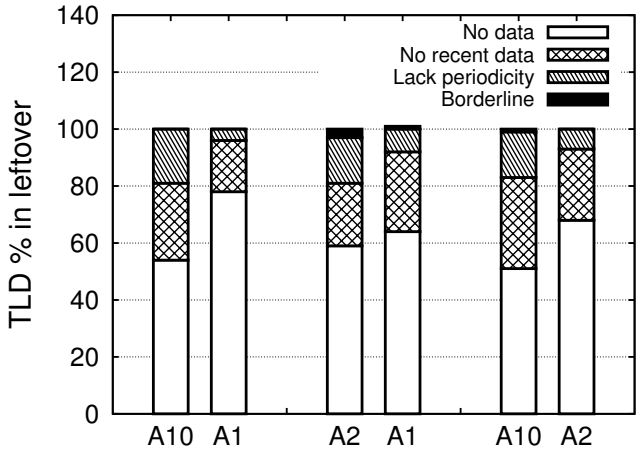

Figure 4.12: Breakdown of new gTLDs in the leftover part in cross AS attack surface comparison.

Finding 4. Most SLD strings only appear in one new gTLD's attack surface. We then measure the popular SLD strings shown across the new gTLD attack surface in A1. From the result, the 5 most popular SLD strings are `us`, `corp`, `local`, `home`, and `net`, which are mostly generic ones. Out of the 204 distinct new gTLD string in A1, we find that the most popular SLD string, `.us`, is only shared by 7 new gTLDs' attack surface. As shown in column 7 of Table 4.5, for all the 10 highly-vulnerable victim ASes, more than 90% SLD strings only appear in one new gTLD's attack surface in the victim AS. This suggests that if applying SLD reservation as a defense strategy, each new gTLD registry

needs to identify its own SLD reservation list based on its WPAD query traffic patterns.

Finding 5. A large portion of the attack surface domains are victim AS unique. As shown in Table 4.5, 8918 domains across 406 new gTLDs are in the union set of the attack surface of the 10 highly-vulnerable victim ASes, but only 90 (1%) of these domains are in the intersection set. Thus, very few attack surface domains are in common among different victim ASes. Moreover, 3689 (41.4%) of these domains are included in only 1 AS's attack surface. These results indicate that most attack surface domains are actually victim AS unique.

To understand why large numbers of domains are AS unique, we pick 3 home access network ASes in the highly-vulnerable AS set, and pair-wisely compare their attack surface domains. More specifically, for comparison between AS A_x's attack surface, S_{A_x}, and AS A_y's attack surface, S_{A_y}, we study the leaked query traffic in A_y for domains in $S_{A_x} - S_{A_y}$ and also leaked query traffic in A_x for domains in $S_{A_y} - S_{A_x}$. We classify the reason why these domains are not left out in the other AS's attack surface into 4 categories: *No data*, *No recent data*, *Lack periodicity*, *Borderline*. Category *No data* means that none of the domains' leaked queries are observed in the other AS in our 2-year root NXD WPAD dataset, and *No recent data* means none of such queries are observed in one month before the delegation of the corresponding new gTLDs. Category *Lack periodicity* means that the domain's queries appear in less than 50% of the days in 3 months before the delegation of the corresponding new gTLDs, which indicates that they are left out due to low persistence according to our attack surface definition. Category *Borderline* means that we could include them in the other AS's attack surface, but we left them out due to the balancing of persistence level and query ratio as discussed in §4.5.1.

The breakdown analysis result of the AS-unique attack surface domains is shown in Fig. 4.12. In the figure, we find that more than 80% of these domains are left out because they have no leaked queries for at least a month before the delegation of the corresponding new gTLDs, which can thus hardly be eligible to be considered as highly vulnerable ac-

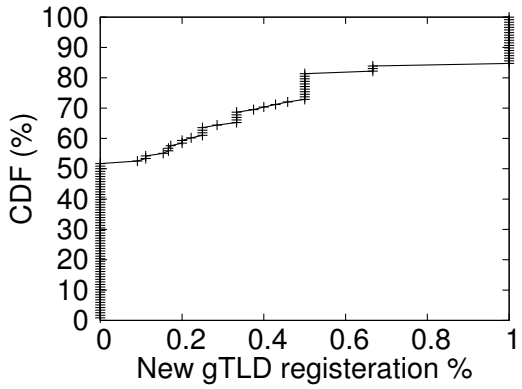

Figure 4.13: Attack surface domain registration percentage for new gTLDs in the top vulnerable AS A1.

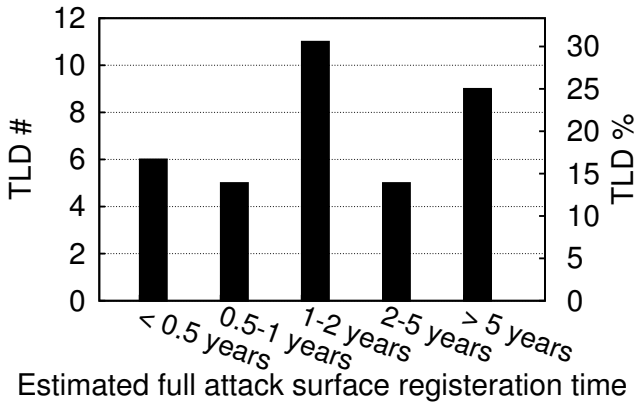

Figure 4.14: Linear fitting results for the estimation time for a new gTLD to have all attack surface domains fully-registered.

cording to our attack surface definition. For the other 20% domains, almost all of them lack periodicity, and only at most 3% of the domains are left out due to the balancing process in our quantification method. Thus, each victim AS indeed has a large portion of HVDs that are unique to it. This suggests that to deploy effective defense at the AS level, each victim AS, especially those highly-vulnerable ones, should customize its own domain filtering list.

4.6.2 Registration Status

Once these HVDs are registered, the actual exploitation can start at any time. Next, we use new gTLD zone files and WHOIS data to characterize the current registration status of these HVDs.

Registrant email	# of total registered domains	# of registered ⟨legacyTLD⟩.⟨new gTLD⟩ domains
⟨email1⟩	19	19 (100%)
⟨email2⟩	7	7 (100%)
⟨email3⟩	2	2 (100%)
⟨email4⟩	16	10 (62.5%)
⟨email5⟩	19	9 (47.4%)
⟨email6⟩	7	3 (42.9%)

Table 4.7: Registration ratio of legacy gTLD string for some registrants, showing potential blind attack attempts. The email addresses are anonymized for privacy reason.

Finding 6. While for some new gTLDs their highly-vulnerable domains have already been fully registered, the overall registration status is still in the early stage. Table 4.6 include statistics of the registered HVDs as of 2015/09/26 for the 10 highly-vulnerable victim ASes, along with the intersection and union sets. As shown, all 10 victim ASes have some of these HVDs registered, but the registration percentages are in the range of 7% to 13%, which is not high. On the TLD level, approximately 22% to 28% of new gTLDs with attack surface in a victim AS have at least 1 attack surface domain already registered. For most victim ASes, around 10% of them have already had all of their attack surface domains registered, indicating that their attack windows are fully open. Recall that once an HVD is registered, the management of the underlying zones is delegated from the new gTLD registries to the domain registrants, and thus the WPAD name collision attack can be set up at any time outside of the new gTLD registries' control. Fortunately, our results show that even though some new gTLDs' attack surface domains in victim ASes have already been fully-registered, the overall registration has just started, and most HVDs are still under new gTLD registries' control.

Finding 7. For majority of the new gTLDs that have not been fully registered yet, the attack window is opening quickly. Besides a current snapshot of the vulnerability status, we also analyze the registration trend of these highly-vulnerable domains. In this analysis, we choose the top vulnerable AS, A1, and estimate how fast the attack surface domains for a new gTLD in this AS will be fully-registered. For most new gTLDs, we find

that generally the total domain registration numbers increase linearly with time after a big increase at the beginning. Thus, we use a basic linear model to fit the attack surface domain registration trend for a new gTLD, and enumerate different starting dates until the average absolute error of the computed registered attack surface domain number is less than 0.5.

Using this method, we estimate the full registration time for the 38 new gTLDs in A1 which have at least one HVD registered (so that the analysis has input) but still not yet fully registered. Among these 38 new gTLDs, 2 new gTLDs' HVD registration numbers do not change in our zone file data set, and thus our method cannot perform linear fitting for them. For the other 36 new gTLDs, our method is able to find a linear curve with less than 0.5 average absolute error for the registered HVD number. In the fitting, 89.4% (272.1 days) of the available zone file data for a new gTLD are used on average. Fig. 4.14 shows the estimation results for these 36 new gTLDs. In the figure, 33% of them are likely to be fully registered in 1 year, and this percentage increases to 60% in 2 years. This is just a rough estimation, but does indicate that even though currently most of the new gTLDs' attack surface domains are not yet fully-registered, their attack surface is being registered quickly, suggesting that immediate precautions need to be applied to prevent these vulnerabilities from further expansion.

Finding 8. We did not find strong evidence of adversaries actively registering attack surface domains, but do observe potential blind attack registrations. Given that many of these highly-vulnerable domains have been registered, we next analyze whether some registrants are aware of these highly-vulnerable domains and thus deliberately register them for the WPAD name collision attacks. In this analysis, we also choose the top vulnerable AS, A1. For each new gTLD in this AS, we use 2 time series data each day: the registered number of attack surface domains, and the registered number of other domains shown in root NXD WPAD data for a new gTLD before delegation. For new gTLDs with some of their attack surface domains registered, we compute the Pearson product-moment correlation coefficient, and find an average correlation score of 0.76, showing a very strong

correlation. This means that it is just as likely to register attack surface domains as other domains appearing in the root NXD WPAD data, suggesting that there are no strong evidence of adversaries actively registering these HVDs.

However, interestingly, we observed registrations that may be used for malicious purposes, such as name collision attacks. More specifically, we find that there are a number of registrants specifically targeting the registration of legacy TLD strings, e.g., `com`, `net`, etc. as SLDs, under new gTLDs. In this analysis, we refer to the strings of TLDs delegated before the new gTLD program as legacy TLD strings, which include gTLDs such as `.com` and country-code TLDs such as `.uk`. We obtain legacy TLD string list by comparing the TLD list on IANA's root zone database webpage [150], and the new gTLD list on ICANN's website [151]. Using the new gTLD WHOIS dataset, we identify a list of registrants having a very high registration ratio of legacy TLD strings under new gTLDs, which is shown in Table 4.7. For example, one registrant with email ⟨`email1`⟩[1] has registered 19 domains as of 2015/09/26, which all contain `com`, `edu`, `gov`, and `org` as SLD strings among over 10 new gTLDs. In our new gTLD WHOIS dataset, only less than 20% of the registrants (identified by email addresses) registered more than 1 domain. Among the 20%, majority of them use corporate email addresses, and the registration targets are usually product related domains, e.g., a registrant with a company email registered 351 domains with a SLD that is the name of their product. The registration behavior in Table 4.7 are very unlikely for brand protection, since (1) they used individual email addresses, and (2) they targeted legacy TLD strings instead of product names, which in combination make such behavior suspicious. One likely reason is that these registrants are trying to exploit one of the earliest reported name collision vulnerability due to an old BIND resolver bug [199]. These results suggest that potential adversaries do exist who are fully aware of the name collision vulnerability. Fortunately at this point, they probably just have not found an effective way of identifying highly-vulnerable domains.

[1] We anonymize the email addresses of the registrants for privacy considerations.

4.6.3 Exploit Status

For the registered HVDs, we are also wondering whether the domain registrants have already started exploiting the vulnerability by serving a valid MitM proxy. Since the domain registrants have full control of the zone after the registration, it is not possible for a 3rd party like us to get an accurate list of subdomains under these HVDs. In our experiment, we use the list of query names in previous WPAD queries to these domains before the delegation of their TLDs as a guess of potential attack subdomains. For each WPAD domain query name $qname$ in the list, we issue request using `wget http://`$qname$`/wpad.dat` and check whether we can get a valid proxy configuration file. Note that even with this list, this experiment can still have false negatives since our probing queries can be intentionally filtered by attackers for only targeted attacks (i.e., only resolve the queries from certain AS, IP, etc.) in order to prevent external detection.

We perform such probing several times for all the domains in the union set of the 10 victim ASes' attack surface domains, but are not able to find valid proxy files. This indicates that the registrants of the highly-vulnerable domains may not realize this attack vector yet, implying that now would be a good time to start deploying remediation strategies, which is discussed in the next section.

4.7 Remediation Strategy Discussion

Considering that the overall vulnerability registration and exploitation are still in the early stage, it presents an opportunity to proactively mitigate this attack. In this section, we discuss the potential remediation strategies by 3 different parties involved in the DNS ecosystem: new gTLD registries, victim ASes, and end users.

Table 4.8 summarizes the results for the estimated effectiveness and deployment difficulties for these remediation strategies. In contrast to the previous sections, which focused on the 10 highly-vulnerable ASes in the U.S., here we consider estimations based on the

Level	Remediation strategy	Effectiveness	Deploy #
New gTLD registry	Scrutinize the registration of the union set of highly-vulnerable domains	97.4%	494
Victim AS	Filter the intersection set of highly-vulnerable domains	36.4%	11305
	Filter AS-specific highly-vulnerable domains	97.4%	
	Filter responses w/ public IP	Not evaluated	
End user	Disable WPAD service (if not used in internal networks)	Not evaluated	$>$ 6.6 million
	Update OS, no hardcoding	$\sim$100.0% (in theory)	
	Filter device-level leaks		

Table 4.8: Effectiveness and deploy number estimation for remediation strategy at new gTLD registry, victim AS, and end user levels. "Not evaluated" means that we cannot evaluate its effectiveness using current dataset.

attack surface quantification using all ASes with leaked WPAD queries in our 2-year root WPAD NXD dataset. This allows us to present more accurate global vulnerability reduction percentages and deployment numbers.

New gTLD registry level remediation. To reduce the chance of an attack, the new gTLD registries, especially the ones we find to have large attack surface (shown in Fig. 4.11), need to ensure that these HVDs are not registered, or treat them more carefully and propose policies to scrutinize their registrations. A naïve approach is to reserve the registrations of all domains seen in NXD traffic. However, according to the experience of deploying the block list in ICANN's Alternate Path to Delegation (APD) [68], merely using 2 days of root NXD data for 3 years, each new gTLD registry needs to block 7449.3 domains on average, and 7 new gTLDs need to block over 100,000 domains. Preventing such a large number of them from being registered, especially those popular ones, is in conflict with the original goal of providing more registration choices, and also hurts new gTLD registries' revenue model. ICANN now changes the policy to allowing their registrations after a 90-day "controlled interruption" period instead of blocking them forever [67].

According to our attack surface characterization, for most of the new gTLDs, relatively few SLD are highly vulnerable to the WPAD name collision attack and need scrutinized registration. For example, for `.network`, 96% of its domains in NXD traffic have very

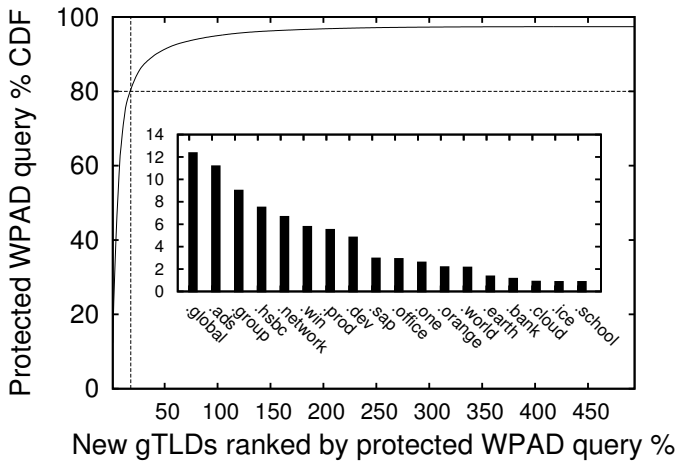

Figure 4.15: Protected leaked WPAD query percentage CDF for partial deployment of new gTLD registry level defense. The figure inset lists the top 18 new gTLDs which can protect 80% of total leaked queries if the defense is deployed.

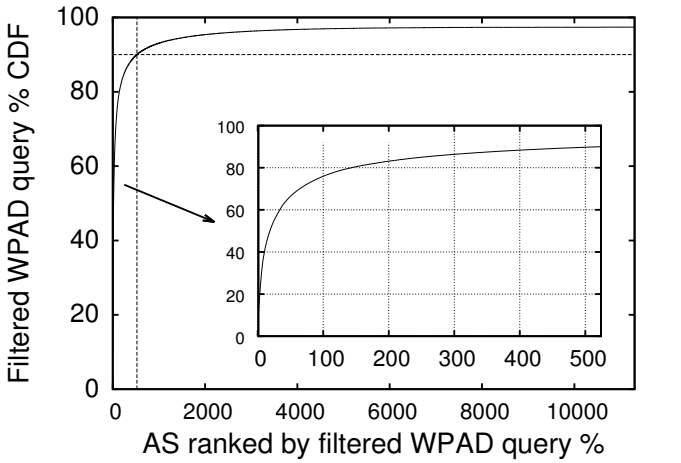

Figure 4.16: Filtered leaked WPAD query percentage CDF for partial deployment of AS level defense. The figure inset shows the CDF for the top 524 ASes.

low volume and/or low persistence of WPAD queries. This is why a general-purpose block list is counterproductive, as opposed to per-SLD and per-TLD analysis performed in this work. Thus, the attack surface defined and quantified in this work offers a cost-effective way of deploying new gTLD registry level domain registration scrutinization. With the attack surface quantification results for all victim ASes, we take the union of the attack surface domains, and find that in total 494 new gTLDs among the 738 ones delegated before 2015/08/25 have HVDs. If all of them have registration scrutinization, 97.4% of the

global leaked WPAD queries in our dataset can be protected. Consistent with our findings in §4.6, most of the new gTLDs have only very few HVDs which need protection – among the 494 new gTLDs, 302 (61.3%) of them have less than 10 HVDs. Thus, for majority of new gTLD registries, this defense can be deployed with very little sacrifice of the business revenue while still being highly effective.

Considering that having all 494 new gTLD registries agreeing on the deployment may be difficult in practice, we also evaluate the effectiveness of a partial deployment. In this analysis, we rank the 494 new gTLDs by the protected leaked WPAD query percentages if they deploy scrutinized registration of HVDs, and the CDF is shown in Fig. 4.15. As shown, deployment at only the top 18 (3.6%) new gTLDs can already protect 80% of the leaked WPAD query globally. Thus, in the deployment, a more feasible and also very effective strategy is to start with the most important 20–40 new gTLDs.

Victim AS level remediation. As shown in §4.4, majority of the leaked WPAD queries come from a few home access network ASes. In addition to new gTLD registry level defense, another direction is to prevent their leaks at the victim AS level. Each victim AS can distribute a black list of vulnerable domains to their DNS resolvers, and filter the queries to these domains before sending them to the public namespace. To create such a list for all ASes, one quick approach is to find the common vulnerable domains using the intersection set of the attack surface domains for the victim ASes. We estimate the effectiveness of this approach using the HVD intersection set for 10 highly-vulnerable ASes, which contains 90 domains as shown in Table 4.5. We find that if all ASes adopt this black list, approximately 36.4% of the leaked WPAD queries globally in our 2-year dataset can be filtered. Thus, even though the creation of the black list is convenient without AS-specific customizations, this approach has limited effectiveness, mainly because many HVDs are AS-specific as characterized in §4.6.

To increase the effectiveness, each victim AS should customize its black lists based on their own query traffic patterns. This can be enabled by DNS traffic monitoring and filtering

in the recently-proposed name collision risk management framework [217]. One candidate approach to create such list is to use the attack surface quantification method proposed in §4.5 based on NXD query data, which can be obtained either by collecting DNS queries on their own, or collaborating with DNS root server operators. The deployment locations are the ASes with HVDs, including 11,305 ASes globally according to our quantification results. If every AS deploys this, it is capable of filtering 97.4% of the leaked WPAD queries globally in our dataset. Compared to the new gTLD registry level defense, this approach can achieve the same level of high effectiveness, but may have higher deployment challenges due to significantly more deployment locations. Thus, we also evaluate partial deployment strategy, shown in Fig. 4.16. In this figure, the X-axis is the 11,305 victim ASes ranked by their leaked WPAD query percentages. As shown, deploying at the top 143 (1.2%) ASes can effectively filter more than 80% of the leaked queries. Thus, similar to new gTLD registry level defense, it is not entirely necessary to cover all 11,305 victim ASes, and targeting the top 1–5% ASes can already achieve a relatively high level of effectiveness.

Victim AS level filtering can also be IP based. In the WPAD discovery process, the leaked WPAD queries are intended to get internal proxy server IP addresses, while in the WPAD name collision attack the attacker needs to return public proxy IP addresses. Thus, victim AS resolvers can prevent the attack by filtering the DNS responses with public IP addresses. The effectiveness of this approach cannot be evaluated using our root NXD dataset, which is left as future work.

End user level remediation. As shown in §4.4, the major cause of the WPAD query leaks is using devices with internal domains outside of the internal network. Thus, to fundamentally solve this problem, this unintended client-side behavior needs to be fixed. If WPAD proxy discovery service is not actually used in the internal network, we suggest that the local network administrator, e.g., the IT department in a company, disable this feature in the supported browsers and OSes (Table 4.1) during corporate device setup process. To

more efficiently enforce this policy without the need of enumerating the configurations of all installed browsers or other related software, the administrator can change OS-level domain name mapping files such as `/etc/hosts` to map all permutations of WPAD URLs within the internal namespace to `127.0.0.1`. In this work, the effectiveness of this approach is not evaluated since it is difficult to measure the amount of leaked WPAD queries belonging to local networks that do not use WPAD service internally.

For the corporate devices depending on WPAD for internal network proxy discovery, the WPAD feature in OSes and browsers still needs to be enabled. To prevent WPAD query leaks for these devices, leveraging our insights of the device-side causes found in §4.4, companies or other entities with internal domains need to stop hardcoding the internal domain search list on their devices. If Windows OS is used, they need to stop setting the Windows PC domain, and also upgrade their OSes. As we mentioned before, these causes may just be the tip of the iceberg, and there might exist plenty of other causes under different conditions. Moreover, considering the large variety of software on the device, new causes, for example domain hardcoding behavior in certain applications, can be created at any point in the future. Thus, these are only short-term solutions and not future proof.

As a long-term solution, we propose to design an OS-level daemon which can filter queries based on the network environment. This daemon is a background process which intercepts DNS queries issued by all applications on the device, and can correctly tell and filter queries with domains not belonging to current network environment. In order to distinguish unintended queries, it tracks the network environment at each network status change, and stores a list of intended domains suffixes for each network environment, either by learning from DHCP configuration messages, or directly being configured by the user. To realize this approach, there are still some design challenges, for example how to accurately tell network environments apart when they use same IP address prefixes, which we leave as future work.

For the short-term and long-term device-side solution above, in theory they can fun-

damentally solve the problem; thus, we consider its maximum effectiveness as 100% in Table 4.8. However, the downside is that it is extremely difficult to reach and apply these solutions to all end user devices, which is estimated to have at least 6.6 million deployment points using the number of distinct $\langle IP, sld.tld \rangle$ pairs in our 2-year root NXD WPAD dataset, where IP is the resolver IP sending WPAD queries, and $sld.tld$ is the WPAD query domain. This is only a lower bound estimation as there might be more than one user device with domain $sld.tld$ behind a resolver, but it is already at least 2 orders of magnitude larger than the new gTLD registry and victim AS level defenses described above.

To help facilitate the deployment process, OSes and browsers can displaying warning messages when detecting potential name collision risks. For example, if the issued WPAD query is leaked to the public namespace, the response will include a special IP address, `127.0.53.53`, during the 90-day "controlled interruption" period [67]. Browsers and OSes can thus leverage this to display risk warnings and recommend the users to consult their IT department immediately to resolve the problem. Note that the 90-day "controlled interruption" period [67] was ineffective to mitigate such issue since the victim machines automatically perform the vulnerable operations even without user awareness [265]. With more support from OS and browser sides, end users can be better notified of the imminent threat to help with the mitigation progress.

To summarize, no single defense approach discussed here can easily solve the problem. To maximize the chance of preventing the attack in practice, the best choice would be using these approaches jointly. Considering the serious and disseminated nature of this vulnerability as shown in this work, actions need to be taken as soon as possible.

4.8 Summary

In this chapter, we perform a systematic study of the underlying problem cause and the vulnerability status for WPAD name collision attack in the new gTLD era. We first characterize the severity of the problem, and uncover that the major cause of the fundamental

leakage problem is very likely devices used in their non-intended network, such as work laptops at home. Then, using a candidate attack surface definition and a quantification method, we systematically assess the vulnerability of the attack in the wild. We find that even though some attack surface domains have already been registered, the overall registration and exploitation status are still in the early stage, indicating that proactive protection strategies are still feasible. Based on these insights, we discuss remediation strategies at the new gTLD registry, AS, and end user levels, and estimate their effectiveness and deployment difficulties. Our work demonstrates the importance of addressing known security vulnerabilities, which might become more exploitable as assumptions change. This work also serves as the first in-depth study of one type of name collision problem in the new gTLD era, hopefully inspiring other follow-up studies.

CHAPTER V

Systematic Analysis and Detection of Client-side Name Collision Vulnerability

5.1 Introduction

With the unprecendented delegation of new generic top-level domains (gTLDs) since late 2013, increasing amounts of leaked internal domain name system (DNS) namespace queries are now resolvable in the public DNS namespace [245]. This has exacerbated a long existing problem, which has been lying fallow, called name collisions, in which a DNS query is resolved in an unintended namespace [245, 69]. One concrete exploit of such problem was recently annouced (US-CERT alert TA16-144A), which specifically targets the leaked WPAD (Web Proxy Auto-Discovery) service discovery queries [172, 127]. In this attack, the attacker simply needs to register a domain that already receives vulnerable internal WPAD query leaks. Since WPAD queries are designed for discovering and automatically configuring web proxy services, exploiting these leaks allows the attacker to set up Man in the Middle (MitM) proxies on end-user devices from anywhere on the Internet.

The cornerstone of this attack exploits the leaked service discovery queries from the internal network services using DNS-based service discovery. With over 600 services registered to support DNS-based service discovery [65], the name collision problem seems

likely to be much broader than the WPAD service alone. However, previous work primarily focus on analyzing and preventing name collisions at the new gTLD registry and the network levels [69, 172, 217, 245], little attention has been paid to understand the vulnerability status and the defense solution space *at the service level*. Since services are the direct victims of name collision attacks, it is necessary to provide service-level solutions so that they can proactively protect themselves. More importantly, since the underlying cause is the domain name resolution in an unintended namespace, compared to defenses at other levels, only the service clients, the actual issuers of the exploited queries, know the intended namespace and thus have the chance to fundamentally solve the problem.

In this chapter, we perform the first systematic study of the robustness of the service client design and implementations under the name collision attack threat model for internal network services using DNS-based service discovery. Our goal is to systematically identify *client-side name collision vulnerability* in the client software, which causes the client to mistakenly accept the identity of a name collision attack server. Our results are expected to serve as a guideline for understanding whether and why a certain client software is vulnerable, as well as providing insights on how to mitigate against this emerging class of attacks. To perform the study, we first measure the services that are exposed to potential name collisions today by analyzing the leaked queries to the delegated new gTLDs. Based on the measurement, we form an exposed service dataset with 80 services with high volumes of service discovery query leaks. Compared to the recent study on the WPAD service [172], our study for the first time uncovers the wide spectrum of services affected by the name collision problem and the potential security implications.

With the set of exposed services, we manually collect their client software, with prioritization for services with higher query leak volumes and clients that are more popular among corporate or end users. In total, we are able to collect 57 client implementations covering 48 exposed services. To systematically perform vulnerability analysis, we develop a dynamic analysis framework capable of analyzing the clients in a simulated name collision

attack environment. The analysis is performed by constructing attack server responses, and a vulnerability is revealed if the client accepts the responses and proceeds with the designed service functionality.

From the vulnerability analysis, our results reveal that nearly all (45) of these 48 services have popular clients vulnerable due to several common software design or implementation choices. We find that the lack of server authentications, which is also exhibited in the WPAD exploit, is the root cause for one third of these vulnerable services. For the remaining two thirds, their clients do use standard server authentications by default, leveraging TLS certificates or pre-shared keys (PSK). However, nearly all clients using TLS certificates are found vulnerable due to the default choice of accepting publicly-valid but previously-unseen certificates from a colliding domain. For the clients using PSK, we find that majority (88.1%) of them are vulnerable since they do not enforce server authentication. We also find a common vulnerable design choice specific to a previously uncovered but popular use of DNS-based service discovery, Zero-configuration networking (Zeroconf) [22], which mixes the service discovery in different namespaces. These results show that even with standard server authentication adopted, the name collision attack threat model still broadly breaks common security assumption in today's internal network service clients. We find that one fundamental cause is the lack of namespace differentiation in the current service discovery and server authentication methods. This problem is newly introduced by the name collision problem and it leaves the clients incapable of handling potential name collisions.

To demonstrate the severity of the discovered vulnerabilities, we construct exploits in our analysis framework and report our findings on a number of new name collision attacks. These attacks are able to induce exploitation to a wide range of popular internal network services, including MitM attacks on the Windows tunneling service, malicious library injection on the Ruby library discovery service, document leakage on the macOS printer discovery service, credential theft on the remote connection services in macOS Terminal,

and phishing attacks on the VoIP service in Linphone and the contacts and calendar services in macOS and iOS. Through these case studies, we demonstrate the high end-to-end exploitability of the identified vulnerabilities in practice.

With increasingly more new gTLDs being delegated, such widespread vulnerabilities in the exposed service clients become more critical than ever and require immediate attention and remediation. Based on the insights from our study, we propose a series of service client software design guidelines, e.g., proposals to enable namespace differentiation in the existing service discovery and server authentication methods. Our proposals complement the previously-proposed DNS ecosystem level solutions [172, 217] and enable the victim services to actively defend against name collision attacks.

In summary, our key contributions are as follows:

- We generalize the WPAD name collision attack to a new class of attacks on the broad set of internal network services using DNS-based service discovery. We perform the first measurement on the exposed services today and characterize their designed functionality and the potential security implications.
- We collect the client implementations for the exposed services and systematically analyze their vulnerability status under name collision attacks leveraging a dynamic analysis framework. Our results show that nearly all the exposed services have popular clients vulnerable due to several common design choices. This suggests that the name collision attack threat model broadly breaks common security assumptions made in the service clients today.
- Based on the analysis results, we construct exploits and report our findings of a myriad of new name collision attacks with severe security implications, including MitM attack, malicious library injection, credential theft, etc. These findings show high end-to-end exploitability of identified vulnerabilities in practice.
- We identify several fundamental vulnerability causes, including a cause newly introduced by the name collision problem, the lack of namespace differentiation. Based on

the insights, we propose a set of service software level solutions, which enables the victim services to actively defend against name collision attacks.

5.2 Client-side Name Collision Vulnerability

In this section, we describe a generalized name collision attack threat model and the vulnerability definition.

5.2.1 Threat Model

As characterized in §4.4, internal DNS namespace queries are observed to be leaked into the public namespace. Among them, as we later characterize in §5.3, are a broad set of internal DNS-based service discovery queries. With the vast expansion of the public namespace via the New gTLD Program, many iTLDs are now delegated and these leaked service discovery, intended only for a internal administrative domain, are now resolvable in the public namespace.

In this work, we consider the attacker to control delegated new gTLD domains with internal query leaks, or *name collision domains*, and provide malicious responses to exploit these leaks. Such attacker may be (1) sophisticated registrants who become aware of name collision domains by analyzing local DNS traffic or DNS traffic from OSINT (open-source intelligence) sources such as DNS-OARC [120], (2) registrants not specifically targeting name collision attacks at the domain registration time, but realize and start exploitation after observing the leaked queries, or (3) miscreants who compromise the DNS servers of the name collision domains, e.g., leveraging software vulnerabilities, to perform exploitation.

Fig. 5.1 illustrates the concept of a generalized name collision attack. Due to the name collision problem, the leaked service discovery queries from a victim service client first reach the attacker's DNS servers. Based on the service name specified in the queries (§2.2.3), the attacker's DNS server points the client to an attacker-controlled server for the service in request. In this step, the attacker controls the domain and thus can provide

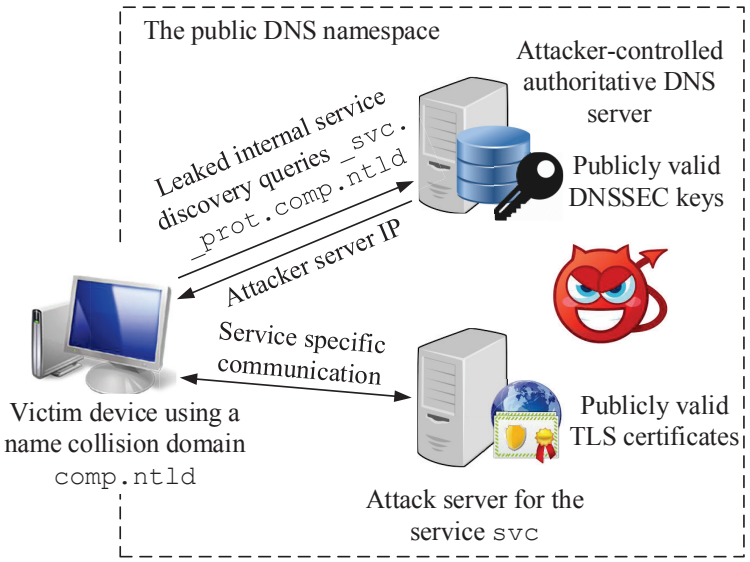

Figure 5.1: The generalized name collision attack threat model.

authoritative responses signed by publicly valid Domain Name System Security Extensions (DNSSEC) keys. Such discovery process may involve multiple rounds of DNS queries depending on the usage scenario of the service protocol, which is characterized later in §5.4.

After the service discovery step, the service client initiates a connection to the attack server. In this step, the client mistakenly accepts the identity of the attack server and proceeds with the intended service functionality. Since the intended server is typically located in an internal network, we do not assume the attacker is capable of relaying the client requests to the intended internal server and perform MitM attacks. Instead, the attacker's goal is to only leverage the server position to induce security breaches. Even though the attacker is not performing a MitM attack on the sevice discovery process, the attacker may still be able to exploit the server position to ultimately perform a MitM attack on the end device as demonstrated by the WPAD name collision attack [172].

To perform the attack, we assume that the attacker can use any resource available in the public namespace. An important example of such resource is a valid TLS certificate for the attack server, which can be obtained freely in a few minutes from authorities such as Let's

Encrypt [75]. Compared to the threat model in the WPAD name collision attack, which only considers one service discovery usage scenario with no modern server authentication components, the threat model here considers a more general form of a name collision attack that applies to a much broader set of internal network services using DNS-based service discovery.

Compared to previous attacks on internal network services, which typically have tight requirements of both the attack placement and timing, name collision attacks are much more severe due to several unique properties. The first is the ease in which they can be launched. Internal network attacks typically require an attack device in the victim's internal network, but name collision attacks only require the registration of certain vulnerable domains. Second, they are of larger scale in terms of victim sources. Compared to the limited internal scope of internal network attacks, name collision attacks affect all leaked queries within the same colliding domain from potential victims all over the world. Third, they are also more powerful, since the attacker can use a number of valid identities in the public namespace, e.g., DNSSEC keys and TLS certificates, that are typically not available for internal network attackers. This class of attacks is also stealthy, since after the domain registration, it is difficult for third parties to further check the subdomains for attack attempts due to privacy consideration [207].

5.2.2 Vulnerability Definition

Under the threat model above, we define a vulnerable internal network service with two properties:

(1) Service query exposure. For a service software to be vulnerable, it needs to (1) use DNS-based service discovery, and (2) have the discovery queries being leaked to the public namespace. In §5.3, we use the leaked query traffic collected at the DNS root servers to measure the services with query leaks, which we call are *exposed* to the name collision problem. In this work, the query leakage volumes are used to quantify the degree of such

exposure.

(2) Client-side name collision vulnerability. With service query exposure, the service client software needs to have vulnerable design or implementations that accept the identity of the attack server from the discovery. In this work, if these vulnerable design or implementations, alone or in combination, cause the client software to pass all server authentication logic if implemented, and reach the execution point of starting the intended service functionality with the attack server, we call the client software to have a *client-side name collision vulnerability*. Since in our threat model the attackers cannot access the legitimate internal server to obtain the right proof of identity, the client should be able to tell the attack server apart. However, our vulnerability analysis results indicate that the server authentication logic in today's service clients is generally not robust enough to correctly handle name collision attacks. Later in §5.4, we detail the analysis results and findings.

5.3 Exposed Service Characterization

In this section, we measure the exposed internal network services (defined in §5.2.2), and characterize their functionality.

5.3.1 Methodology

Leaked query dataset. We perform the leaked DNS query measurement using query traffic collected at DNS root servers in the DNS-OARC Day In The Life of the Internet (DITL) project [8]. The DITL project has collected DNS traffic from participating DNS root servers for 48 hours annually since 2006, which delivers the largest scale simultaneous DNS traffic collection from the global DNS infrastructure [168]. Considering that the dataset has multi-year collections but each collection is limited to two days, our analysis is performed at the granularity of days and aims at identifying the most frequently requested services observed during the collection.

Our analysis uses the 2011 to 2016 query traffic data, which are collected at 10 to 11

Exposed service functionality	Exposed service name	Potential security implications
Proxy/tunnel config.	wpad① (N), isatap② (N), proxy② (N)	MitM attack
Time config.	ntp③	Time shifting attack
Software activation	vlmcs② (N)	DoS
Directory service (help a client locate a server of the requested service)	ns*① (N), alt*① (N), lb① (N), db① (N), dns-sd①, dr① (N), tracker② (N), dns-llq⑤, dns-update⑤	Server spoofing, service info. leakage
Web service	www*① (N), api① (N), static① (N), cf① (N), share① (N), http②, https③	Web-based phishing attack, malicious script execution
Server config. retrieval	stun④	Config. info. spoofing
Multimedia file access	ptp③, dpap④	Phishing attack
Authentication service	kerberos①	DoS
Coding library retrieval	rubygems⑤	Malicious code injection
Database service (organization data, calendar, contacts, etc.)	gc① (N), ldap①, carddav④, ldaps④, caldav④, caldavs④, carddavs④	Phishing attack, organization data leakage
Remote access to computers/file systems	afs3-vlserver④, adisk④, smb④, afpovertcp④, ftp④, sftp-ssh④, rfb④, webdav⑤, odisk⑤, eppc⑤, telnet⑤	Phishing attack, info. leakage
System management	kpasswd②, airport③, servermgr⑤	System config. info leakage
Mail	autodiscover① (N), outlook① (N), mail*① (N), pop3②, smtp②	Email spoofing, phishing
VoIP	sipinternaltls① (N), sip① sipinternal① (N), sipexternal① (N), sips③	Call spoofing, phishing
Messaging	xmpp-server③, xmpp-client③	Msg. spoofing, phishing
Printer	printer③, pdl-datastream③, riousbprint③, ipp③	Internal/personal document leakage
Scanner/camera	scanner③, ica-networking⑤	Phishing attack
Distributed computing	xgrid④	Malicious code execution
System monitoring	syslog⑤	Organization info. leakage

Table 5.1: Functionality characterization of the exposed internal network services and the potential security implications. Circled numbers are the ranges of the average daily query leak volumes: ① > 100,000, ② 10,000 – 100,000, ③ 1,000 – 10,000, ④ 100 – 1,000, ⑤ 10 – 100. N denotes non-registered service. Documentations for individual services are in Table 5.2 and Table 5.3.

out of the total 13 root servers each year. To estimate the total global leakage volume to all root servers, the query volumes in our results are calibrated by multiplying the average volumes per root server by 13.

Service name	Service description and documentation
wpad (N)	Web Proxy Auto-Discovery (WPAD) protocol, used by web clients to locate web proxies [198]
isatap (N)	Intra-Site Automatic Tunnel Addressing Protocol (ISATAP), used by dual-stack (IPv6/IPv4) clients to automatically tunnel IPv6 packets in IPv4 networks [98]
proxy (N)	Popular first label for a web proxy server [139]
vlmcs (N)	Microsoft Key Management Services (KMS), used by Microsoft clients to automatically activate volume license editions of Microsoft Windows and Office [79]
ntp	Network Time Protocol (NTP), used by clients to synchronize computer clocks in the Internet [85]
ns* (N), alt* (N)	Popular first label for a DNS name server [83, 91]
lb (N), db (N), dr (N), dns-sd	Labels for domain enumeration in DNS-SD [2]
tracker (N)	Used by BitTorrent users to locate the tracker, which manages BitTorrent peers in a torrent [118]
dns-llq	DNS Long-Lived Queries, used by clients to locate DNS servers with long-lived query support, which allows clients to learn DNS data changes without polling the server [41]
www* (N), api (N), static (N), share (N), cf (N)	Popular first labels for a server hosting web content, web elements, and web operations [32, 97, 116, 16, 33]
http, https	Hypertext Transfer Protocol, used by web clients to browse web content [64, 63]
stun	Session Traversal Utilities for NAT (STUN), used by clients to get the IP address and port allocated to it by a NAT [107]
ptp	Picture Transfer Protocol (PTP), used by clients to transfer images from digital cameras [93]
dpap	Digital Photo Access Protocol (DPAP), used by iPhoto clients to share photos starting in iPhoto 4.0 [23]
kerberos	The Kerberos service, used by clients to perform network authentications [73]
rubygems	Used by RubyGems, the package manager in Ruby to help clients download Ruby coding libraries [42]
gc (N)	Used by clients to locate a Microsoft Global Catalog (GC) server in a domain [80]
ldap, ldaps	Lightweight Directory Access Protocol (LDAP), used by clients to access directory services [76]
carddav, carddavs	vCard Extensions to WebDAV (CardDAV), used by clients to access, manage, and share contact information [26]
caldav, caldavs	Calendaring Extensions to WebDAV (CalDAV), used by clients to access, manage, and share calendaring information [25]
dns-update	Dynamic Updates in the DNS, used by DNS clients to add or delete resource records in DNS zones [45]
afs3-vlserver	Used by clients to access the Andrew distributed file system (AFS) [90]

Table 5.2: Descriptions and documentations of the exposed internal network services (Part 1). N denotes non-registered service.

Service name	Service description and documentation
adisk	Used by Apple Time Machine clients to perform automatic disk discovery [57]
smb	Server Message Block (SMB), used by clients to share file over a network [106]
afpovertcp	Apple Filing Protocol Over TCP, used by clients to share file over a network [9]
ftp, sftp-ssh	File Transfer Protocol (FTP), used by clients to transfer file over a network [50, 108]
webdav	HTTP Extensions for Distributed Authoring (WebDAV), used by web clients to manage remote web content [62]
odisk	Used by Mac Clients to access remote CD or DVD
rfb	Remote Framebuffer (RFB) protocol, used by clients to view and control a window system on a remote computer [122]
ssh	Secure Shell (SSH) protocol, used by clients to access a remote computer [23]
eppc	Used by clients to send remote Apple events [23]
telnet	Used by clients to access a remote computer [23]
kpasswd	Used by clients to change Kerberos passwords [74]
airport	Used by clients to configure a AirPort base station [23]
servermgr	Used by macOS clients to manage macOS servers [23]
autodiscover (N), outlook (N)	Exchange Autodiscover service, used by clients to automatically configure Microsoft Exchange [15]
mail*	Used by clients to locate POP3 or SMTP mail servers [31]
pop3	Post Office Protocol (POP), used by clients to locate POP mail servers [94]
smtp	Simple Mail Transfer Protocol (SMTP), used by clients to locate SMTP mail servers [109]
sip, sips, sipinternaltls (N), sipinternal (N), sipexternal (N)	Session Initiation Protocol (SIP), used by clients to create, modify, and terminate Internet telephone call sessions [110, 96]
xmpp-server, xmpp-client	Extensible Messaging and Presence Protocol (XMPP), used by clients to manage sessions for messaging, network availability, and request-response interactions [6]
printer	Used by client to locate network printers [23]
riousbprint	Used by the AirPort base station to share USB printers [23]
pdl-datastream	Used by client to locate network printers supporting Page Description Language (PDL) [92]
ipp	Internet Printing Protocol (IPP), used by clients to locate network printers supporting IPP [71]
scanner	Used by macOS clients to locate network scanners [23]
ica-networking	Used by macOS Image Capture app to share cameras [23]
xgrid	Used by macOS clients to locate Apple xGrid agents for distributed computing [78]
syslog	The Syslog protocol, used by clients to send and receive event notification messages [99]

Table 5.3: Descriptions and documentations of the exposed internal network services (Part 2). N denotes non-registered service.

Before the delegation of a new gTLD, the leaked internal service queries are answered by the DNS root servers as non-existent domains, or NXD [239]. Thus, from the DNS root traffic, we form the leaked query dataset by extracting queries with (1) NXD responses, and (2) TLD strings that have been delegated in the New gTLD Program today. In this work, we consider the delegated new gTLDs as of March 4, 2017, which include 1,216 new gTLDs in total [151].

Exposed service measurement. To measure the exposed services, we extract the service names from the queries in the leaked query dataset using the service discovery query format (§2.2.3). In our study, our main focus is the services officially registered in the IANA registry [65]. These are services that are widely used in industry, e.g., `sip` and `ldap`, and their IANA registration entries have service information such as protocol description, which are critical for us to understand and characterize their functionality.

To measure the registered services, we calculate the average daily query leak volume for each service in the IANA registry. One problem is that our measurements are impacted starting late 2013 as many of the new gTLDs began delegation and our observation space of the leaks decreases. To solve it, we obtain the delegation dates for the targeted new gTLDs and compute the per-TLD daily query leak volumes for each service, only using the data collected before each new gTLD's delegation date. Then, the average leak volume for a service is the sum of its per-TLD leak volumes.

In this work, we also study the non-registered services, some of which are also popular, e.g., the WPAD service. However, compared to registered services they are significantly more challenging to study due to the lack of readily available documentation as they are typically proprietary. It is especially difficult to identify services with non-standard queries, since all the first labels in the queried domain names are considered as candidate service names. This loose filtering condition results in a large number of potential service names and as described later in §5.3.2 a large portion of them are actually irrelevant, e.g., random strings potentially sent by Chrome for infobar customization [27].

To effectively identify valid non-registered service names from the extremely large candidate string set (78.5 million from our measurement in §5.3.2), we use an automated approach to conservatively rule out service name candidates that lack sufficient information for our study. We label each candidate service name with *nochar*, *noinfo*, *noinfo_suf*, or *info_suf*. If the name string does not contain an English letter, we label it *nochar*, indicating that the string itself lacks useful information about the service. Otherwise, we use a python script to search the string using Google, and if there are no search results, we label it as *noinfo*, indicating that the string is either not related to a service, or not popular enough so that no related information is available online for our study to proceed. If the label has search results, we then append it with popular service discovery suffixes and perform another Google search. For standard queries, we append suffixes `_tcp` and `_udp`. For non-standard queries, we append `example.com`, `example.net`, `contoso.com`, and `contoso.net`, which are popular example domain names in network service documentations [47, 66]. If these searches do not have results, it is labeled as *noinfo_suf*; otherwise it is *info_suf*. In the subsequent protocol study, we then focus on the candidates with *info_suf* labels.

5.3.2 Exposed Services

From our measurement, 115 registered services in the IANA registry are found to have service query exposure. The leakage volume distribution exhibits a long tail property with 40.9% (47) of the services receiving less than five queries globally per day. To focus our analysis on the ones with considerable degrees of query exposure, we pick the top 50 services for subsequent analysis.

For the non-registered services, since the query formats are loosely defined, the output of our measurement includes 78.5 million candidate strings. Fig. 5.2 shows the automatic labeling results using the automatic labeling script (§5.3.1). As shown, for the top 50 server string candidates, the majority (60%) of them are popular names that at least have some

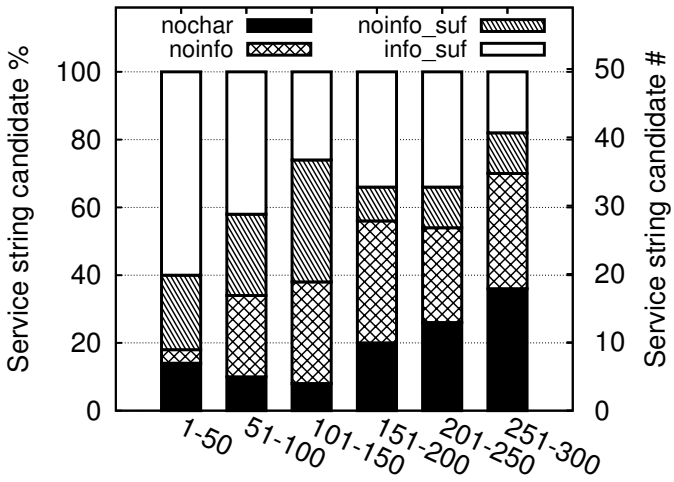

Figure 5.2: Automatic labeling results for the top 300 non-registered service string candidates.

online references or configuration tutorials. After the top 50, 60–80% of the names do not have related online information. As shown, the majority of them either have no letters in the name, or have no search results even without DNS domain suffixes. While registered services are the main focus of this work, we pick the 30 service names with *info_suf* labels among the top 50 candidates for our subsequent analysis. This enables us to cover the most popular non-registered services, making our study more comprehensive.

In total, we form an exposed service dataset of 80 services with highest levels of service query exposure today. Table 5.1 characterizes 68 of them according to their designed functionality and potential security implications under name collision attacks. Table 5.2 and Table 5.3 include the documentations we collected for each of them. For the remaining 12, we are unable to further analyze them since they either have no online documentation or no precise service information (details in Table 5.4). In Table 5.1, the average daily query leak volumes are presented as circled numbers ① to ⑤ indicating five volume ranges. For non-registered service names, strings that only differ in the suffix numbers, e.g., `www1` and `www2`, are aggregated into names ending with "*", e.g., `www*`.

As shown in Table 5.1, in addition to the previously-studied WPAD service [172], the name collision problem actually affects a wide spectrum of internal network services with

Service name	Exclusion reason
ssp④, grid⑤, dltimesync⑤, fmserver-admin⑤	No online service documentation
server*① (NR), your① (NR), test① (NR), int① (NR), personalize① (NR), dc*① (NR), ad*① (NR), domaindnszones① (NR)	No service specific information

Table 5.4: Services in the exposed service dataset without sufficient information for us to perform service design characterization. Numbers in circle denote the range level of the average daily query leak volume: ① > 100,000, ② 10,000 – 100,000, ③ 1,000 – 10,000, ④ 100 – 1,000, ⑤ 10 – 100. NR denotes non-registered service names.

diverse functionality today. More importantly, many of these exposed services are critical for security and privacy, e.g., proxy configuration, coding library discovery, printer discovery, etc. As shown in Table 5.1, if the service software has client-side name collision vulnerabilities, attackers may cause a wide range of security problems. In the next section, we collect the service implementations to concretely evaluate their robustness under the name collision attack threat model.

5.4 Vulnerability Analysis

To evaluate the robustness of the exposed services under name collision attacks, in this section we perform vulnerability analysis on the service client implementations and analyze the causes.

5.4.1 Methodology

Service client implementation collection. For each exposed service, the goal is to collect its client implementations that are generating the leaked service discovery queries observed in our measurement. Based on the service names and registration information, we read over ten pages of Google search results, download and test candidate software. We only pick a candidate if it is manually confirmed to (1) use DNS-based service discovery, and (2) automatically combine the service name and a discovery domain to form the discovery query. The discovery domain configuration processes depend on the client

Exposed service	Client implementation	Usage	Vulnerability cause				Vuln. ?
			V1	V2	V3	V4	
ldap	In-domain Windows 10 logon, official Linux command `ldapsearch`	U1	✗	-	-	✓	✓
	IPA Client logon	U1	✗	-	-	✗	✗
wpad	Windows 10 WPAD service	U1	✓	-	-	-	✓
isatap	Windows 10 ISATAP tunnel service	U1	✓	-	-	-	✓
kerberos	In-domain Windows 10 logon, IPA logon	U1	✗	-	-	✗	✗
dns-sd, lb, db, dr	macOS 10.12 domain enumeration	U1	✓	-	-	-	✓
sip, sipinternal, sipinternaltls, sipexternal	Skype for Business 2016	U1	✗	✓	-	✓	✓
	X-Lite, Blink, Phoner, Linphone, Jisti	U1	✓	-	-	✓	✓
gc	In-domain Windows 10 `DSQUERY` command	U1	✗	-	-	✓	✓
mail	Outlook 2016 IMAP service	U1	✗	✓	-	✓	✓
autodiscover, outlook	Outlook 2016 Autodiscover service	U1	✗	✓	-	✓	✓
kpassword	Kerberos for Windows	U1	✗	-	-	✗	✗
pop3	Outlook 2016 POP service	U1	✗	✓	-	✓	✓
smtp	Outlook 2016 SMTP service	U1	✗	✓	-	✓	✓
sips	X-Lite, Blink, Phoner, Linphone	U1	✗	✓	-	✓	✓
	Jisti	U1	✗	✗	-	✓	✓*
ipp, printer pdl-datastream, riousbprint	macOS 10.12 printer discovery	U2	✓	-	$\checkmark^{q,r}$	-	✓
xmpp-server	ejabberd	U1	✓	-	-	-	✓
ntp	IPA Client logon	U1	✓	-	-	-	✓
xmpp-client	PSI logon, Adium logon	U1	✗	✓	-	✓	✓
http	macOS 10.12 Safari Bonjour browser	U2	✓	-	$\checkmark^{q}$	-	✓
stun	X-Lite, Blink	U1	✓	-	-	-	✓
afs3-server	IBM OpenAFS	U1	✗	-	-	✗	✗
carddav	iOS 10.3 Contacts CardDAV account	U1	✗	-	-	✓	✓
adisk	macOS 10.12 Time Machine disk discovery	U2	✗	-	$\checkmark^{q,r}$	✓	✓
afpovertcp, smb, rfb	The Shared section in macOS 10.12 Finder	U2	✗	-	$\checkmark^{q}$	✓	✓
ssh, telnet ftp, sftp-ssh	"New Remote Connection..." in macOS 10.12 Terminal	U2	✗	-	$\checkmark^{q,r}$	✓	✓
caldav	iOS 10.3 Calendar CalDAV account	U1	✗	-	-	✓	✓
dpap	macOS iPhoto photo sharing	U2	✓	-	$\checkmark^{q,r}$	✓	✓
carddavs	Contacts CardDAV in macOS 10.12, iOS 10.3	U1	✗	✓	-	✓	✓
webdav	Cyberduck discovery	U2	✗	-	$\checkmark^{q}$	✓	✓
dns-llq	macOS 10.12 Back To My Mac service	U1	✓	-	-	-	✓
severmgr	macOS Server 5.1 discovery	U2	✗	✓	$\checkmark^{q,r}$	✓	✓
dns-update	macOS 10.12 dynamic global hostname	U1	✓	-	-	✓	✓
rubygems	RubyGems `gem` and `bundle` commands	U1	✓	-	-	-	✓
caldavs	Calendar CalDAV in macOS 10.12, iOS 10.3	U1	✗	✓	-	✓	✓

Table 5.5: Vulnerability analysis results for the collected client implementations of the exposed services. "q" and "r" denote query-side and response-side mixing in domain discovery (detailed in §5.4.3). "*" means that the vulnerability status depends on user decisions.

Service name	Exclusion reason
scanner	Need a physical scanner for server setup
ptp	Need a physical camera for server setup
airport	Need an Apple AirPort for server setup
vlmcs (NR)	Need a valid Microsoft production key for server setup
xgrid, ica-networking	Deprecated in the latest macOS
ldaps, https, eppc, odisk, syslog	Failed to find a client software using the service name with unicast domain discovery
www* (NR), api (NR), static (NR), share (NR), cf (NR), ns* (NR), alt* (NR), proxy (NR), tracker (NR)	These are non-official naming conventions; Excluded since we are more interested in uncovering the vulnerable design and implementation choices made under the default service discovery configurations.

Table 5.6: Services excluded in the client-side name collision vulnerability analysis. NR denotes non-registered service.

implementation details. For the clients we have explored, they typically use the OS domain or the user account domain. For services with multiple client implementations, our analysis mainly focuses on the ones that are more popular among corporate or end users and thus has higher impact. We focus our analysis on the most recent releases available to us so that our analysis results are current and relevant.

Column one and two in Table 5.5 list the services and the collected client implementations. This collection includes 57 client implementations covering 48 (70.6%) out of the 68 services with service design information in our exposed service dataset. We prioritize our efforts to cover the registered services with the highest level of exposure. Specifically, our collection covers 14 out of the 17 registered services with over 1,000 daily query leaks in Table 5.1. For the remaining ones, we were unable to obtain valid software for our study (details in Table 5.6).

Many services are registered for a single product, e.g., `gc` and `outlook`. Thus, most of the services in the table only have one client listed. Also, the list of clients has a skew towards a particular vendor's products because that vendor is the major supporter for DNS-SD and has registered many of them for their own use, e.g., `adisk`, `afpovertcp`, `dpap`, etc. [65]

Analysis steps. Due to service functionality or usage model differences, these service clients may need to contact multiple servers for different purposes and ultimately result in different levels of vulnerability exposure in name collision attacks. Thus, to systematically analyze the client-side name collision vulnerability, we first perform a characterization of the service discovery usage scenarios implemented in the collected clients. In this analysis, we first configure the client and server software, and ensure the service functionality is performed as expected. We then trigger the service discovery in the client, and analyze the network traffic to understand the usage.

Based on the usage characterization results, we identify the attack points for each usage scenario and perform vulnerability analysis for a client at every attack point. Our analysis uses a simulated name collision attack environment and dynamically analyses the vulnerability status. The victim client software is installed in the same computer being connected to two namespaces, simulating an internal DNS namespace and the public DNS namespace in which name collisions can occur. Our analysis assumes the absence of the attacker at the first-time software usage. Thus, we first trigger the designed functionality without attack in the simulated internal DNS namespace. After that, we disconnect the client from all legitimate internal servers and switch to the simulated public namespace to start the vulnerability analysis. For the client-side requests at each attack point, we construct possible attack server responses only using resources available in the public namespace. Note that since we do not assume the attacker can access the legitimate internal servers (§5.2.1), the attack and legitimate servers are configured differently in the non-default settings, e.g., server names, zone file content, user credentials, etc. Using this analysis method, a client-side name collision vulnerability is revealed if the client accepts the attack server responses at all attack points of its usage scenario. Since this method identifies vulnerabilities by directly testing attacks, it does not have false positives, but can have false negatives since it may not explore all vulnerable paths in the software, which is an inherent limitation for dynamic analysis [115].

Analysis framework. We develop a dynamic analysis framework to support the analysis tasks above. The victim client software is installed in a virtual machine configured in two network environments, each having its own DNS server but using a local zone file with the same domain name. To conform to our threat model, we registered a real new gTLD domain to set up the name collision domain. During the vulnerability analysis, we switch the network environment by changing the client DNS resolver address and shutting down all server virtual machines in the previous network environment. In this framework, the traffic between the client and the server are intercepted by a MitM proxy for the protocol analysis on TLS traffic and the attack server response injection. We develop this proxy by customizing the SSLSplit tool [113].

For the services using TLS, the servers in the simulated internal namespace are configured with certificates signed by a local CA. To simulate the public namespace, we obtain valid public certificates for the attack servers for free through the Let's Encrypt CA [75]. After obtaining the certificates, we configure our DNS servers to only serve the IP addresses we control.

Note that currently most parts of this analysis are manual. We made this decision because generally automating the analysis, e.g., identifying, configuring, and triggering a targeted behavior for arbitrary software, is very challenging. The diversity in platforms and software design further complicates the task. Despite the manual effort, our analysis covers a relatively complete range of services so the analysis results, our main contribution, are not significantly affected. The efforts made in this work also helps shed light on how to automate the analysis in the future.

5.4.2 Service Discovery Usage Scenarios

Our analysis identified two usage scenarios:

U1. Locate a single server in the discovery domain. In our collection, the clients for 33 of the 48 exposed services are in this usage scenario. As shown on the left of Fig. 5.3,

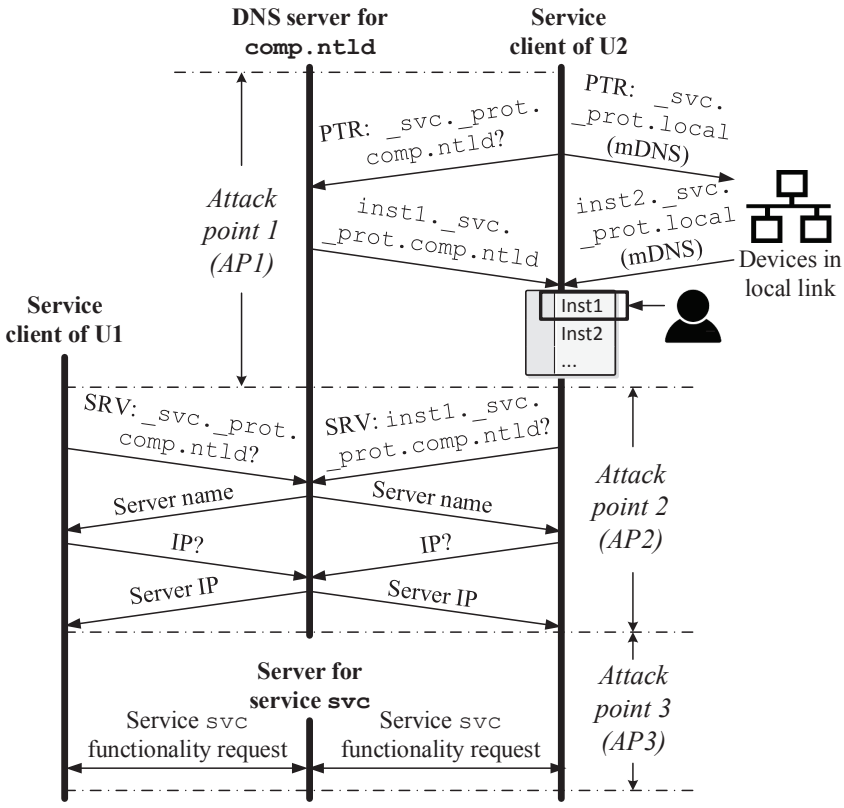

Figure 5.3: Illustration of usage scenario U1 and U2 of DNS-based service discovery (§5.4.2) in our service client collection.

these clients use the traditional service discovery methods via SRV or A/AAAA queries as introduced in §2.2.3. In U1, the client contacts two attacker-controlled servers, the DNS server for `comp.ntld` and the server for the requested service `svc`. In the figure, they are labeled as AP2 and AP3, denoting the two attack points in name collision attacks.

U2. Locate multiple servers in both local link and the unicast discovery domain. The remaining 16 clients, covering 15 of the 48 exposed services, use DNS-based service discovery to find a list of services instead of a single one. These clients uses PTR queries to retrieve a list of server names for the user to choose as shown on the right of Fig. 5.3, which conforms to the Zeroconf usage of the DNS-SD standard [144]. Even though Zeroconf is mostly designed for discovering nearby devices without unicast DNS server support [22], these clients have unicast query leaks mainly due to their support of discovering both the

local link via mDNS and the discovery domain via unicast DNS. Compared to U1, clients in U2 have an additional user selection step. Thus, to increase the attack success rate, the attacker needs to spend extra effort to carefully craft PTR responses to trick the user into choosing the attack server in the list, which is labeled as AP1 in Fig. 5.3.

5.4.3 Vulnerability Analysis

Using the analysis framework, 57 clients in 45 (93.8%) of the 48 exposed services are found to be vulnerable. In this section, we report four common vulnerable software design or implementation choices causing such widespread vulnerability exposure. The analysis results are summarized in Table 5.5.

V1. Lack of server authentication by default. At AP1 and AP2, even though the attack zone file setup and the DNS response content are different from the legitimate one, we find that all 57 clients accept the malicious DNS responses after switching namespaces. This is not entirely surprising: DNS clients solely rely on their recursive resolvers to locate the appropriate DNS servers, and thus are are not in the position to differentiate namespaces.

Even though name collision attackers can pass AP1 and AP2, the 57 clients have the full potential to block the attack at AP3, where various server authentication methods can be used. However, to our surprise, we find that the clients for 16 (33.3%) out of these 48 exposed services do not implement any server authentication method by default for the server(s) from discovery. It includes four printing service software, seven communication service software, tunneling services `isatap`, etc. As shown in Table 5.1, the potential security implications for these services are high severe, including MitM attacks, malicious script execution, document leakage, etc.

This lack of server authentication is not entirely poor implementation choices. Services such as `wpad`, `isatap`, `stun`, etc., have no server authentication specified in their protocol design [198, 72, 107]. Also for services such as `sip` and `ftp`, server authenti-

cation is mentioned but the implementation is not enforced [51, 105]. Thus, to solve this problem, both the service design documentation and the actual implementations need to be strengthened.

V2. Accept a publicly-valid but previously-unseen TLS certificate by default. In our client collection, 36 clients for two thirds (32) of the 48 exposed services do use server authentication in AP3. Seventeen of them use TLS certificates, and when we first trigger the service functionality in the simulated internal namespace, all of them require explicit user addition or approval steps to trust the legitimate internal server certificates signed by the local CA. However, after switching to the simulated public namespace, we find that 16 of the 17 clients by default accept the publicly-valid but previously-unseen TLS certificate we prepared for the attack server. As shown in Table 5.5, they involve highly popular clients for VoIP, mail, contacts, and calendar. The only client that does not accept the attack certificate by default is `sips` software Jisti, which requires user approval for any certificate that was not previously seen. However, it still relies on the user to make the correct security decision instead of directly terminating the connection.

Since in our experiments the connection to the intended server in the internal network is established first, these clients should be capable of distinguishing the attack certificate from the legitimate one, e.g., they are not signed by the same CA. Unfortunately, they make the choice of accepting any previously-unseen publicly-valid certificate by default. This may be because they are not designed to be only used in the internal network. For example, for the SIP and XMPP clients, the targeted use cases are not only for the internal servers but also for the public ones such as `sip2sip` and `xmpp.jp`. Thus, to increase the convenience in the latter use cases their trusted CA lists by default include the public CAs.

Note that this is not a weakness in TLS-based server authentication. The server authentication in TLS is only designed for validating the certificate chain for a given domain name, and in this case, the certificate chains are indeed valid in both internal and public networks. The fundamental cause is in the use of TLS-based server authentication in these

service clients: they are designed to be used in both internal and public namespaces but lacks the awareness of differentiating the use in different namespaces. Later in §6.8, we have more discussions on this problem cause.

V3. Mix local-link and unicast DNS domain discovery. As discussed in §5.4.2, clients in U2 are mainly designed to discover servers at nearby devices in the local link using mDNS queries [22] and thus should not be exposed to the name collision problem. However, to extend such discovery to wider areas beyond a local link, e.g., large corporate networks with multiple subnetworks, these clients also browse a configured discovery domain using unicast DNS queries. Unfortunately, such extended functionality support causes the 16 clients in U2 to have mixed queries to both local link and the unicast DNS domain, causing service query exposure.

In fact, we find that for many Zeroconf software with unicast DNS discovery, such extended functionality is actually not always necessary. Two examples are the macOS Parental Control function, which implements the registered service `parentcontrol`, and the Docs To Go software, which implements the registered service `dxtgsync`. The macOS Parental Control is designed for parents to monitor and manage their children's Mac computers, and Docs To Go is used for a single user to synchronize documents on various personal devices under the same WiFi. They are mainly designed for accessing nearby devices in local link without a local unicast DNS server setup, but they by default generates unicast DNS discovery queries along with the mDNS queries. Another example is application DropCopy that implements the registered service `dropcopy`. It is designed to transfer files among nearby devices similar to Docs To Go. It uses unicast DNS discovery but does not show the results in the server list, making it more obvious that the unicast DNS discovery functionality is actually not needed.

To understand why these software implementations choose to support the redundant unicast DNS discovery, we take a close look at the most popular Zeroconf framework, Bonjour, and find that this is potentially caused by the default behavior of the discovery

API. We analyze the API according to the documentation [21], and find that if the domain parameter is not specified, the discovery API by default discovers both the local link using mDNS and the system-configured domain using unicast DNS. Thus, if the developer is not careful enough, such default API behavior with mixed local-link and unicast domain discovery unnecessarily causes the software to be exposed to name collision attacks.

Besides the queries, the mixing of local-link and unicast domain discovery also happens to the discovery results in the responses. Among the 16 implementations in U2, 11 of them do not differentiate the servers from local-link discovery and those from unicast domain discovery in the user selection step. These implementations are all in macOS system applications, including the printer discovery process and the system terminal's remote shell functionality. As illustrated in Fig. 5.3, only the service instance name strings, e.g., "inst1" if the response is `inst1._svc._prot.comp.ntld`, are shown to the user without any indicator of the discovery domain. This makes it impossible for even a security-savvy user to tell the associated discovery domain for the discovered servers, allowing name collision attackers to have arbitrary control over the content shown on the user interface and thus more easily influence the user choice. Later in §5.5, we use concrete examples to illustrate how this can be exploited to directly prevent the user from choosing the legitimate server. For the other five clients, the discovered servers in the list are labeled with the namespaces, e.g., "local" for local link and "comp.ntld" for discovery domain `comp.ntld`. Actually among them there are some macOS system applications, e.g., Finder and Safari Bonjour browser, but unfortunately this practice is not consistently enforced throughout the system.

V4. No enforcement of server authentication in PSK-based authentication. Besides using TLS certificates, PSK can also be used to provide server authentication at AP3. Since the password is only shared with the intended internal server, the client can detect a name collision attack server since the attacker cannot prove the possession of the correct password. In our collection, 42 clients for 33 services use PSK-based authentication by default. However, we find that 37 clients for 30 (90.9%) out of these 33 services have no

enforcement of server authentication. Nineteen of them only use the PSK for client authentication without requiring server authentication by default. For some, the user name and password are even sent in plain text to the attack server. This not only fails to detect the attack server but also leads to credential theft.

The remaining 18 clients choose to use mutual authentication methods such as Kerberos or DIGEST-MD5 by default. However, they are still found vulnerable since they can all be downgraded to not use mutual authentication if the attack server suggests so, without even notifying the user that the current authentication is less secure. For XMPP implementations PSI and Adium, the CardDAV, CalDAV, and WebDAV implementations including macOS and iOS Contacts and Calendars, HTTP Authentication is used with DIGEST-MD5 by default, but these clients can all be downgraded to use Basic, which sends the Base64 encoded password. We find that without TLS being enabled, PSI and Adium actually refuse to use Basic by default. We suspect that they choose to accept the weaker authentication method under TLS mainly because they assume that the server has already been authenticated after it passes the TLS certificate validation. Unfortunately, according to our analysis results for V2, this assumption is generally broken for the collected clients using TLS, including PSI and Adium.

The `ldap` implementation used during Windows 10 logon, the SIP client Microsoft Skype for Business 2016, the mail client Outlook 2016, and the macOS Finder implementation of `smb` all by default use Kerberos. However, they can be downgraded to using NTLM, which does not provide server authentication [30]. For another `ldap` client, the official Linux LDAP command `ldapsearch`, when the PSK-based authentication method is not explicitly specified, the program by default accepts the server suggestion of using NTLM. The disk discovery implementations in macOS Finder and Time Machine, which implement `afpovertcp` and `adisk` respectively, both use the AFP (Apple Filing Protocol) [9] and by default use Kerberos. However, it can be downgraded to use DHX2 (Diffie-Hellman Key Exchange 2) that is clearly explained in the documentation that "there

is no way for the client to verify that the server knows the password" [3].

Among the 57 clients, only six do not exhibit the client-side name collision vulnerability in our analysis. Besides the SIP implementation Jisti that depends on the user to make the right decision, the other five are not vulnerable since they only support Kerberos.

5.4.4 Discussion

As shown above, nearly all (51) of the 57 service clients we collected are vulnerable to name collision attacks, suggesting that the name collision attack threat model broadly breaks common security assumptions made in today's internal network service clients. From the analysis, we observe two fundamental causes at the software design level. First, internal service clients today tend to place excessive trust on the server side. As shown, clients for one third of the 48 exposed services do not use any server authentication by default. For clients using PSK-based authentication, around 90% of them have no enforcement of server authentication. This is probably because by design these clients are expected to be used in internal networks, e.g., in companies, which have network isolation and certain levels of security protections, and thus make the security assumption that the servers are trusted. Such assumption is broken under the threat model of local network attacks, e.g., ARP spoofing attacks, but it may be deceptively safe considering that these attacks usually have tight requirements of the attack placement and timing. However, with internal query leaks exposing these service clients to malicious servers in the public network, the name collision attack threat model is thus a new attack vector to break such assumption, and as discussed in §5.2.1, it is more powerful and easier to launch than typical local network attacks.

Second, the service clients today using DNS-based service discovery generally lack the awareness of differentiating the namespaces of the domain names in the DNS responses. In this work, we call it a lack of *namespace differentiation*. As shown, in service discovery, 11 of the 16 clients in U2 do not differentiate the responses from the local-link namespace

(`.local`) and the unicast domain namespace. For server authentication, 16 out of the 17 clients relying on TLS to authenticate a domain name accept a certificate from the public namespace by default, even though the previously user-approved certificate for such domain name is from an internal namespace. This is a fundamental problem introduced by the newly-emerged name collision attack threat model. By design, DNS namespaces should be isolated and thus internal network service clients are not expected to differentiate namespaces. However, since the name collision problem is happening today, such a general lack of namespace differentiation leaves the service clients incapable of handling potential name collisions, causing the vulnerability exposure in our analysis.

Based on these insights, in §5.6 we propose a set of defense strategies at the service client software design level.

5.5 Exploitation Case Study

To demonstrate the severity of the identified vulnerabilities, we construct realistic attack scenarios in our analysis framework. Table 5.7 summarizes our results. As shown, a number of new name collision attacks are uncovered with a wide range of security implications, i.e., a new MitM attack vector in addition to the WPAD name collision attack [172], document leakage, malicious code injection, credential theft, and phishing attacks. Note that the new MitM attack vector exploits an IP tunneling service, and thus besides intercepting web traffic like the WPAD attack, it can also intercept DNS traffic and launch DNS-based exploits such as DNS response spoofing.

5.5.1 MitM Attack

In the previous work [172], the name collision attack on the Windows implementation of `wpad` has been found to cause MitM attacks. In this study, we find that the Windows implementation of `isatap`, a service with completely different design purpose in comparison to `wpad`, can also be exploited by name collision attackers to launch MitM attacks.

Section	Client software	Service name	Vuln. design/ impl. choice	Exploitation
§5.5.1	Microsoft Windows 10	isatap	V1	MitM attack (web traffic & DNS traffic)
§5.5.2	RubyGems 2.6.12	rubygems	V1, V2	Malicious code injection
§5.5.3	macOS 10.12 printer discovery	ipp	V1, V3	Document leakage
§5.5.4	PSI 0.15, macOS 10.12 and iOS 10.3 Contacts, Calendar	xmpp-client, caldav, caldavs, carddav, carddavs	V1, V2, V4	Credential theft
	macOS 10.12 Terminal New Remote Connection	ftp, ssh, sftp-ssh, telnet	V1, V3, V4	
§5.5.5	Linphone 3.10.2, PSI 0.15	sip, xmpp-client	V1, V2, V4	Phishing calls & msg.
§5.5.6	macOS 10.12 and iOS 10.3 Contacts, Calendar	carddav, carddavs caldav, caldavs	V1, V2, V4	Phishing name card & calendar event injection

Table 5.7: Exploitation case studies for the identified client-side name collision vulnerabilities. V1 to V4 are detailed in §5.4.3.

In this section, we report the attack construction for the implementation of `isatap` in Windows 10.

The protocol for `isatap` service is ISATAP (Intra-Site Automatic Tunnel Addressing Protocol) [72]. It is an IPv6 transition mechanism to enable a client to use IPv6 in an internal network that only has IPv4 network infrastructure. The ISATAP server is connected to both the internal IPv4 network and an external IPv6 network. When the client uses IPv6, the traffic is encapsulated in IPv4 packets between the client and the ISATAP server, and then decapsulated by the ISATAP server to contact the IPv6 sites on behalf of the client. This mechanism is implemented primarily in Windows OSes and also supported in Linux.

Service query exposure. We first configure a Windows 10 client to log into a Windows domain, which is common practice in a corporate network [34]. In our simulated name collision attack environment, we find that the ISATAP service is by default enabled and the service query exposure happens at the OS booting time, during which the client sends a query by prepending the `isatap` label to the Windows domain to discover the ISATAP server.

Exploitation. We set up the attack server by configuring the Linux ISATAP router program `radvd` with IP forwarding enabled [77]. Since the client makes vulnerable choice

V1, i.e., having no server authentication by default, to launch the exploit we only need to point the client requested query name to our ISATAP server IP address in the colliding domain zone file.

In the virtual machine setup, our host machine does not have a public IPv6 address allocation. This emulates a network without IPv6 support, which is common in home networks using popular cable services in majority of the states in the U.S. [29]. After switching to the attack environment, after booting the Windows 10 client is found to have accepted and already configured to use the attack server in the ISATAP tunnel interface. Then, all web traffic to IPv6 sites are found to be intercepted by our attack server.

In addition, we find that as an IP tunneling protocol, ISATAP can affect not only web traffic like WPAD, but also DNS traffic. In our experiments, we find that the Windows 10 OS prioritizes the IPv6 DNS servers when both IPv4 and IPv6 DNS servers are configured. Thus, as long as the DNS configuration includes one IPv6 DNS server, for example popular public open resolvers [54], the ISATAP tunnel is tried first for any DNS request. This causes all DNS queries to be intercepted, leading to another dimension of exploits leveraging DNS response spoofing. In this case, since the attacker acts as the client's DNS resolver, she can bypass the DNSSEC integrity check due to the last mile problem for DNS [119] and can thus read and modify arbitrary DNS responses.

5.5.2 Malicious Library Injection

Service `rubygems` is used in RubyGems, the Ruby package manager serving Ruby coding libraries, called gems. In this section, we detail a name collision attack on this service to inject malicious libraries.

Service query exposure. To use the service, RubyGems first needs to configure the discovery domain, called adding sources. The sources typically include the official Ruby package server `rubygems.org` to discover standard packages, and can also include internal domains to download libraries developed internally, e.g., those for company-wide use

only. After switching to the attack environment in our analysis framework, service queries to both the official package server and the internal domain are triggered when installing or updating a gem with the `gem` commands or the `bundle` commands. Since the source configuration is at the coding platform level, as long as the user runs these commands on a computer with the internal source configured, e.g., a corporate computer, the internal domain discovery is always triggered even if the user's coding task is unrelated to internal libraries.

Exploitation. We set up the attack server using Gem in a Box [124]. The attack goal is to let the client install an attacker-prepared version of a public library that should be served at the official Ruby package server. Compared to infecting internal libraries, preparing malicious public libraries are easier since the API information is public, and can potentially infect more parts of a developer's program when targeting popular libraries. Also it is more stealthy since the attacker can carefully insert code so that the API functionality still appears normal. In RubyGems, we find that when the `gem` or `bundle` commands are triggered, all the sources are contacted simultaneously to download the required gem instead of being contacted one by one. Since the public library we target is served by the official package server, the malicious library on the attack server needs to compete with the legitimate one.

From the source code, we find that the client first retrieves the metadata of the requested gems from all servers, merges them into a list, and picks the last one in the list to install. Each gem metadata is a tuple with several package information, and the different fields for gems with the same name are mainly the version number and the source server name. In the code, this list is sorted using the default Ruby tuple comparator; thus, for gems from the same source, normally the one of the latest version is picked. For the `install` commands of `bundle` and `gem`, we find that the source server name field is put before the version number field in the tuple. Thus, as long as the malicious server name is larger than the official Ruby package server name in string comparison, the gem from the attack server is

picked. In the SRV response, the attacker controls the attack server name string, and can thus deterministically force the installation of the malicious gem. For the `update` commands of `bundle` and `gem`, only the version number is used so that the gem is installed only if its version number is larger than the one that is already installed by the client. Thus, for `update` commands, the attack gem just needs to use a version number that is higher than that in the latest public one to force the installation.

For both `install` and `update` commands, we have verified that the injection succeeds by modifying the official `net-dns` gem to include additional code. These commands can also use HTTPS, but similar to the results in §5.4.3, they are found vulnerable due to V2. Using our analysis framework, we have also verified the injection attack under HTTPS. Also note that since the version number is used in the gem selection during the `update` commands, the attacker can use a large version number to prevent future updates of a gem, making the malicious library injection permanent.

5.5.3 Document Leakage

In macOS, the printer service discovery is triggered by going to the Printer & Scanner preference page, or by clicking on the Print option in an editor. The discovery process discovers both the local link and the configured device domain for exposed service `ipp`, `printer`, `pdl-datastream`, and `riousbprint`. The attack strategies for these services are the same, and in this section we only describe the attack on `ipp` as an illustrative example.

In the `ipp` discovery in macOS, only the service instance name fields are extracted and presented in a list for the user to choose (illustrated in Fig. 5.3). This leads to the vulnerable design choice V3 on the discovery responses, allowing the attacker to have arbitrary control over all the user-visible content. Thus, the attack strategy is to pick a deceptive name to trick the user into using the attacker's printer. Since printers are typically named using the brand and the model number, e.g., "Brother HL-6180DW series", the attacker can just

pick some popular names. Even if the user is intended to use a nearby printer in local link instead of a network printer in the discovery domain, she can still make the wrong choice since the service instance names alone provide no such information.

In addition, we find that the attacker can control not only the content, but also *the order* of the printer names in the list. In the implementation, the discovered printer name list is sorted in ascending alphabetical order. Thus, an attacker can start the attack printer name string with an invisible character that is smaller than any English letter in character comparison, e.g., `STX` (ASCII code 2 [12]). With this, we have confirmed that the attacker-injected printer name is always ranked the first so that it is more likely to be chosen, especially when the names in the list all appear legitimate.

For the macOS `ipp` implementation, no server authentication is used in AP3. Thus, once the user picks the attack printer, the printed documents are leaked to the attacker. Thus, to prevent such exploit it solely depends on whether the user can choose the legitimate name. However, since only the service instance name field is used, both the printer name and order in the user interface are fully controlled by the attacker to influence the user's choice.

5.5.4 Credential Theft

For the services in U1, sending a password in plain text to the server in the PSK-based authentication directly causes credential theft. XMPP client PSI, CardDAV and CalDAV clients macOS and iOS Contacts and Calendar, all use TLS but accept a publicly-valid certificate by default. After bypassing such check by exploiting V2, the attack server then exploits V4 to get the password in plain text by suggesting to use PLAIN for PSI, and Basic for macOS and iOS Contacts and Calendar. These exploits are stealthy, since these clients support background launching or syncing: PSI by default launches during OS booting, and macOS and iOS Contacts and Calendar have periodic syncing that can be as frequent as every one minute. Details about the attack setup are in §5.5.5, §5.5.6.

For the services in U2, causing password leakage requires not only a weak PSK-based authentication method, but also an effective way to trick the user to select the attack server in the discovered server list. In the following, we detail credential theft attacks on macOS service discovery implementations for `ftp`, `ssh`, `sftp-ssh`, and `telnet`. When using the macOS default terminal, the user can choose to browse remote connections with these four options. Once they are clicked, the client issues the corresponding service discovery queries to both the local link and the unicast discovery domain. If the user picks the attack server to connect, these clients send the credential in plain text to the attack server.

For these clients, the key step of the exploitation is to leverage V3 to trick the user to pick the attacker-provided server. In their UI design, we notice that the server names from the discovery is displayed in a ranked list with a limited height, which requires scrolling if the list is too long. Thus, the attacker can send a long list of server name strings with invisible characters that are smaller than any English letter in character comparison, e.g., a TAB [12], to push the legitimate server names out of the first page of the list. Meanwhile, the attacker sends a deceptive server name starting with another invisible character that is even smaller in character comparison, e.g., `STX` [12]. In the current implementation, when the server name list is longer than the visible area, the scrolling bar is not shown by default. Thus, in the list, the attack server appears to be *the only choice* when the user browses for remote connections. With the use of a legitimate looking name such as "MacBook Pro," it is likely that the user will at least try this only choice, especially when she sees the same results after re-browsing multiple times.

5.5.5 Phishing Calls and Messages

Service query exposure. For the SIP client Linphone and the XMPP client PSI, a user account, for example `alice@comp.ntld`, is needed for service registration. When the account is configured, these clients perform service discovery of U1 in the user account domain, typically happening at the software launching time. Since these clients remem-

ber the account name, when launching them after switching to the attack environment in our analysis framework, they still perform discovery based on internal domains in the account names, causing service query exposure. Both Linephone and PSI support automatic launching during OS booting. Thus, their query exposure may happen even without user interaction. In fact, this automatic launching is the default configuration for PSI.

Exploitation. We set up the attack server using Asterisk for Linphone [13], and ejabberd for PSI [48]. During the account logon, Linphone sends password in MD5 hash without server authentication, which is the designed SIP authentication method [110]. For PSI, the authentication process by default uses DIGEST-MD5, which provides server authentication [61]. By modifying the ejabberd server response in our analysis framework, we let the server claim to only support PLAIN, which requests the client to send the password in plain text. As reported in the vulnerability analysis (§5.4.3 V4), with a publicly valid certificate used to bypass its certificate check, PSI accepts to use PLAIN.

In this attack, our attack servers are implemented to allow a client to log in with any password. More specifically, after getting the user name from the client logon request, the attack server creates an account with the same user name but a different password on the attack server. To allow the victim client to use her password to log in without modifying the server software, we utilize an attacker-side proxy to replace the credential used in the client authentication with the valid credential for the attack server. Since the proxy is attacker-controlled, the credential is still leaked, while the victim client appears to be logged in as normal.

After the victim client is logged in, the attacker uses another account on her server to initiate phishing calls or messages. Under both Asterisk and ejabberd, we have confirmed that the displayed caller or sender names of the attacking accounts are controlled by the attacker. Thus, the attacker can choose a deceptive name, e.g., “Manager” or “IT Department”, to increase the success rate.

5.5.6 Phishing Contacts & Calendar Events

Service query exposure. For system applications Contacts and Calendar on macOS and iOS, the user can configure CardDAV and CalDAV accounts in the form of `alice@comp.ntld`, and the account domain becomes the discovery domain. After contacting the discovered server, new contacts and calendar events are retrieved and merged with those from all other accounts such as the iCloud account to present to the user. These clients have periodic synchronization with the server, which can be every one minute, one hour, etc. When the user leaves the internal network, e.g., at home after work, this periodic synchronization can thus directly lead to service query exposure. In our analysis framework, we set it to synchronize every 1 minute, and have confirmed the query leakage after switching to the attack environment.

Exploitation. We use Baïkal to set up the attack server for these CardDAV and CalDAV clients [19]. To avoid using server authentication, we configure the server to use Basic instead of the default choice DIGEST-MD5 as the PSK-based authentication mechanism [61]. During synchronization, Contacts and Calendar in both macOS and iOS accept the server suggestion of Basic, and directly send the password encoded in Base64 to the attack server. All these clients by default choose to use TLS, but they all make the vulnerable design choice of accepting our publicly valid certificate by default. Using the proxy approach detailed in the last section, our attack server lets any client to pass the client authentication, and thus these CardDAV and CalDAV clients all proceed with the synchronization functionality.

After the clients are connected, our phishing attack goal is to inject malicious contacts and calendar events. Following the protocol design, after a synchronization, the server gives the client a synchronization token to record the latest synchronization state. In these implementations, we find that a state number is used as the token. During the synchronization, these clients first request the server state number, and only pull new data from the server if the server state number is higher than the one from the last synchronization. If

the server state number is lower, the client makes no action except storing this lower server state number as the latest state. Thus, for the name collision attacker, the attack strategy is to keep a high state number, so if the client-stored state number is lower, the attacker directly triggers the client data pulling request. In case that the attack server state number is lower than the client-stored one, the attacker can wait until the client stores the lower state number after the first round of synchronization, and then start adding phishing contacts or calendar events to trigger data pulling from the client side.

For macOS and iOS Contacts, to increase the attack success rate, the attacker can choose to inject name cards that are likely to be frequently searched and dialed by the user, for example hotline numbers like "Customer Care". In such attack, since the victim voluntarily dials the phishing number, she is more likely to follow the attacker's instruction, for example telling sensitive personal information such as the SSN number or the account password. For macOS and iOS Calendar, the phishing calendar events are best used as the delivery method for other exploits. For example, the events can include links to phishing websites or PDF files with malicious scripts. To increase the success rate, they can masquerade as reminders for popular corporate events such as "Weekly Meeting" and set up to pop up during working hours.

5.6 Defense Discussion

As shown, the widespread client-side name collision vulnerabilities in the exposed service clients cause a wide range of security risks, and thus require immediate attention and remediation. In this section, we propose a set of defense strategies at both the service client software level and the DNS ecosystem level.

5.6.1 Service Level Defense Discussion

Based on the insights from our vulnerability analysis in §5.4, we propose several software design guidelines to secure future clients using DNS-based service discovery.

Integrate and enforce server authentication. As discussed in §6.8 and exhibited in V1 and V4, one fundamental cause for the exposed vulnerabilities is the general lack of server authentication in today's internal network service clients. Since these services are expected to be used in internal networks, e.g., corporate networks, where each user has an internal account, adding PSK-based authentication into the client software design may be most appropriate. To avoid vulnerable choice V4, the implementation needs to strictly enforce mutual authentication during the negotiation. Since the secret is pre-shared, the server has no excuse to not prove that it knows the secret. For the cases in which PSK may be difficult to deploy, e.g., NTP servers or printers, the client should use TLS certificates to verify the sever identity. To avoid vulnerable choice V2, the TLS certificate validation with namespace differentiation proposed next should be used. Adding these server authentication logic is generally beneficial for defending not only name collision attacks but traditional local network server spoofing attacks as well.

Enable namespace differentiation in service discovery and server authentication mechanisms. As discussed in §6.8 and exhibited in V2 and V3, the other fundamental vulnerability cause is the general lack of namespace differentiation in today's internal service clients using DNS-based service discovery. To solve the problem for the service discovery process, the client software developers need to be explicit about which namespace the discovery is expected to occur in and limit the discovery process to only local link if appropriate. At the platform level, we suggest the Bonjour or other Zeroconf platforms to limit their discovery APIs to only perform local link discovery if unicast DNS domain discovery is not explicitly specified. Since these platforms are mainly designed for local link discovery, the default API behavior should not include the unicast domain discovery. These additional resolution requests unnecessarily enlarge the attack surface and allow name collision attacks to happen.

To enable namespace differentiation in the TLS-based server authentication, service clients need additional functionality to differentiate certificates with the same name sub-

ject but from different namespaces. One candidate solution for this may come from a set of recent standards from the IETF, called DNS-based Authentication of Named Entities (DANE) [39]. In DANE, authorities specify the authentication information of their services through the DNS lookup. This natively addresses the name collision vulnerability in the same substrate where it is normally exploited: DNS [244]. More concrete evaluation of this defense solution direction is left as future work.

Application-specific defenses. Internal service clients can also add application-specific defenses. For example, for the malicious library injection attack on the Ruby library discovery service (§5.5.2), a library signing process can be added on the internal servers for the client to check the authenticity of the libraries. Also, for the document leakage attack on the printer discovery service (§5.5.3), the client software can disable the use of invisible characters to prevent the attacker from manipulating printer name display.

5.6.2 DNS Ecosystem Level Defense Discussion

Besides the service level, defense solutions can also be deployed at the DNS ecosystem level, i.e., by relevant parties such as new gTLD registries, victim Autonomous Systems (ASes), and end users. In this section, we discuss how to extend the previous DNS ecosystem level remediation strategies for the WPAD name collision attack [172] to the general form of name collision attacks in this work.

New gTLD registry and victim AS level remediation. Previous work proposes that the new gTLD registries and the victim ASes with high volumes of query leakage can mitigate the WPAD name collision attack based on a set of highly-vulnerable domains (HVDs) [172]. With the HVD set, new gTLD registries can prevent the attack by ensuring that these HVDs are not registered or at least treated more carefully during registration. The victim ASes can filter or alter the queries to these domains before directing them to the public namespace. These remediation strategies are still applicable for the general form of name collision attacks in this work. The previously established HVD set would need to

include the extended service list in addition to the WPAD service.

End user level remediation. At the end user level, the defense mechanisms are more challenging since the leakage may be caused not only by OS-level hardcoding like those in the WPAD name collision attack but also by application-level hardcoding such as by the user account configurations in SIP and XMPP clients. Thus, we propose to design an name collision defense software which can filter out DNS queries to the public namespace if they are only intended to be resolved locally. To perform such filtering, a policy configuration needs to be provided to specify whether the queries to a domain should be "local resolution only". This defense software can be integrated into corporate OS images and IT departments can set such policies during the initial device setup. For example, if the company using the local domain name `comp.ntld` does not own the domain in the public namespace, it can simply set the policy for this domain as "local resolution only". A long-term remediation, though one that could require a significant amount of operational effort, is to convert from using iTLDs to fully qualified domain names (FQDNs) as the root of this threat stems from the use of iTLDs that collide with the globally delegated TLDs.

5.7 Summary

In this chapter, we perform a systematic study of the robustness of the service client design and implementations under name collision attacks for internal network services using DNS-based service discovery. We measure the services exposed to this threat, and perform vulnerability analysis on their clients. Our results show that nearly all the exposed services have popular clients vulnerable, suggesting that the name collision problem broadly breaks common security assumptions made in today's internal network service clients. To demonstrate the severity, we construct exploits and find a set of new name collision attacks with severe security implications. Based on the insights from our study, we propose a series of service software design level solutions, which enables the victim services to actively defend against name collision attacks.

CHAPTER VI

Systematic Discovery and Analysis of Algorithm-level Vulnerabilities in Next-generation Smart Transportation

6.1 Introduction

Connected vehicle (CV) technology will soon transform today's transportation systems. In September 2016, the USDOT (U.S. Department of Transportation) launched the CV Pilot Program as a national effort to deploy, test, and operationalize a series of CV-based transportation systems [129, 35]. In these systems, vehicles and infrastructure are connected through wireless communication, and leverage such connectivity to improve mobility, safety, environmental impact, and public agency operations. These systems are currently under testing in three cities including New York City [129]. To push for a nationwide deployment, USDOT has already proposed to mandate all new light-duty vehicles to equip CV technology [128].

While having a great potential, such dramatically increased connectivity also opens a new door for cyber attacks. To ensure the security of vehicles and transportation infrastructure and the safety of drivers and pedestrians, it is highly important to understand potential security vulnerabilities so that they can be proactively addressed before nationwide deployment.

In this work, we perform the first security analysis on the next-generation CV-based

transportation systems. As a first step, we target the USDOT sponsored design and implementation of a system called Intelligent Traffic Signal System (I-SIG), which performs one of the most basic urban traffic operations, traffic signal control. In this system, real-time vehicle trajectory data transmitted using the CV technology are used to intelligently control the duration and sequence of traffic signals. Such system is fully implemented and has been tested on real road intersections in Anthem, AZ, and Palo Alto, CA, and has shown to achieve a 26.6% reduction in total vehicle delay [82]. In this study, our goal is to identify fundamental security challenges, especially those specific to CV-based traffic control. Thus, we are particularly interested in security problems that are at the signal control algorithm level and are caused by design or implementation choices instead of implementation bugs. The analysis results are expected to serve as a guideline for understanding whether and why the current design or implementation choices in the I-SIG system are vulnerable, as well as providing insights on how to fundamentally secure it before large-scale deployment.

The only attack requirement in our study is that attackers can compromise the vehicle-side devices on their own vehicles or other people's vehicles, and send malicious CV messages to the I-SIG system to influence the traffic control decisions. As reported by previous work, such compromise can be performed physically [224], wirelessly [171], or through malware [236]. Also, we assume that the vehicle certificate system developed by USDOT (§2.3.1) can correctly authenticate all CV messages. Thus, instead of the sender identity, the attack vehicle can only spoof its trajectory data, e.g., speed and location, in the CV messages. To maximize the realism, in this work we assume that *only one attack vehicle* exists in an intersection. This ensures that both our analysis and the discovered security problems have high practical implications.

With such a threat model, the attack goal in our analysis is to create congestion in an intersection. Traffic signal control has been proven to be one of the most cost effective way to increase transportation productivity, and thus it is highly important to ensure its correct

and efficient functioning. This is exactly the reason why the USDOT focuses on deploying the CV-based signal control system [129]. Thus, as the first security study, this work focuses on identifying the congestion creation vulnerabilities, aiming at directly subvert such design goals.

We first analyze the I-SIG system design and identify a set of trajectory data spoofing strategies that can potentially influence the signal control algorithms used in the system. We then enumerate all the data spoofing options for the identified strategies on the I-SIG system to understand the upper bound of the congestion attack effectiveness. A commercial-grade traffic simulation software, PTV VISSIM [95], is used to generate synthetic traffic snapshots as the input to the I-SIG system for this analysis. Based on the results, we analyze the causes for the highly effective attack results, and construct practical exploits under real-world attack resource constraints.

In our analysis, we find that data spoofing attacks are highly effective for the signal control algorithm with the default configurations in I-SIG: the spoofed trajectory data from one single attack vehicle is able to increase the total delay by as high as 68.1%, which completely reverses the benefit of using the I-SIG system (26.6% decrease) and cause the mobility to be even 23.4% worse than that without using the I-SIG system. This is very surprising, since the I-SIG system uses an optimal signal control algorithm that can minimize the total delay of typically over 100 vehicles in an intersection. Thus, the data from a single vehicle should not have such significant influence. We find that this is due to a vulnerability at the signal control algorithm level, which we call *the last vehicle advantage*, meaning that the latest arriving vehicle can determine the signal plan. Fundamentally, we find that this is due to a trade off between security and deployability: due to the limited computation power on the infrastructure-side devices, the developers are forced to choose a less optimal configuration of the theoretically optimal signal control algorithm, which unexpectedly exposes the congestion creation vulnerability.

Even though the deployability issue exists today, this problem may be resolvable in the

future when the infrastructure-side devices have more computation power. Thus, we then analyze whether the I-SIG system is still vulnerable with more optimal configurations. In such scenario, we find that data spoofing attacks can still be highly effective when the I-SIG system is in the operation mode for the transition period, i.e., when the market penetration rate (PR) of the CV technology is lower than 95%. In such period, an algorithm that estimates the status of unequipped vehicles, i.e., vehicles without CV technology, is performed before the signal control algorithm. This is because the signal control algorithm can be very ineffective due to lack of visibility of the unequipped vehicles, but we find that this allows the attacker to manipulate such estimation process to create congestion using spoofed data.

To understand the real-world implications of the identified vulnerabilities, we construct and fully implement the exploits, and evaluate them using simulations on a real-world intersection map. To increase the realism, we videotaped all traffic flows in the intersection for one hour and manually counted the passing vehicles as the input to the simulation model. The results are consistent with our vulnerability analysis, and surprisingly, we find that the attacks can even cause a blocking effect to jam an entire approach. Fig. 6.1 shows an snapshot in the simulation when the blocking effect is created. As shown, in the northbound, southbound, and eastbound approaches, the vehicles in the left-turn lanes spill over and block the through lanes, causing massive traffic jams. In such jamming period, 22% of the vehicles need to spend over 7 minutes for an originally half-minute trip, which is 14 times higher.

Based on our analysis, even though the I-SIG system has shown high effectiveness in reducing traffic delay in benign settings, the current algorithm design and configuration choices are high vulnerable to data spoofing, and even the data from one single attack vehicle is able to manipulate the traffic control to a great extent, causing massive congestion. To addressed these problems, we discuss promising defense directions leveraging the insights from our analysis.

We summarize our contributions as follows:

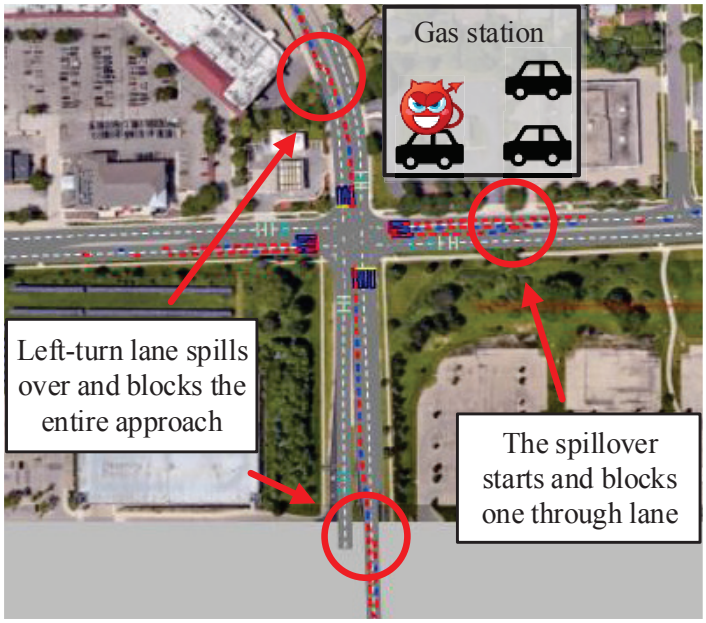

Figure 6.1: The blocking effect created by our congestion attack on a real-world intersection map with real traffic demand. Due to the attack from *one single attack vehicle* parking nearby, in the northbound and southbound approaches the vehicles in the left-turn lanes spill over their lanes and directly block the entire approaches, causing massive traffic jams.

- We perform the first security analysis of a CV-based transportation system, the USDOT sponsored I-SIG system. We formulate the problem with a highly realistic threat model, data spoofing from one single attack vehicle, and analyze the system design to identify a set of data spoofing strategies.
- Targeting the goal of creating congestion, we first perform vulnerability analysis to understand the upper bound of the attack effectiveness. We analyze the causes for the highly effective cases, and find that the current signal control algorithm design and configuration choices are highly vulnerable to data spoofing from even a single attack vehicle. These vulnerabilities exist throughout the full deployment and the transition periods, and can cause the mobility to be even worse than that without using the I-SIG system.
- For the identified vulnerabilities, we construct practical exploits and evaluate them under real-world intersection settings. The results validate the attack effectiveness; further-

more, for the transition period, the attacks can even create a blocking effect that jams an entire approach.

6.2 The I-SIG System

As the first security study on CV-based transportation systems, we target the CV-based traffic control system developed in the DMA program, called Intelligent Traffic Signal System (I-SIG) [132]. In this system, real-time vehicle trajectory data transmitted via DSRC are leveraged to perform more effective traffic signal control in an intersection.

In the DMA program, the development of I-SIG was assigned by USDOT to a team of signal control experts. In this work, we use the latest released version, MMITSS-AZ [37]. This version is fully functional in the field, which has been tested in real intersections in Anthem and Palo Alto and shown high effectiveness with a 26.6% reduction in the total vehicle delay [82]. In this section, we first introduce some key concepts in signal control, and then describe the I-SIG system design.

6.2.0.1 Traffic Control Concepts

As shown in Fig. 6.2, the I-SIG system is operated on an RSU located at an intersection to control the traffic signals. As shown, there are 8 traffic signals, called *phases*. Phases with odd numbers are for left-turn lanes; the others are for through lanes. Each phase is initially configured with the minimum green light lasting time, $t_{g_{min}}$, the maximum green light lasting time, $t_{g_{max}}$, the yellow light lasting time t_y, and the clearance red light lasting time t_r. During the signal control, a phase can be set to turn green and last for a duration t_g. The green duration t_g must be at least $t_{g_{min}}$ and at most $t_{g_{max}}$; this is enforced at the hardware level. After t_g time passes, the phase will be yellow for t_y, and then red for t_r before the subsequent phase turns green, which is for safety purposes since there might be red light runners.

Signal control is performed by setting t_g and the phase sequence, which in combination called a *signal plan*. Fig. 6.3 illustrates a signal plan. Number 1 to 8 are phases, and the green, yellow, and clearance red light periods for each phase are filled with the corresponding colors. As shown, this plan has two phase sequences, called *rings*, operating simultaneously. The phases in the same ring is in conflict with each other, and thus need to be planned sequentially. This is called dual-ring signal phasing, which is the NEMA (National Electrical Manufacturers Association) standard and the most common in the U.S. [272]. For each ring, the phase sequence is broken down to stages. Two types of stages are planned alternatively, one for phase 1, 2, 5, and 6, and another for phase 3, 4, 7, and 8. The phases in the former stage type are in conflict with those in the latter stage type, and thus the phases in the same stage are planned as a whole.

A signal control algorithm needs to follow the rules above, and plan (1) t_g for each phase, and (2) the sequence of phases in the same ring and same stage, e.g., phase 1 and 2 in the figure. A typical goal of such algorithm is to reduce the total delay for all vehicles in the intersection. The delay time for a vehicle spent in an intersection is calculated as the actual time the vehicle spent to pass the intersection subtracting the so-called *free-flow* travel time, meaning that the vehicle is traveling at the speed limit without slowing down or stopping due to red lights or other vehicles. The traffic load for an intersection is called *traffic demand*.

6.2.0.2 System Design

Fig. 6.4 shows the design of the I-SIG system. The BSM messages broadcast by the equipped vehicles are received by a component called trajectory awareness, which maintains the latest trajectory for each vehicle indexed by the vehicle ID in the BSM messages. It also does some pre-processing tasks for the use in the signal planning component, e.g., assigning vehicle data to their requested phases based on the intersection map. The signal planning component listens to the traffic signal status reported by the signal controller,

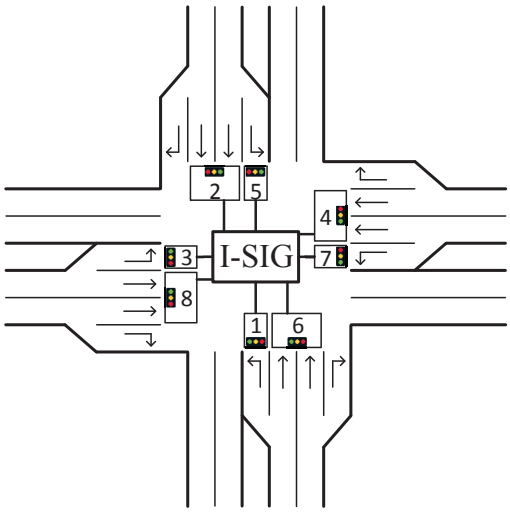

Figure 6.2: The operation scenario for the I-SIG system.

and launches signal planning stage by stage. More specifically, at the beginning of each stage, the signal planning component pulls the pre-processed real-time trajectory data for the vehicles in the intersection, performs the planning, and sends signal control commands to the signal controller. In the current design, the following algorithms are used for signal planning:

The COP algorithm. The signal planning in the I-SIG system uses a dual-ring version of the COP (Controlled Optimization of Phases) algorithm[260, 197]. The input of the COP algorithm is each approaching vehicle's estimated arrival time at the intersection, which is defined as the estimated remaining time for a vehicle to reach the stop bar of its current lane. Based on the arrival time, COP uses dynamic programming to calculate an optimal signal plan with the least estimated total delay. To estimate the total delay, COP first estimates the releasing time for a vehicle based on the queue length at its arrival time. If there is no queue, there is no delay; otherwise, it uses a queuing model to estimate when the queue before the vehicle is cleared. Then, the delay for a vehicle is calculated as its releasing time subtracting its arrival time. If there are no vehicle requesting a certain phase, COP skips this phase in its planning so that the subsequent phases that have vehicle request

Figure 6.3: Illustration of a signal plan. Number 1 to 8 are phases.

can be planned earlier.

In the design, COP can plan for unlimited number of stages until all vehicles in the intersection can be served based on its estimation. Since there might be more vehicles arriving at the phases in the second stage, the I-SIG system only applies the planned signal duration for first stage at each signal control time. Since the operation of the signal controller requires to know what the next phase is after the current phase, the I-SIG system also sets the phase sequence for the next stage at the time of signal control. This means that in Fig. 6.3, the I-SIG system cannot change the order of phases in the first stage, since this is set by the signal control last time. It can change the duration of these phases, and the sequence of the phases in the second stage based on the output of the COP algorithm.

In the current I-SIG system, a limit of the planning stages is configured in COP. This is because in practice the signal planning needs to finish within $t_{g_{min}}$, usually 5-7 seconds, in order to be applied to the signal controller in time. Thus, with computation and memory resource constraints in practice, COP cannot plan with unlimited stages like in its design. With limited planning stages, the COP algorithm may not be able to serve all vehicles. Thus, the current implementation in the I-SIG system first finds the plans with the least unserved vehicles, and then choose the one with the least total delay. As shown later in 6.6.3.1, such planning stage limit unexpectedly leaves the I-SIG system vulnerable to congestion attacks.

Transition period: the EVLS algorithm. If the COP algorithm only optimizes the

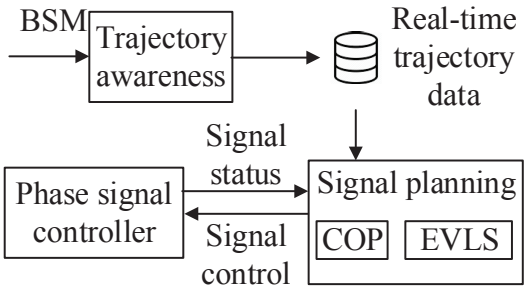

Figure 6.4: The I-SIG system design.

signal plan for the equipped vehicles, its effectiveness is found to be largely reduced if the portion of the equipped vehicles is not sufficiently high, e.g., less than 95% [197]. Since it is estimated that the market penetration rate needs 25-30 years to reach at least 95% [135], the I-SIG system uses an algorithm called EVLS (Estimation of Location and Speed) to estimate the trajectory data of the unequipped vehicles. In the EVLS algorithm, the trajectory data of the equipped vehicles is used for such estimation leveraging multiple traffic models (detailed later in §6.5.2).

Design representativeness and current deployment. The use of COP and EVLS is chosen by the I-SIG designer, the team of USDOT-selected signal control experts, based on a 2015 paper published in Transportation Research Part C [197], a top-tier journal in transportation research. The COP algorithm is chosen because it is very suitable for the CV environment: its input is the arrival time for individual vehicles instead of aggregated traffic information, and thus can best leverage the per-vehicle trajectory data in the CV environment to effectively handle traffic dynamics. As discussed earlier, the EVLS algorithm is developed to overcome the limitation of COP in the transition period. To the best of our knowledge, this is the only design in the transportation literature that is fully implemented and tested on real roads. In the CV Pilot Program, this system is currently under deployment in Tampa [36].

6.3 Threat Model

As illustrated in §6.2, the operation of the I-SIG system involves both infrastructure-side devices, i.e., RSUs and signal controllers, and vehicle-side devices, the OBUs. Previous work found that the traditional transportation infrastructure side devices tend to use weak credentials so that attackers can easily take full control [202]. This is a known problem across many embedded network devices [183] and we assume that the next generation CV-based transportation systems will be fully aware of this problem, and adopt sufficiently strong authentication mechanisms as advised by previous work [202] so that they cannot be easily compromised.

Thus, in this work we focus on the attacks from the vehicle-side devices, the OBUs. More specifically, we assume that the attacker can compromise the in-vehicle systems or OBUs on their own vehicles or others' vehicles so that she can send malicious BSM messages to the RSUs to influence the signal plan. It's important to note that we do not assume that the attackers can spoof the sender identities in the BSM messages. Introduced in §2.3.1, the USDOT will deploy the SCMS system to ensure that all BSM messages are authenticated. Since in this work we are more interested in new security problems specific to CV-based traffic control, we assume that the SCMS system is sufficiently tested and not easily exploitable.

Thus, in our threat model the attack vehicles need to use their true identities so that the sent BSM messages are still correctly signed, but send spoofed vehicle trajectory data, e.g., speed and location, in these messages. This can be achieved in two ways. First, the attacker may directly compromise OBUs by exploiting software vulnerabilities, similar to the demonstrated compromises on other Electronic Control Units (ECUs) [224, 171]. Second, if compromising OBUs is difficult, the attacker can send fabricated CAN messages with spoofed sensor data to the OBUs by compromising other ECUs [224, 171, 180]. Since the attack model includes malicious vehicle owners who have *arbitrary* physical accesses, as long as in-vehicle systems are not vulnerability-free, which has been proved repeat-

edly [224, 171, 236], such compromises are always achievable in practice, just like the smartphone jailbreaking/rooting practices today.

To maximize the realism of our threat model, in this work we assume that *only one attack vehicle* presents in an intersection. Since the COP algorithm targets optimized total delay for all vehicles in an intersection, which normally have over 100 of them, it should be very challenging for the data from one single vehicle to significantly influence the signal planning. However, as shown later, this is actually possible due to several newly-discovered vulnerable design and configuration choices.

The attacker is assumed to have limited computation power to launch the attack, e.g., only using a consumer laptop. More specifically, when using paralleled computation, the attack laptop is assumed to have four processors to execute simultaneously, which is a common specification for consumer laptops such as Macbook Pro. Before attacking an intersection, the attacker is assumed to have performed sufficient reconnaissance and thus already knows (1) the signal control algorithm choices, by testing the algorithm-specific vulnerabilities identified in this work (detailed later), and (2) signal control configurations and the intersection map, by measuring the opened phases, the corresponding signal duration, and the intersection map beforehand.

Since in the CV environment the vehicles are broadcasting BSM messages to the surrounding devices (§2.3.1) and the attack vehicle is in the victim intersection, we assume that the attack vehicle can receive the same set of BSM messages as those in the RSU. Thus, they can run the COP and EVLS algorithms themselves to know the executed signal plans and also estimate the signal plans to be executed, which is also implemented in our exploitation process(§6.7.1).

6.4 Analysis Methodology Overview

In this section, we describe the target attack goals and overview the analysis methodology.

6.4.1 Attack Goals

Targeting the traffic signal control functionality, our analysis considers three attractive attack goals as follows:

Congestion attack. One immediate attack goal is to directly subvert the core design goal of the I-SIG system, total vehicle delay reduction. More specifically, the attacker aims to send spoofed trajectory data to influence the signal plan in order to increase the total delay of other vehicles in the intersection. The attack vehicle is not necessarily in the traffic flows; it might just park nearby, e.g., in a gas station as shown in Fig. 6.1, listening to the BSM messages from other vehicles, and seek chances to launch attacks.

Such attacks can be politically or financially incentivized, e.g., blocking routes to business competitors, like denial-of-service attacks on Internet. Since one attack vehicle can only attack one intersection, to cause larger-scale damage, attackers can form groups to attack consecutive intersections along arterial roads in an area.

Personal gain attack. Besides causing traffic jams, another attractive attack goal is to change the traffic signal plan to decrease the travel time for the attack vehicle so that the attacker can gain unfair advantage at the cost of other vehicles' travel time in an intersection. Different to the congestion attack scenario, the attack vehicle in this attack is inside the traffic flow. While it approaches an intersection, instead of broadcasting BSM messages like normal, it first listens to the broadcast driving data from other vehicles and then launches data spoofing attacks for its own advantage.

Safety attack. Besides influencing the vehicle travel time, attacking traffic signal control can also increase the safety risks of other vehicles in an intersection. More specifically, we target a safety problem specific to the traffic signal control domain, *dilemma zone (DZ)*. A dilemma zone is an area on the high-speed intersection approach, i.e., at least 35 mph, where vehicles at the onset of the yellow light can neither safely stop before the stop line nor proceed through the intersection by the start of the red light [286]. It has been rec-

ognized as a major factor causing rear-end and right-angle crashes [279], leading to more than one million car collisions a year where a quarter of them involves death or injury [38]. On average, it is estimated that each vehicle in the dilemma zone costs $1.13 for the victim and the government [264].

Targeting this severe safety issue, the concrete goal of this attack is to change the traffic signal plan to put as many victim vehicles as possible into the dilemma zone. Like the congestion attack scenario, in this attack the attack vehicle does not need to be in the traffic flow, and might just park near the intersection when launching the attack.

Later in §6.6.2, we will detail how we quantify the attack effectiveness for each of these attack goals in our security analysis.

6.4.2 Analysis Methodology Overview

To understand how vulnerable the current I-SIG system design and implementation is under our threat model, our security analysis consists of the following key steps:

(1) Data spoofing strategy identification. Before analyzing the vulnerability of the I-SIG system, we first need to identify the meaningful data spoofing strategies. Since the attack input is the data in the BSM messages, we analyze the data flow of the I-SIG system starting from the receiving BSM messages to understand how the spoofed data can potentially influence the signal control.

(2) Vulnerability analysis for each attack goal. With data spoofing strategies identified, we then enumerate all the data spoofing options for these strategies on the I-SIG system to understand the upper-bound attack effectiveness. To quantifiably evaluate attack effectiveness, we design quantification metrics for each of the attack goals based on their definitions. To analyze the I-SIG system, we need realistic vehicle trajectory data as input to trigger the signal planning. Since it is impossible to use real vehicles in an intersection due to ethical concerns, our analysis uses a commercial-grade traffic simulation software, PTV VISSIM [95], to simulate traffic patterns with a realistic modelling of driver behav-

iors.

To ensure the generality of this analysis, we create an intersection map with the a generic intersection structure and the common phase configuration in the U.S. We then use VISSIM to generate traffic flows of normal demand following the common practices in the transportation research area. We take snapshots of the vehicle trajectory data in the simulation periodically, which is then used as the input to our analysis. For each snapshot, we run the signal planning in the I-SIG system with and without attack data input, and quantify the attack effectiveness using the defined metrics for the three attack goals.

(3) Cause analysis and practical exploit construction. With the attack effectiveness for all possible data spoofing options quantified, we perform cause analysis for the highly effective attacks to understand why the current signal control is vulnerable. Leveraging the insights, we construct practical exploits under real-world attack resource constraints, e.g., computation power of a normal laptop as described in our threat model (§6.3). As detailed later in §6.7.1, this means that the attacker cannot exhaustively try all possible data spoofing options before making the attack decision; instead, she needs to strategically plan the attack decision process to ensure that she does not miss the attack timing.

(4) Evaluation using simulations with real-world intersection settings. To more concretely understand the practical impact of the constructed exploits, we implement and evaluate these exploits using simulations with real-world intersection settings. We use the map of a real-world intersection with its real phase configurations, and generate traffic flows according to the real traffic demand that we manually measured for one hour on that intersection. Also, compared to attacking individual snapshots in the vulnerability analysis step, in this experiment the attacks are continuously launched for one hour, closely evaluating real-world attack situations.

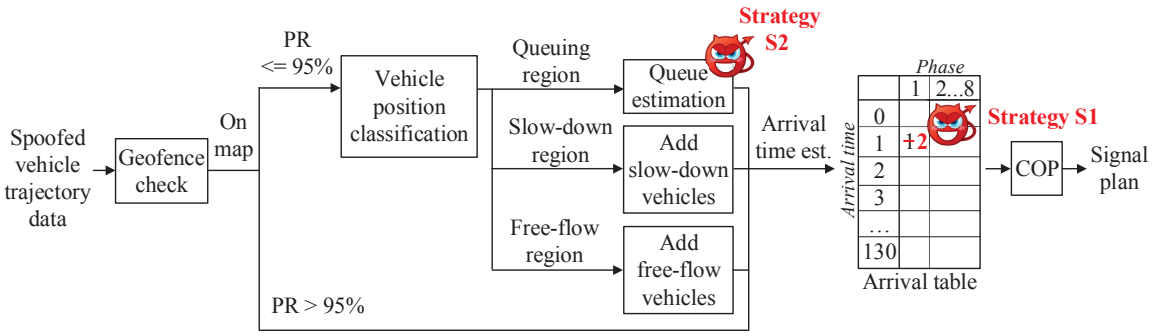

Figure 6.5: The data flow of spoofed vehicle driving data in the I-SIG system. PR means penetration rate.

6.5 Data Spoofing Strategy

As the first step in our analysis, in this section we analyze attack input data flows to identify data spoofing strategies.

6.5.1 Attack Input Data Flow and Direct Spoofing Strategy

Fig. 6.5 shows the attack input data flow in the I-SIG system. When the spoofed vehicle trajectory data is received, it first performs a geofence check, and only accepts the data if its location is within the geographic boundaries of the intersection. Thus, as described in §6.3, the attacker needs to perform reconnaissance to know the geographic coordinates of a targeted intersection, and only generate valid location data to pass the geofence check.

Then, if the configured PR in the I-SIG system is lower than 95%, it is considered a transition period and the attack data are feed into the EVLS algorithm to estimate the trajectory data for the unequipped vehicles. Otherwise, it is considered a full deployment period and the EVLS algorithm is skipped.

A list of vehicle trajectory data entries, including the ones for both the equipped vehicles and the estimated unequipped vehicles if it is during the transition period, is then processed to a structure called *arrival table*. An arrival table is an array with two dimensions: the estimated arrival time and the phases. The arrival time is rounded to seconds. Each array

element at (i, j) is the number of vehicles for the arrival time i at phase j. The first row is for vehicles with zero arrival time, meaning that they are stopped (speed is 0) and waiting in queue.

The COP algorithm computes a signal plan with the optimal total delay for all vehicles based on the arrival table. Thus, the direct goal of the data spoofing attack is to change the values in the arrival table so that it can influence the planning in the COP algorithm. Since each vehicle has a position in the arrival table, the direct data spoofing strategy is:

S1. Arrival time and phase spoofing, for both the full deployment and transition periods. In both the full deployment and transition periods, the attacker can change the speed and location in its BSM message to set the arrival time and the requested phase of her choice and thus increase the corresponding arrival table element by one. In current implementation, the arrival table considers vehicles arriving in no more than 130 seconds. Thus, in this strategy the attacker has 131 (arrival time) $\times$ 8 (phase) data spoofing options.

6.5.2 Spoofing Strategy For The Transition Period Only

To change the arrival table, besides directly spoofing the attack vehicle's own data, the unequipped vehicle estimation process in the transition period is another attractive attack target. Since both the data from equipped and unequipped vehicles are considered in the arrival table, manipulating the estimation results may more significantly influence the signal plan than only changing one vehicle's data in S1.

The unequipped vehicle estimation process, i.e., the EVLS algorithm [197], is detailed in the lower part of Fig. 6.5. As shown, the equipped vehicle data for each lane are first assigned into three regions: (1) queuing region, including vehicles waiting in the queue with zero speed, (2) slow-down region, including vehicles slowing down because of the front vehicles, and (3) free-flow region, including vehicles far away from the queue so that they behave independently. The algorithm first finds the stopped equipped vehicle that is the farthest from the lane stop bar and uses its location as the end of the queuing

region. The slow-down region started right after the queuing region, and the algorithm uses the equipped vehicle's trajectory data to judge whether it is slowing down due to an unequipped front vehicle based on a car-following model. After the slow-down region begins the free-flow region.

After the region assignment, the algorithm first estimates the number of vehicles in queue by dividing the length of the queuing region by the sum of the vehicle length and the headway in queue, which is 6.56 meters in the implementation. For the slow-down region, for each pair of adjacent equipped vehicles, the algorithm inserts unequipped vehicles between them based on the car-following model. Then if the number of vehicles after the vehicle addition in the queuing and slow-down regions is smaller than the number of equipped vehicles divided by the PR, the algorithm adds the remaining unequipped vehicles to the free-flow region.

Among the three regions, we find that manipulating the estimation of the queuing region is most effective. The attacker can just set the speed to zero and set its location to the farthest possible point of the most empty lane within the geofence so that the lane can be fully filled with queuing vehicles after the estimation. In comparison, attacking the slow-down region is less effective since (1) the number of vehicles it can add is fewer since the space headway between moving vehicles in the car-following model is larger than that between queuing vehicles, and (2) the increased delay by adding moving vehicles is no greater than that by adding queuing vehicles, since the queuing releasing process can create more delay as introduced in §6.2. Since the COP algorithm is designed to optimize the total delay, more vehicles to add and more delay time to increase can have more impact on the signal planning.

Thus, the best strategy is attacking the queue estimation:

S2. Queue length manipulation, for the transition period only. In the transition period, the attacker can change the speed and location data in its BSM message to set the location of the farthest stopped vehicle in a chosen lane, and thus add a number of

unequipped queuing vehicles after the original farthest stopped vehicle in the EVLS algorithm. Since this attack only adds queuing vehicles, the change to the arrival table is at the first row. For each phase, the attacker has multiple data spoofing options that can increase the value from by one to by the maximum queue length she can add considering the location of the originally farthest stopped vehicle and the geofence range of the lanes in that phase.

6.6 Vulnerability Analysis

In this section, we use the identified data spoofing strategies to analyze the vulnerability status of the I-SIG system.

6.6.1 Experiment Setup

Traffic snapshot generation. As described earlier in §6.4.2, we use a generic intersection settings for this analysis. The intersection structure, e.g., number of lanes for each phase, is shown earlier in Fig. 6.2. The speed limits for all approaches are 40 mph. Each arm of the intersection is set to about 300 meters from the center of the intersection, which is similar to the DSRC communication range [195]. The $t_{g_{min}}$, $t_{g_{max}}$, t_y, and t_r of each phase are configured according to the recommendations from the Signal Timing Manual [272]. In this generic intersection, we use VISSIM to generate vehicles at 0.7 v/c (vehicle per capacity), which corresponds to the medium traffic demand level [235]. Then we run the I-SIG system, and take vehicle trajectory snapshots every time the I-SIG system needs to perform signal planning.

We run the traffic simulation for each scenario three times, each time lasting one hour with a different random seed following the common practices in the transportation research area [260, 164]. In total, we generated 873 snapshots. These snapshots are directly used when we experiment for the fully deployment period. When experimenting for the transition period, we consider three PR levels, 25%, 50%, and 75%, which is the same as that in

the EVLS algorithm paper [197]. In these experiments, we still use the 873 snapshots, but randomly select a subset of data according to the PR. The random seed for such selection is the same for all experiments with the same PR so that their results are comparable.

Attack data generation. Using these snapshots, we perform vulnerability analysis of the I-SIG system for congestion attacks by trying all data spoofing options for the strategies identified in §6.5. For the full deployment period, only strategy S1 is experimented, and for the transition period, both S1 and S2 are experimented. For each data spoofing trial, a new vehicle trajectory data entry with the spoofed data is added to the traffic snapshot as the attack input.

6.6.2 Attack Effectiveness Quantification

For each snapshot, we run the I-SIG system to get the signal plans with and without attack. Since our goal is to understand the upper bound attack effectiveness, for a snapshot and a chosen data spoofing strategy, we pick the attack result from the most effective data spoofing trial. Based on the semantics of the attack goals, we calculate the attack effectiveness as follow:

Congestion attack. For the goal of creating congestions, we analyze the attack effectiveness by comparing the total delay of all vehicles with and without the attack input in each snapshot. For the signal plans with attack, the total vehicle delay time is calculated after the attack vehicle data being removed. For the transition period, the ground truth unequipped vehicle data (instead of the estimated data) are used in the calculation.

In the delay calculation, we use the same vehicle delay estimation method in the COP algorithm (§6.2). Since this calculation is based on the arrival time estimation, the calculated delay is not the actual delay since the vehicles may not behave as predicted after the snapshot is taken. However, considering that the COP algorithm has a demonstrated effectiveness [82, 197], such estimation is effective for our purpose, i.e., comparing the attack effectiveness among different attack trials. In addition, since our goal is to study the vul-

nerabilities at the signal control algorithm level, using this estimation method allows us to directly evaluate the attack's influence on the signal planning in the COP algorithm. Later in our attack evaluation (§6.7.2), we will directly measure the actual vehicle delay using the ground truth vehicle trajectory in VISSIM.

In this analysis, we quantify the attack effectiveness using three metrics: (1) attack success rate, which is the percentage of the snapshots with the total delay increased under the attack, which we also call *vulnerable snapshots*, (2) average delay increase time, which is the average absolute increase of the total delay under attack, and (3) average delay increase percentage, which is the average ratio of the increased total delay under attack to the total delay without attack.

Personal gain attack. For the goal of decreasing the attacker vehicle's travel time, we need to specify an attack vehicle first for the effectiveness calculation. In our analysis, we randomly pick a vehicle in each traffic snapshot as the attack vehicle, and remove it from the snapshot before attack trials. For the signal plan with and without attack, we only compare the travel delay for the selected attack vehicle. Like in the congestion attack, we use the same vehicle delay estimation method in the COP algorithm for the delay calculation.

In this analysis, we quantify the attack effectiveness using three metrics: (1) attack success rate, which is the percentage of the snapshots with the attack vehicle's delay decreased under the attack, (2) average delay decrease time, which is the average absolute decrease of the attack vehicle's delay under attack, and (3) average delay decrease percentage, which is the average ratio of the decreased attack vehicle's delay to its delay without attack.

Safety attack. As defined in §6.4.1, for this goal the attacker aims to put as many vehicles as possible into dilemma zones. For the signal plan with and without attack, we calculate the total number of vehicles inside dilemma zones using the definition of dilemma zone in the transportation field [286].

In this analysis, we quantify the attack effectiveness using three metrics: (1) attack success rate, which is the percentage of the snapshots with the number of vehicles inside

CV deployment	Full deployment		Transition period			
	100% PR		75% PR			
COP configuration	2-S	5-S	2-S		5-S	
Strategy	S1	S1	S1	S2	S1	S2
Success %	99.9%	96.4%	99.1%	98.3%	83.2%	96.8%
Average delay increase (s) & %	1078.7	162.7	982.2	536.3	167.3	533.9
	68.1%	11.5%	60.2%	32.7%	10.6%	33.5%

Table 6.1: Vulnerability analysis results for congestion attacks in full deployment period and transition period with 75% penetration rate. PR is short for penetration rate. Two-stage planning and five-stage planning in the COP algorithm configuration are denoted as 2-S and 5-S respectively, with the former being the default choice.

dilemma zone increased under the attack, (2) average in-DZ vehicle increase, which is the average absolute increased number of vehicles inside dilemma zone under attack, and (3) average in-DZ vehicle increase percentage, which is the average ratio of the increased number of vehicles inside dilemma zone to that without attack.

6.6.3 Congestion Attack Analayis

In this section, we analyze the results for congestion attacks. The results for the full deployment period is in §6.6.3.1 and those for the transition period is in 6.6.3.2.

6.6.3.1 The Full Deployment Period

The attack results for the full deployment period are shown in Column 2 to 3 in Table 6.1. In these columns, non-successful attacks means that the total vehicle delay is not changed. As introduced in §6.2, the COP algorithm implemented in I-SIG configures a limit on the number of planning stages. By default it uses two-stage planning, which is denoted as 2-S in the table. We first analyze the results with such default configuration:

Two-stage planning results. As shown in Column 2 in Table 6.1, we find that S1 is quite effective in creating congestions: it is able to successfully increase the total delay for nearly all (99.9%) snapshots with as high as 68.1% delay increase. In comparison, the benefit of using the I-SIG system is only a 26.6% total delay reduction [82], but our attack can completely reverse such benefit and cause the traffic mobility to be even 23.4% worse

CV deployment	Transition period							
	50% PR				25% PR			
COP configuration	2-S		5-S		2-S		5-S	
Strategy	S1	S2	S1	S2	S1	S2	S1	S2
Success %	99.4%	99.2%	83.0%	97.4%	99.9%	98.9%	82.0%	91.6%
Average delay	1001.3	536.2	206.6	569.6	1009.2	531.1	295.8	616.7
increase (s) & %	61.4%	33.0%	12.5%	34.6%	60.6%	32.4%	17.0%	34.3%

Table 6.2: Vulnerability analysis results for congestion attacks in transition period with 50% and 25% penetration rates. PR is short for penetration rate. Two-stage planning and five-stage planning in the COP algorithm configuration are denoted as 2-S and 5-S respectively, with the former being the default choice.

than that without using the I-SIG system. This is very surprising, since COP optimizes for the total delay of typically over 100 vehicles in an intersection, and a single vehicle data should not have such significant influence.

Vulnerability cause: last vehicle advantage. By manually examining the signal plan output, we find that for all the vulnerable snapshots, the most successful attack trial adds a spoofed vehicle with very late arrival time. In this work, we call it the *last vehicle advantage*, which is illustrated in Fig. 6.6. As shown, in the signal plan, such late vehicle determines the green light end time for its requested phase. This delays the green light begin time for all the phases after it, and thus increases the delay for the vehicles in these phases. If t_g of the phase with this late vehicle reaches $t_{g_{max}}$, the t_g for the phases before this phase will also extend in order to serve this late vehicle, which further delays the vehicles in later phases. Fig. 6.6 illustrates such attack on phase 2. As shown, due to the spoofed late arriving vehicle, the t_g of all the phases in the first stage are extended in order to be able to serve it, causing long delay to serving time of the vehicles in the second stage.

However, as an algorithm optimizing for the delay of all vehicles, COP should just give up serving this very late vehicle in this green light if serving it costs too much delay for other vehicles. We find that the root cause lies in the planning stage limit when implementing COP in practice. Since the default configuration uses two-stage planning, each phase can only be planned once. Thus, for each phase, planning has to serve all vehicles in this only serving opportunity, causing the planning to be significantly affected by the last

arriving vehicle.

This issue can be alleviated when the COP algorithm is allowed to plan for more stages. For example, if the planning stage limit is four, COP now has two opportunities to serve the vehicles for one phase. Thus, even if a vehicle arrives very late, it can delay serving it to the second opportunity. In this case, vehicles in other phases can be served in the first opportunity and thus is less likely to be affected. Fig. 6.7 shows the percentage of snapshots vulnerable to the last vehicle advantage for the COP algorithm configured with two-stage to eight-stage planning. In the calculation, a snapshot is concluded vulnerable if the most successful attack trial comes from a spoofed vehicle arriving the last in its request phase. As shown, for two-stage planning, nearly all the snapshots can be the most successfully attacked using the last vehicle advantage, and such percentage decreases when more planning stages are configured. The most significant decrease is at four-stage planning, since with such configuration all phases get two serving opportunities. With over four planning stages, the last vehicle advantage is no longer the best trial for any snapshot.

Trade off between security and deployability. Knowing that two-stage planning is highly vulnerable to late arriving vehicles, we are curious why the I-SIG system developers chose to set it as the default value. We contacted the developers and find that it is actually an interesting trade off between deployability and security. As indicated by the developers, they chose two-stage planning because the running time for more planning stages are too high in practice to meet the planning deadline. Since the planning has to finish in $t_{g_{min}}$ (§6.2), which is typically around 5-7 seconds [272], they told us that running three-stage planning on their RSUs takes more than three seconds due to the limited computation power on RSUs, making it too risky to use. Meanwhile, in their testing, they find two-stage planning does not have much planning effectiveness degradation in comparison to five-stage planning, so they choose it as the default value.

They told us that they use the mainstream Savari StreetWAVE RSU [103] and the 95 percentile running time for two-stage planning takes 1.2 seconds. We then use the ratio

between this number and the corresponding running time on our machine to estimate the running time for more planning stages on these RSUs. As shown in Fig. 6.7, our estimation results are consistent with their observations: purely running COP with three planning stages takes around 3 seconds, and with communication delay and the running time of other parts, e.g., the EVLS algorithm, it is indeed risky to use more than two planning stages. In our snapshots without attack, we also confirmed that using two-stage planning only has 6.5% increase in total delay on average than that using five-stage planning. Thus, choosing two-stage planning is indeed a practical choice that trades small planning effectiveness degradation for reliability. However, such choice is found to be highly exploitable leveraging the last vehicle advantage.

Expected to be mounted outdoor in every intersection, RSUs need to be sufficiently reliable with low cost, which leads to performance constraints just like many real-time embedded systems today [241, 231]. While we have shown that such constraints today cause security vulnerabilities, we envision that this situation may be resolvable in future when the infrastructure-side devices have more computation power. Thus, we are also interested in exploring whether the I-SIG system is still vulnerable after the last vehicle advantage is largely mitigated, i.e., with more planning stages configured. Thus, next we perform analysis for the I-SIG system with five-stage planning, with which exploiting last vehicle advantage is no longer the most successful trial (shown in Fig. 6.7).

Five-stage planning results. Column 3 in Table 6.1 shows the results after we configure the COP algorithm to use five-stage planning. As shown, even though the success rate is still high, the attack is much less effective: both the increased total delay time and percentage are nearly 7$\times$ less. Thus, without the last vehicle advantage, the I-SIG system becomes much less vulnerable to the data spoofing from one attack vehicle.

Nevertheless, the attacks can still cause a 11.5% total delay increase on average. Considering that the benefit of using the I-SIG system is around 26.2% total delay reduction [82], the attack result still shows moderate effectiveness. We analyze the causes and

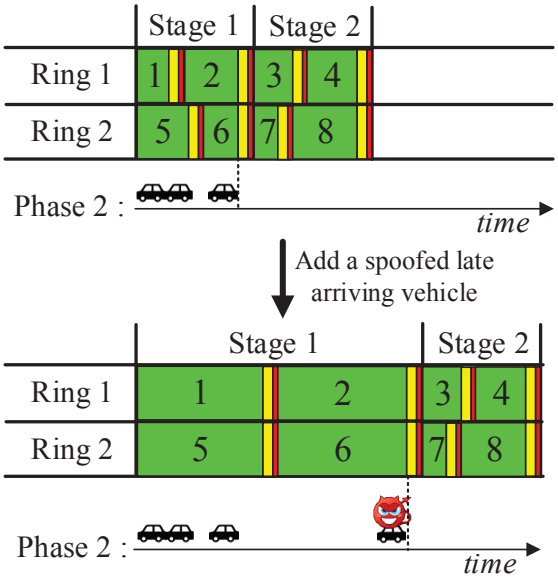

Figure 6.6: Illustration of the last vehicle advantage. By exploiting it, even the spoofed data from a single attack vehicle can significantly influence the signal planning.

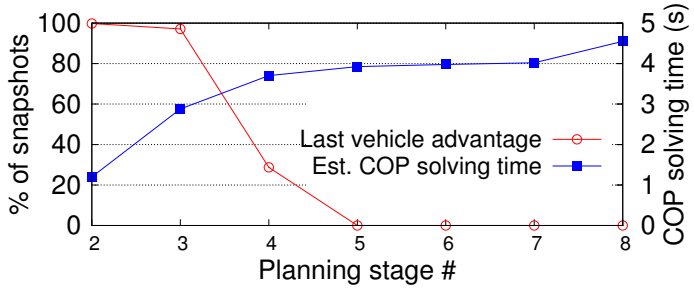

Figure 6.7: Percentage of snapshots vulnerable to the last vehicle advantage and the estimated COP solving time with two to eight planning stages.

find two types of effective spoofing trials:

- Open a skipped phase. If there are skipped phases, the attacker can add the spoofed vehicle to one of them to force the signal planning to open it. Since an open phase needs at least $t_{g_{min}}$ green light time, which is 7 seconds in our generic intersection settings [272], this causes the signal plan under attack to waste the time in serving an empty phase at the cost of the vehicle delay in other phases. If only trying this category of data spoofing options, the total delay increase percentage is 8.9%, which is already very close to that

(11.5%) with all data spoofing options enumerated.

- Extend the green light end time. Besides opening a skipped phase, the most successful data spoofing options are to set the spoofed vehicle arrival time to a few seconds after the original green light end time for a phase. This vehicle needs to wait for a whole planning stage if its serving is delayed to the next serving opportunity, which increases its delay and also the total delay by 20-50 seconds depending on the length of the next planning stage. Thus, in COP it is sometimes more cost effective by just extending the original green light end time for a few seconds to serve this vehicle. However, such extension is usually at most 4 seconds since it is no longer cost effective if the total delay added to the vehicles waiting in the subsequent phases is too much.

The data spoofing options for these two categories in total has around 10.1% in the total delay increase percentage. For the remaining 1.4% difference to that with all options enumerated, we find that the left-out successful trials are highly dependent on the traffic pattern and do not have a clear pattern.

6.6.3.2 The Transition Period

In this section, we analyze the vulnerability status of the I-SIG system in the transition period. The analysis results are shown in Column 4 to 7 in Table 6.1 and column 2 to 9 in Table 6.2. In the transition period, strategy S2 can now be used in addition to S1. Thus, in this section we analyze both strategies for each PR and planning stage configuration. As described in §6.5.2, S2 can add a number of non-existing unequipped vehicles by exploiting the queue length estimation in the EVLS algorithm. Since there are around 100 vehicles in each snapshot, these non-existing vehicles constitute a substantial share of total vehicles in the signal planning. This should trick the COP algorithm into giving more priorities to this big group of non-existing vehicles at the cost of other vehicles' delay.

Overall effectiveness. As shown in the table, for a combination of a PR, a planning stage configuration, and a data spoofing strategy, the attack success rates and the average

total delay increase percentages are 94.0% and 38.2% on average. This show that both strategy S1 and S2 are effective in creating congestion and can completely reverse the mobility benefit of using the I-SIG system. Also, we find that for each combination, the three attack effectiveness metrics are relatively the same, with less than 6% absolute differences in the average total delay increase percentages. This shows that the attack effectiveness is not significantly affected by PR. Next, we perform more in-depth analysis for the attacks on the two-stage and five-stage planning configurations respectively.

Two-stage planning results. Column 4-5 in Table 6.1 and column 2-3, and 6-7 in Table 6.2 show the attack results for the two-stage planning. As shown, strategy S1 can still achieve over 99.1% success rate, and increase over 60.6% in the total delay. We find that the underlying cause is the same as that for the full deployment period: the last vehicle advantage (§6.6.3.1). Since the arrival time to maximally extend the green light time of the phases is not affected by the traffic conditions, the last vehicle advantage can always be reliably exploited for the two-stage planning scenarios regardless of the PR.

Strategy S2, which is newly enabled in the transition period, also shows high effectiveness. For all three PRs, the attack success rates are over 98.3%, and the average total delay increase percentages are over 32.4%. However, the increased percentages are still around 50% less than those using S1. We compare the most successful data spoofing options from S1 and S2, and find that for 99.0% of the snapshots, the best trial from S1 is no less than that from S2. We find that this is because even though adding the non-existing vehicles can indeed cause the signal planning to extend the t_g of a target phase to $t_{g_{max}}$ like S1, last vehicle advantage is able to further cause the t_g of the preceding phases to extend so that the vehicles in the subsequent phases can be further delayed.

Five-stage planning results. Column 6-7 in Table 6.1 and column 4-5, and 8-9 in Table 6.2 show the results for five-stage planning. As shown, since the last vehicle advantage is much less effective for five-stage planning, the success rates and average delay increase percentages for S1 reduce to at most 83.2% and 17% respectively, as opposed to at least

99.1% and 60.2% for two-stage planning. Very similar to the full deployment period, we find that the most successful data spoofing trials are opening a skipped phase and extending green light end time.

Thus, with the last vehicle advantage becoming much less effective, S2 is now the dominating strategy. We compare the results between these two strategies for each snapshot, and find that for 93.5% of the snapshots, the best trial from S2 is no less than that from S1. We then analyze which data spoofing trials in S2 are the most successful. We find that for a certain phase, the best trial is to add the most non-existing unequipped vehicles, i.e., adding a farthest stopped vehicle using S2. If we only try these 8 options (one for each phase), the best trials among them and those among all possible data spoofing options only have 0.009% differences in the average total delay increase percentage. This is expected since adding more non-existing vehicles should gain more priority in signal planning and thus cost more delay to the other vehicles. For very few cases these 8 options fail to hit the most successful data spoofing trial. This is caused by the differences between the estimated and actual arrival time of the unequipped vehicles; if we calculate the attack effectiveness based on the estimated arrival time from the EVLS algorithm, these 8 options are always the best. Thus, in our exploit construction later, we only need to consider these 8 options, which is much less than trying all (usually over 250) possible options.

6.6.4 Personal Gain and Safety Attacks

In this section, we analyze the results for personal gain and safety attacks. Compared to the congestion attack, launching these two types of attacks effectively requires the attackers to have much higher signal control capabilities. For the congestion attack, total vehicle delay increase can be achieved as long as the attack can create a mismatch between the signal plan and the actual traffic demand, e.g., by giving a long green light to a phase with nearly empty traffic. The more significant the mismatch is, the longer vehicle delay time can be causes. Thus, as long as the attack is able to change the original signal plan, the goal

Attack goal	Effectiveness metrics	Attack scenarios		
		S1, 2-S		S2, 5-S
		100% PR	75% PR	75% PR
Personal gain attack	Success %	9.5%	12.7%	38.2%
	Ave. delay	17.4	16.2	13.6
	dec. (s) & %	32.4%	28.9%	29.1%
Safety attack	Success %	0%	0%	9.4%
	Ave. in-DZ	0	0	0.11
	vehicle inc. # & %	0%	0%	132%

Table 6.3: Vulnerability analysis results for personal gain attack and safety attack. PR is short for penetration rate. Two-stage planning and five-stage planning in the COP algorithm configuration are denoted as 2-S and 5-S respectively, with the former being the default choice. For the transition period, we only show the results for 75% PR since the results for 50% and 25% PRs are very similar to those for 75% PR.

of congestion attack is achieved.

In comparison, achieving personal gain and safety attacks are not just about changing the original signal plan. For the personal gain attack, it needs to (1) change the green light for a specific phase that the attack vehicle is in, and (2) change that green light in favor of the travel time of a specific vehicle, i.e., the attack vehicle, not just a mismatch with the overall traffic demand for that phase. For the safety attack, it does not require the attack to be capable of controling a specific traffic signal, but it needs very precise control of the green light length, i.e., with the error of at most 2 to 3 seconds, so that it can put vehicles at the dilemma zone when the green light is off [286].

Thus, the signal control capability requirements for personal gain and safety attacks are the subset of those for the congestion attacks. According to the vulnerability analysis for the congestion attack in §6.6.3, there are two effective attack scenarios that can potentially have the signal control capability needs for personal gain and safety attacks:

- S1 for two-stage planning. As analyzed in §6.6.3, when I-SIG is using two-stage planning, the default configuration, in both full deployment and transition periods, the attacker can exploit last vehicle advantage to control the green light length for a specific phase.
- S2 for transition period. As analyzed in §6.6.3.2, when I-SIG is not vulnerable to

last vehicle advantage, e.g., when using five-stage planning, in the transition periods the attacker can exploit the vulnerability in the queue estimation in the EVLS algorithm to control the green light length for a given phase.

Thus, in our experiments we perform vulnerability analysis for personal gain and safety attacks in these two attack scenarios. Table 6.3 shows a summary of the results.

6.6.4.1 Personal Gain Attack Analysis

In this section, we analyze the results for the personal gain attack under the two attack scenarios: S1 for two-stage planning and S2 for transition period.

S1 for two-stage planning. As shown in Table 6.3, different from the congestion attack, the success rates for personal gain attacks in the attack scenario for S1 is very low: only around 10%. We inspect the attack logs and find that the cause is that last vehicle advantage is in fact not effective for personal gain attack. In two-stage planning, due to the last vehicle advantage, the system is already trying to extend all the green light to serve all the vehicles for all the phase. Thus, for an attack vehicle, if it comes after the planned green light, the system simply cannot extend the green light any more due to hard traffic signal length limitations such as $t_{g_{max}}$ (introduced in §6.2.0.1). If it comes before the planned green light, it needs to reduce the green light lengths of earlier phases, which cannot be achieved with last vehicle advantage. Thus, in the table, none of the successful personal gain attacks comes from lst vehicle advantage.

For the successful cases in Table 6.3, we find that they are dependent on specific traffic patterns. For example, when there is a skipped phase that should happen before the attack vehicle's phase, the attack vehicle can inject a spoofed vehicle to open that phase. In this case, even originally the attack vehicle cannot be served even when the green light of its corresponding phase is $t_{g_{max}}$, the skipped phase opened by the attack vehicle can delay the attack vehicle's phase, thus making it possible to serve the attack vehicle. Even thought these cases do not appear very often, leading to only around 10% success rate, the benefit

of attacking them is substantial: the attacks can reduce the vehicle delay by around 30% as shown in Table 6.3.

S2 for transition period. Compared to the attack scenario for S1, the attack success rate for the scenario for S2 in the transition period is much higher, achieving nearly 40%. For these success cases, the personal gains are all achieved by exploiting the queue estimation vulnerability in the EVLS algorithm. Without last vehicle advantage, the attack vehicles can be left out in green lights and thus adding queues of unequipped vehicles can extend the attack vehicle's phase till the attack vehicle's arrival time to achieve personal gain. At the same time, for the cases when the attack vehicle comes before the planed green light, it can also add queues of unequipped vehicles to increase the importance of its corresponding phase. This can force the algorithm to decrease the green light lengths of the earlier phases and thus serve the attack vehicle earlier. As shown in Table 6.3, personal gain attacks under this attack scenario not only have high success rate, but also have around 30% gains in vehicle delay time. It's important to note that even though the delay decrease percentage for this attack scenario is similar to the attack scenario for S1, the success rate is around 3$\times$ higher, making this attack scenario much more cost-effective.

6.6.4.2 Safety Attack Analysis

In this section, we analyze the results for the safety attack under the two attack scenarios: S1 for two-stage planning and S2 for transition period.

S1 for two-stage planning. As shown in Table 6.3, safety attacks in the attack scenario for S1 is not effective at all: the success rate is 0%. We double checked the attack logs and confirmed that none of the attack cases are able to put more vehicles into the dilemma zone. Similar to the personal gain attacks in this attack scenario discussed in §6.6.4.1, this is mainly because the I-SIG system tries to serve all vehicles due to last vehicle advantage. Thus, in nearly all cases the I-SIG system already serves all vehicles and no victim vehicles can be found even if last vehicle advantage can extend the green light. Theoretically, if there

are vehicles coming within a few seconds after the green light when the green light length is $t_{g_{max}}$, the attack vehicle can use last vehicle advantage to extend the green light to $t_{g_{max}}$ and thus put those vehicles in the dilemma zone. However, such cases may be too rare and thus we are not able to observe them in our experiments.

S2 for transition period. Same as the personal gain attacks for this attack scenario, the attack cases are able to exploit the queue estimation vulnerability in the EVLS algorithm to achieve safety risk increases. However, the success rate is only 9.4%, which is not high. Like we discussed earlier, safety attacks require very precise control of the green light length. Thus, even with the ability to add queuing unequipped vehicles, the attack is still not easy to succeed. Nevertheless, for the success cases, the attacker is able to increase the number of in-DZ vehicles by 132% on average. This more than doubles the safety risks in normal cases, which is thus quite substantial considering the severity of the attack consequences.

6.7 Exploitation Case Study: Congestion Attack

6.7.1 Exploit Construction

Real-time attack requirement. In the last section, to understand the upper bound of the attack effectiveness, we enumerate all data spoofing options, which takes around 24.5 minutes on average on a single core computer. Since we only assume the attacker to have a consumer laptop that has four processors with usually around 3× speedup, this full enumeration takes 8 minutes on average. However, in practice the attack decision needs to be made fast enough so that the traffic condition does not change so much that the attack decision no longer applies.

Thus, to explore the end-to-end exploitability of the identified congestion creation vulnerabilities, in this section we take the real-time attack requirement into consideration and leverage the insights from our analysis in the last section to perform practical exploit con-

struction.

6.7.1.1 Attack Decision Process

To meet the real-time attack requirement, our exploit construction uses a budget-based attack decision process. In this process, the attacker first passively tracks the phase changes. Once the phase in the current stage turns yellow, the attacker waits for 1 second and then triggers the decision process. This is based on our observation that after one second of yellow light all moving vehicles slow down and their trajectories start to stabilize. Since typically $t_y + t_r$ is 6 seconds [272], this gives the attacker up to 5 seconds of decision time.

In the decision period, the attacker first predicts the vehicle trajectory data at the next signal planning time. Like in the trajectory awareness component in the I-SIG system (§6.2.0.2), the attacker maintains a vehicle trajectory database to store data like location, speed, and acceleration for the equipped vehicles based on the received broadcast BSM messages. In the prediction, the attacker assumes that the vehicles maintain their accelerations and thus predicts their speeds and locations after 5 seconds. In this step, the attacker needs to use the intersection map obtained from the reconnaissance step (§6.3) to determine whether a vehicle passes the stop bar of that lane after 5 seconds. If so and the current acceleration value is negative, we predict that it plans to have a hard stop at the stop bar and set the stop bar location as the predicted location.

Next, the attacker needs to make decisions about whether to attack, and if so, what data spoofing option to use. According to our vulnerability analysis, some of the most successful data spoofing trials are related to the signal plan without attack, e.g., the green light end time. Thus, the attacker first runs the I-SIG system for the predicted vehicle trajectory data without trying any data spoofing option. Using the output signal plan and total vehicle delay without attack, the attacker then tries several data spoofing options just like in the vulnerability analysis, and pick the most successful one to use in the actual attack.

Since running the I-SIG system is time consuming, a trial budget is used to ensure that

CV deployment	Full deployment		Transition period					
	100% PR		75% PR		50% PR		25% PR	
COP configuration	2-S	5-S	2-S	5-S	2-S	5-S	2-S	5-S
Strategy	S1	S1	S1	S2	S1	S2	S1	S2
Average trial #	3.8	13.3	3.8	14.7	3.8	23.9	3.6	28.8
Success %	99.8%	84.7%	99.1%	95.6%	99.4%	96.6%	99.8%	91.5%
Average delay increase (s) & %	1077.4	119.8	1057.1	595.3	1061.0	591.7	1008.98	609.6
	68.0%	9.3%	60.0%	35.4%	61.2%	35.1%	60.6%	33.9%

Table 6.4: Practical exploit effectiveness for congestion attacks. PR is short for penetration rate. Two-stage planning and five-stage planning in the COP algorithm configuration are denoted as 2-S and 5-S respectively, with the former being the default choice.

the whole decision process can finish in 5 seconds. Assuming the other parts, e.g., the BSM transition time and other local computation time, take less than 1 second (which typically take much less), we spare 4 seconds in total for (1) running the I-SIG system without attack, and (2) trying the data spoofing options. Since these trials are independent to each other, we use parallel computation to accelerate this part. We first measure the running time for the signal planning without attack, t_{normal}, and then calculate the trial budget as $3 \times \frac{4-t_{normal}}{t_{normal}}$, as the personal laptop with four processors in our lab is measured to have around $3\times$ speedup. With this, the attacker can plan their trials under this budget. The detailed budget-based trial strategies for different attack scenarios are described in the next section.

Based on the trial results, the attacker finds the data spoofing option with the highest total delay increases. If such increase is larger than zero, the attacker uses the corresponding data spoofing option to construct the BSM message and broadcast it out. Otherwise, the attacker does not attack.

6.7.1.2 Exploitation Strategy

In this section, we describe the exploitation strategies, i.e., the budget-based data spoofing trial strategies, for different combinations of PRs and planning stage configurations. Table 6.4 summarizes the attack effectiveness for the constructed exploits in this section.

E1: Congestion attack for two-stage planning:

(1) In the first stage, if there are no skipped phases, try the data spoofing option with the latest arrival time for any of the two latter phases in stage 1, and then jump to (3). Trying the latter phases is because their latest vehicles are able to further extend the t_g of the two former phases to $t_{g_{max}}$.

(2) In the first stage, if there is a skipped phase, try the data spoofing option with the latest arrival time for this phase, and then jump to step (3). If there are two skipped and the budget allows more trials, try both and then jump to step (3). This is because opening an originally skipped phase can cause more total delay increase as explained in §6.6.3.1.

(3) In the second stage, if there are no skipped phases, try the two data spoofing options with the latest arrival time for the two former phases. If the budget allows more trials, try the latest arrival time for the two latter phases. Try the former phases first is because their latest vehicles can cause phase sequence switches to further increase the delay.

(4) In the second stage, if there is a skipped phase, try the data spoofing options with the latest arrival time for this phase. If the budget allows more trials, try the latest arrival time for the former phases, and then the latter phases. If there are two skipped phase, try the two data spoofing options with the latest arrival time for these two phases.

As introduced in §6.2.0.2, at each planning time only the planned duration for the first stage is immediately applied. Thus, in the above strategy we prioritize the attacking on the first stage so that the attack has an immediate effect. Also, in this strategy we only consider at most two skipped phase since we do not observe any snapshot in our analysis has more than two skipped phases under the normal traffic demand.

E2: Congestion attack for five-stage planning in the full deployment period:

(1) If there are skipped phases, try any data spoofing option for each of these phases. If the budget is not enough, prioritize the ones in the earlier stages.

(2) Try the data spoofing options b_g seconds after the originally green end time for each open phase. For the first time entering this step, b_g is 1. If the budget is not enough, prioritize the ones in the earlier stages.

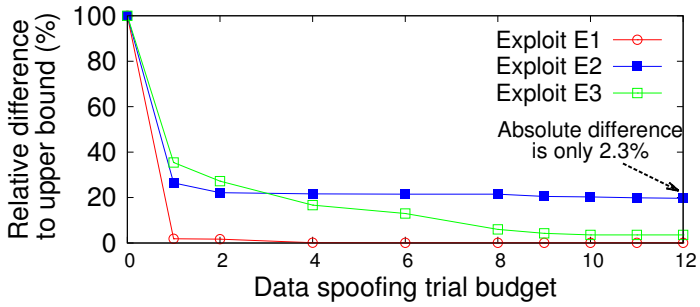

Figure 6.8: Relative differences between the average delay increase percentages using the three exploits with limited trial budgets and those by trying all possible options.

(3) If the budget allows more trials, repeatedly try (2) with b_g being increased by 1 each time until the budget is used up.

E3: Congestion attack for five-stage planning in the transition period:

(1) For the through phases, try the data spoofing options that add Q_p non-existing queuing unequipped vehicles for each phase p. If the budget is not enough, prioritize the ones in the first stage.

(2) For the left-turn phases, try the data spoofing options that add Q_p non-existing queuing unequipped vehicles for each phase p. If the budget is not enough, prioritize the ones in the first stage.

(3) If the budget allows more trials, repeatedly try (1) and (2) with Q_p being decreased by 1 each time until the budget is used up.

In this strategy, we prioritize the through phases since their lanes are longer than those of the left-turn phases, and thus has much (usually twice) larger Q_p.

Fig. 6.8 shows the attack effectiveness of these three exploits with different trial budget on the snapshots in the vulnerability analysis. In the figure, the attack effectiveness metric is the average total delay increase percentage. As shown, for E1, only 4 trials are need to reach the upper bound attack effectiveness, i.e., the one by trying all possible options. For E2, the attack effectiveness converges quickly after using 2 trials, and then decreases very slowly when b_g increases with more available budget. At the tail, the relative difference to the upper bound attack effectiveness is around 20%, but since the upper bound is only 11.5%,

it only has 2.3% absolute difference. As discussed in §6.6.3.1, the best trials responsible for such difference highly depend on specific traffic patterns. For E3, only 8 trials are need to reach the upper bound attack effectiveness, which is consistent with the discussion in §6.6.3.2.

We implement this budget-based trial strategies, and evaluate their effectiveness on the snapshots in the vulnerability analysis. In this experiment, we use the running time without attack for each snapshot to dynamically choose trial budget. The results is shown in Table 6.4. As shown, even though two-stage planning is much faster than five-stage planning, the maximum trial number needed for E1 is only 6 so the average trial number is 3.6-3.8. For five-stage planning scenarios, in the worst case the attacker can at most try 13.3 options due to the real-time attack requirement. This is already much less than trying all possible options, which needs nearly 1000 trials for S1 and around 250 options for S2. Nevertheless, our trial strategies show high effectiveness with less than 2.2% difference to the upper bound attack effectiveness.

6.7.2 Attack Evaluation

In this section, we implement and evaluate the constructed exploits using simulations with real-world intersection settings.

6.7.2.1 Evaluation Setup

Real-world intersection settings. In this evaluation we use the map of a real-world intersection with its real phase configurations. The intersection map is shown in the screenshot in Fig. 6.1. Compared to the generic intersection structure, this intersection has different speed limits on each approach. The speed limits are 30 mph, 35 mph, 40 mph, and 45 mph for southbound, eastbound, northbound and westbound respectively. Only northbound approach has dedicated right turn lane, while in other approaches the right turn lane is shared with the through lane. The map range of the eastbound approach is only extended

to 220 meters because of the existence of a close-spaced upstream intersection.

Real-world traffic demand. To increase the practicality of our analysis, we use the real traffic demand for this intersection in our VISSIM configuration. To measure such demand, we went to the intersection and videotaped the traffic in the intersection on May 16th, 2017, 4-5 pm. Based on the videos, we manually counted the passing vehicles for each lane, and calculated the traffic demand of each approach and the turning ratio for each lane (the possibility of turning left or right for the vehicles), as the input to the VISSIM traffic model.

Experiment setup. In the experiment, the I-SIG system and attack program can receive the BSM messages within their DSRC ranges. The DSRC ranges for all approaches are set to the normal value, 300 meters, except the one for the eastbound approach is 220 meters as its lanes are shorter. On the attacker side, the BSM messages are used in the attack decision process detailed in §6.7.1.1. After that, the BSM message sent with the spoofed data is merged with the other BSM messages. The I-SIG system uses these BSM messages, which may or may not have the attack message, to perform the signal planning and then use the plan to control the traffic signals in VISSIM.

For each combination of PR and planning stage configuration, we run the experiments for one hour three times, each with a different random seed, based on the aforementioned real-world traffic demand. In this experiment we launch the attack continuously for every signal planning in the I-SIG system. This is different to the experiments in the vulnerability analysis in which the attacks are launched individually to each snapshot. In comparison, such continuous attacking is closer to real-world attack situations. As we will show later, this is able to create a cumulative attack effect and thus create even more congestion than that in the vulnerability analysis.

Attack effectiveness measurement. In the evaluation we directly measure the vehicle travel delay using each vehicle's trajectory output by VISSIM. To calculate the per-vehicle delay, we subtract the free-flow travel time, i.e., the travel time at the speed limit, from the

vehicle's actual travel time. Then the total vehicle delay is calculated as the sum of the per-vehicle delay for all vehicles generated in the experiment. In the VISSIM simulation, for the same random seed the vehicle generated with the same ID has exactly the same initial data, e.g., the same generation time and the same initial speed and location. Thus, both the total vehicle delay and the per-vehicle delay for experiments with and without attack are comparable.

CV deployment	Full deployment		Transition period					
	100% PR		75% PR		50% PR		25% PR	
COP config.	2-S	5-S	2-S	5-S	2-S	5-S	2-S	5-S
Exploit	E1	E2	E1	E3	E1	E3	E1	E3
Ave. delay inc. (s) & %	68435.4	4695.9	64008.0	187746.0	66797.4	197410.0	56618.0	146685.0
	66.7%	4.8%	61.7%	181.6%	64.2%	193.3%	46.2%	133.2%

Table 6.5: Evaluation results for the practical exploits. PR is short for penetration rate. Two-stage planning and five-stage planning in COP are denoted as 2-S and 5-S, with the former being the default choice.

6.7.2.2 Results

The results are summarized in Table 6.5 and analyzed below:

E1 and E2. Column 2, 4, 6, and 8 show the results for E1. As shown, E1 is able to increase more than 60% of the total delay for all cases expect when the PR is 25%. These results are consistent with those in Table 6.4, showing high attack effectiveness. When the PR is 25%, we find that the errors in the unequipped vehicle estimations in the EVLS algorithm are greater than those in the generic intersection settings, causing the attack effectiveness to decrease. Nevertheless, the total delay increase percentage is still very high (46.2%): for a vehicle, a one-hour trip now takes nearly one and half hours, showing a significant decrease of the transportation mobility.

The results for E2 are shown in Column 3. As shown, the attack effectiveness is only 4.8%, which is around 50% lower than that in the vulnerability analysis. We find that this is because both categories of the successful data spoofing trials in §6.6.3.1 can be largely affected by errors in the vehicle trajectory data prediction in our attack decision process

(§6.7.1.1). For the one that opens the skipped phase, any legitimate vehicle requesting that phase in 5 seconds nullifies the attack effect. For the one that extends the green light end time, the original green light end time can vary after 5 seconds due to changes in the arrival table. Among the three exploits, E2 is the most dependent on traffic conditions and thus more sensitive to the errors in our prediction. Considering that it also has the least attack effectiveness, E2 is thus the least attractive exploit among the three.

Exploit E3. The results for E3 are shown in Column 5, 7, and 9. Surprisingly, we find that these attacks are much more effective than those in the vulnerability analysis: when the PRs are 75% and 50%, the average delay increase percentages are 181.6% and 193.3%, which are over 5× more than those in the vulnerability analysis. The increase for the 25% PR scenario is a bit lower, but is still around 4× more.

The lane blocking effect. We find that such significant increase is because continuous attacking is able to cause the attack effect to accumulate, and thus greatly escalates the attack effectiveness. More specifically, in five-stage planning, since the planning is allowed to delay serving some vehicles in the current stage for more optimal long-term benefit, these vehicles are attacked for another time in the next signal planning time. If the vehicle is near the end of the queue, it can be attacked multiple times. Since in the vulnerability analysis we only estimate the effectiveness for attacking once, such cumulative attack effect causes the average total delay to significantly increase in comparison to that in the vulnerability analysis. Such cumulative attack effect does not exhibit for the two-stage planning scenarios, since two-stage planning only has one serving opportunity for each phase and it is not allowed to delay serving any vehicle.

We further find that such cumulative attack effect is able to cause an even higher level of congestion, which *can block an entire approach, causing massive traffic jams*. This is because with such effect the queues in the left-turn lanes cannot be effectively released and thus begin to increase with time. Since the left-turn lanes are shorter in nature, at a certain point the queues start to spill over to the through lanes and block the through lane. This

causes the through lane to start queuing after the spilled-over left-turn vehicles. With both the real queuing vehicles and the non-existing unequipped vehicles added by our attack in the through lanes, the COP algorithm sees more than 80 vehicles queuing in the through lanes and thus only gives the spilled-over left-turn phase the minimum green light time. Thus, the left-turn phase can now only release the fewest possible vehicles. When some spilled-over vehicles finally enter the left-turn lane, the following left-turn vehicles quickly block the through lanes again.

Such blocking effect is shown earlier in Fig. 6.1, which is a screenshot taken at the 1785.80 second in the VISSIM simulation for one of the three random seeds and the 75% PR. Note that such spillover and blocking effect always appears on at least one approach in all E3 experiments. As shown in the figure, in both the northbound and southbound approaches, the left-turn vehicles spill over and block the through lanes, causing long queues in the approach. In the real-world traffic demand we collected from 4 to 5 pm, the northbound approach has the most left-turn vehicles and thus is the earliest to block and thus have the longest queue at the time of the screenshot.

Fig. 6.9 shows the average delay every one minute with and without attack in the northbound approach in this experiment. As shown, the delay under attack usually has an increase when the delay without attack increases. This is because when the approach is more congested without attack due to a temporarily higher demand, the congestion attack can further escalate such congestion. As shown, at around second 1125, such higher demand is leveraged to create the blocking effect, and thus the congestion level is significantly increased. After 10 minutes, the spillover is finally cleared, but in as short as 1 minute, the blocking effect happens again. In the figure, we can see such repeated blocking effect till the end of the experiments. In the traffic jam period starting from second 1125 till the end, nearly 600 vehicles arrive and around 50% of them need to spend nearly three minutes for an originally half-minute trip (27.7 seconds on average), and around 22% need to spend over 7 minutes, which is 14 times higher. This means that for these 22% of vehicle, if the

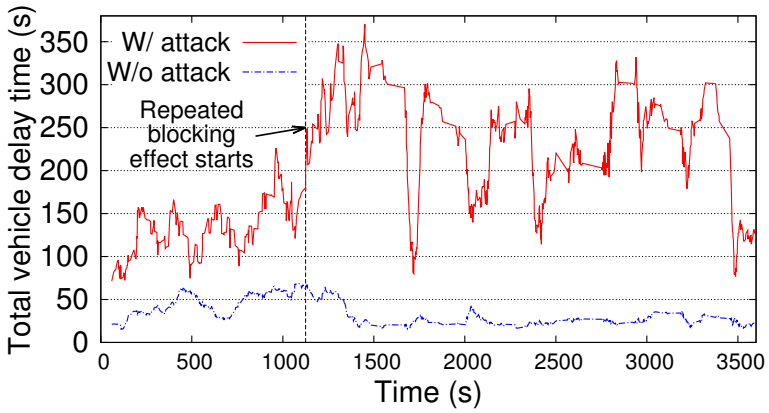

Figure 6.9: Average vehicle delay every one minute with and without attack. The repeated blocking effects start at around second 1125.

trip involves a series of intersections, i.e., in a corridor, a 10-minute trip can now cost over 2 hours.

6.8 Defense Discussions

As shown in our study, even though the I-SIG system has shown high effectiveness in benign settings, the current algorithm design and configuration choices are highly vulnerable to data spoofing. To proactively addressed these problems before larger-scale deployment, this section discusses defense directions based on the insights from our analysis.

Robust algorithm design for the transition period. As concretely shown in our evaluation, the most effective congestion attack is on the transition period: the total delay increase percentage is nearly 200%, and by continuously attacking for less than 20 minutes, it is able to trigger the blocking effect on an entire approach, causing massive traffic jams. According to the current I-SIG system design, such problem can only be largely alleviated when PR reaches more than 95%. This is thus the most urgent problem in the current I-SIG system design: the market penetration rate of CV technology needs to start somewhere, and thus it inevitably needs to go through a transition period. Even after all new light-duty vehicles are mandated to install OBUs, which is exactly what the USDOT is proposing now, there are still heavy duty vehicles and old vehicles on the roads. As estimated by the

USDOT [135], it may take 25-30 years to reach a 95% PR after it starts such mandate. Thus, if such system cannot handle the security challenges for the transition period, it is not robust enough to get the larger-scale deployment even started in practice.

Fundamentally, this is caused by the lack of a sufficiently robust signal control algorithm for the transition period. As introduced in §6.2.0.2, the COP algorithm is a suitable design choice for the CV-based signal control, but it is only optimal in the full deployment period. To ensure that the I-SIG system can still be effective when PR is low, the current design tries to infer the unequipped vehicle data to solve the dilemma. However, if such inference is not robust, it can be greatly manipulated for malicious purposes — which is exactly what we have uncovered in this study. Since the amount of vehicle data input is much less than that in the full deployment period, any signal control algorithm for the transition period is inherently more sensitive to data spoofing attacks, making it fundamentally more challenging to ensure the robustness. Considering that the transition period is unavoidable and may last as long as 30 years, we believe that this calls for a joint research effort among both the transportation and the security communities to design effective and robust signal control algorithms specifically for the transition period.

Performance improvement for RSUs. As introduced in §6.2.0.2 and analyzed in §6.6, the arrival time based signal planning in the COP algorithm is very suitable for the CV-based signal control, and when given enough computation power, such planning is indeed very hard to be maliciously influenced by small amounts of spoofed data in the full deployment period. Unfortunately, due to the limited performance in today's RSUs, the I-SIG system has to use a suboptimal implementation of the COP algorithm, which is found to introduce the last vehicle advantage, allowing the data from one single attack vehicle to significantly influence the signal control. Because of this, even if the security challenge for the transition period is addressed, the I-SIG system can still be greatly manipulated by data spoofing attacks. Thus, it is important to improve the performance of today's RSUs so that more optimal configurations can be used in the traffic control. Such improvement

can be at both the software level, e.g., code optimization, and the hardware level, e.g., CPU and memory upgrades. Such performance improvement is generally beneficial since more computation capabilities can help better balance the trade-off between security and performance.

Data spoofing detection using infrastructure-controlled sensors. Besides improving the robustness at the control algorithm level, another defense direction is to detect and filter the BSM messages with spoofed data on the infrastructure side. Since these messages are still correctly signed, such defense must rely on data validity checks. Unfortunately, in the current design, the I-SIG system only has one data source about the attack vehicle — the attacker-controlled trajectory data via BSM messages [82]. Thus, any data validity check methods based on this are unlikely to be effective since the attacker can strategically control the spoofed data so that the vehicle trajectories appears perfectly normal.

Thus, to ensure high effectiveness, data spoofing detection on the infrastructure side needs to rely on data sources that attackers cannot easily control, e.g., infrastructure-controlled sensors, to cross validate the data in BSM messages. We find that there are actually existing infrastructure-side sensors ready to be used for this purpose. For example, the vehicle detectors buries underneath the stop bar of each lane was used to measure aggregated traffic information in today's traffic control. Even though they are less useful in the CV environment, they may be re-purposed to help detect data spoofing, which may be a cost effective solution since they are installed already. If such aggregated data is not sufficient, the infrastructure side may need to install sensors with more informative data, e.g., cameras. One challenge in this direction is how to best leverage different types of infrastructure-side sensors to design a detection system that is both accurate and hard to evade, which we leave as future work.

6.9 Summary

In this chapter, we perform the first security analysis of the emerging CV-based signal control system. Targeting a highly realistic threat model, data spoofing from one single attack vehicle, we perform vulnerability analysis and find that the current signal control algorithm design and configuration choices are highly vulnerable, which can be exploited to launch congestion, personal gain, and safety attacks. The evaluation results under real-world settings validate the attack effectiveness and show that the attacks can even create a blocking effect that jams whole approaches. Defense directions are then discussed leveraging the insights.

This work serves as a first step to understand the new security problems and challenges in the next-generation CV-based transportation systems. It is expected to inspire a series of follow-up studies, including but not limited to (1) more extensive evaluation with different intersection sizes and traffic patterns, (2) more extensive analysis considering other CV-based transportation systems, algorithms, and security implications, (3) more concrete defense system design and evaluation.

CHAPTER VII

Conclusion and Future Work

7.1 Conclusion

In this dissertation, I develop a series of proactive vulnerability discovery and assessment approaches that can systematically discover and assess security challenge in two most fundamental capabilities in any smart, connected system: network stack and smart control. More specifically, my research is able to demonstrate that static/dynamic program analysis and network measurement can be used to make the following two categories of research contributions:

(1) Systematically identify new code-level and network-level security challenges in smart, connected systems. In this category, the contributions made in my dissertation research include:

- *Discovering and analyzing WPAD name collision attack (US-CERT alert TA16-144A [127]).* In the network stack, we discover a new MitM attack vector exposed in the new gTLD era called WPAD name collision attack. We then conduct the first systematic vulnerability study of this attack vector by systematically characterizing the problem severity and the vulnerability causes using network measurement. Through this analysis, we identify that the cause of the new security problem is that the recent delegation of new gTLDs unexpectedly breaks the implicit namespace isolation assumptions made in the internal DNS namespaces. This is a *network-level security problem newly exposed in the new*

gTLD era, which is fundamentally challenging to solve due to the lack of coordination in a distributed system like the Internet.

- *Analyzing client-side name collision vulnerability.* We generalize the WPAD name collision attack to a new class of attacks on the broad set of internal network services using DNS-based service discovery, and perform the first systematic study of the vulnerability status, named client-side name collision vulnerability, of internal service software under this new class of attacks. We first measure and collect the client implementations of the affected internal service today, and then perform vulnerability analysis using dynamic program analysis. We find that nearly all the exposed services have popular clients vulnerable due to several common design and implementation choices, which suggests that the name collision attack threat model broadly breaks common security assumptions made in the service clients today. Through our subsequent vulnerability cause analysis, we find that such widespread vulnerability exposure is caused by several *code-level security challenges newly exposed in the new gTLD era*, e.g., lack of namespace differentiation. Based on the insights, we propose a set of service software level solutions.

- *Discovering and analyzing algorithm-level vulnerability in Connected Vehicle (CV) based smart signal control.* We perform the first security analysis of a CV-based transportation system, the USDOT sponsored I-SIG system. Targeting a highly realistic threat model, data spoofing from one single attack vehicle, we perform vulnerability analysis and find that the signal control in the system can be manipulated greatly by data spoofing from even a single attack vehicle, which can be exploited to launch congestion, personal gain, and safety attacks. We analyze the vulnerability causes, and find that the current signal control algorithm design and configuration choices in the I-SIG system are highly vulnerable to data spoofing attacks, which is a *code-level security challenge newly exposed in the emerging CV-based smart transportation systems*. To addressed this challenge, we discuss several defense directions aiming at increasing the algorithm robustness and mitigating spoofing attacks.

(2) Gain insights about problem severity to address design trade-offs in the defense solutions. In this category, the contributions made in my dissertation research include:

- *Designing PacketGuardian, the first automated detection tool for packet injection vulnerabilities.* In the network stack, to detect packet injection vulnerability, the current solution is to manually inspect the protocol code bases and apply case-by-case patches. However, due to the complex nature of the problem, new packet injection vulnerabilities are still emerging. To stop this recurring problem, we build a static program analysis tool, PacketGuardian, to systematically examine network protocol implementations to detect such vulnerability. In the design, we face a unique challenge due to *the design trade-off between detection precision and recall*: to accurately detect leaks of secrete protocol states, implicit data flows need to be analyzed; however, performing implicit data flow analysis causes high volumes of false alarms, which is thus a commonly excluded feature. To address this challenge, we find that only a specific type of implicit data flows is highly exploitable for packet injection attacks in practice, which we call *attacker-controlled implicit information leaks*. Thus, we design our tool to prioritize the detection of such highly-exploitable implicit data flows, which effectively reduces false alarms without compromising tool effectiveness. Using PacketGuardian, even though the recently-reported packet injection vulnerabilities have been patched, we are still able to uncover 17 new ones in the Linux kernel TCP implementation with confirmed exploitability.

- *Designing Highly-Vulnerable Domains (HVDs) for name collision attack defenses*. To defend against WPAD name collision attacks, new gTLD operations and vulnerable ASes need to know the attack surface, i.e., the set of vulnerable domains, for domain registration scrutinization and query leakage filtering. However, defining the attack surface has a unique challenge due to *the design trade-off between defense effectiveness and economic incentive*: if including all domain names seen in the query leakage into the attack surface, large numbers of popular domains are blocked from registration, which hurts the economic incentive for operating new gTLDs. To address this challenge, we perform analysis using

network measurement and find that most domain names in leaked queries are transient and low-volume, which are thus both of low value and hard to exploit in practice. Leveraging this insight, we propose a more useful definition of attack surface, called *Highly-Vulnerable Domains (HVDs)*, which are the domains that persistently expose many victims. Using empirical data analysis, we show that using HVDs as the attack surface can achieve a much better balance of defense effectiveness and economic incentive: using the list of HVDs, 61% of new gTLD operators only need to block less than 10 domains in total and at the same time more than 97% of the leaked queries are protected.

7.2 Future Work

Following my dissertation research, there are numerous future directions worth further investigation:

Critical domain: Systems security in transportation. Transportation systems and automobiles today will be soon transformed profoundly due to the recent advances in Connected Vehicle (CV) and Autonomous Vehicle (AV) technologies. To secure the security of such safety critical systems, it is highly desired to perform further research into the following two subdirections:

- *Connected Vehicle (CV) systems security.* Following up my dissertation research on the first security analysis of the CV-based traffic signal control system (§VI), this research direction can be further explored from two perspectives: (1) CV application security, by more broadly and more systematically analyzing the security of the released USDOT-sponsored CV-based transportation system and application prototypes, and (2) CV communication security, by analyzing the security of the CV network protocol stack design and implementations on CV devices. The analysis insights can be used to develop practical defenses at both the infrastructure and vehicle sides. For example, leveraging insights from the current analysis results in §VI,

developing data spoofing detection mechanisms leveraging infrastructure-controlled data sources can be one of the promising defense directions. Since this direction is interdisciplinary by nature, collaborations between security and transportation researchers may be necessary in order to combine the expertise from both domains for effective problem solving.

- *Autonomous Vehicle (AV) systems security.* To enable intelligent driving, all the vehicle subsystems including critical control systems such as brake and acceleration are now controlled by software, making software quality, especially reliability and security, a concern that is more critical than ever. Due to the fact that (1) such control is managed by a complex distributed systems involving tens of microcontrollers, and (2) software development in in-vehicle systems is commonly outsourced to third-party sources [171], it is especially challenging to secure the software stack in such systems. As demonstrated in this dissertation, static/dynamic program analysis techniques can be used to tackle such code-level security challenges. For example, one promising starting point can be developing a security analysis framework using static and dynamic program analysis for open-source AV systems such as Baidu Apollo [18] and Autoware [17].

Software control algorithm security. In smart, connected systems, optimization algorithms and machine learning models are used popularly to enable various smart control capabilities, e.g., the smart traffic signal control studied in this dissertation research (§VI) and machine learning based autonomous driving [138, 49, 60]. Such algorithm-based control in smart, connected systems introduces two new challenges in software security analysis. First, it is unclear how to ensure soundness in the software analysis on algorithm implementations. Current security analysis of optimization algorithm and machine learning model implementations mainly uses dynamic program analysis [176, 249]. However, dynamic analysis cannot fully eliminate false negatives and thus cannot provide soundness guarantee of the absence of a particular security vulnerability, which is highly desired for

smart, connected systems in safety critical domains. However, classic sound software analysis methods, e.g., static program analysis, cannot be directly applied since in algorithm implementations control decisions are made based on numerical computations in data flows instead of explicit logics in control flows. To address this challenge, two directions may be worth exploring: (1) design more fine-grained data flow analysis to directly analyze the implicit control logic in data flows, and (2) design program analysis based approaches to first extract a model from the algorithm implementations, and then use model-based security analysis methods in algorithm analysis and machine learning fields.

Second, besides analyzing the security of the algorithm implementation itself, it is unclear how to incorporate its domain-specific usages into the security analysis. In smart, connected systems, these domain-specific usages, e.g., the pre- and post-processing steps, are also critical since (1) they have a direct impact on the end-to-end exploitability of potential vulnerabilities in the algorithm implementations, e.g., the discovered vulnerability in the EVLS algorithm in §VI of this dissertation, and (2) these usages themselves may cause security problems, similar to the API misuse problem in TLS/SSL clients [184, 196]. The key challenge in this direction is how to effectively perform vulnerability analysis of the whole model usage work flow, which is usually a complicated process involving various types of data processing. For example, in autonomous driving, camera input is first in a machine learning based perception module, and then in a dynamic programming based path and speed planning module, and then in a rule-based control module. To address this challenge, one potential direction is to first explore the possibility of modeling the data processing steps before and after the machine learning model using program analysis techniques, and then design an extension of the existing analysis methodology with the modeled data processing steps plugged in.

Anomaly detection for IoT/CPS using physical properties. In IoT/CPS systems such as smart home and smart transportation systems, the behaviors of the IoT/CPS devices need to follow physical laws. For example, sensor input such as vehicle trajectory and

room temperature needs to have space/time continuity; otherwise, the input is physically invalid, which may be a result of sensor spoofing attacks such as the ones studied in §VI. Thus, physical properties can be a unique opportunity for anomaly detection in IoT/CPS systems. One challenge in this direction is how to systematically map cyber events to their corresponding physical properties. To address this challenge, two solution directions may be worth exploring: (1) using well-established behavior models in specific domains, e.g., car following model and queuing model in the transportation field, and (2) using data-driven modeling approach, e.g., using traces of normal behaviors to infer the mapping between cyber and physical events.

Trusted computing hardware assisted security. The software stacks in modern computer systems are large and complex, making it highly difficult, if not impossible, to fully eliminate software vulnerability. However, in safety-critical systems such as autonomous driving and smart transportation, certain functionality, e.g., safety-related features, needs a much stronger security guarantee. To address the challenge, one potential solution direction is to leverage the recent advances in hardware features for trusted computing, e.g., ARM TrustZone [11], and Intel Software Guard Extensions (SGX) [70]. These features provide processor-level isolation to ensure the confidentiality and integrity of security-critical code and data even when the underlying operating systems or any runtime libraries are compromised. One concrete use of such hardware features directly related to this dissertation is to design a CV data integrity protection system, which may fundamentally prevent the data spoofing attacks we identified in smart transportation systems in §VI.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] A DNS RR for specifying the location of services (DNS SRV). https://tools.ietf.org/html/rfc2782.

[2] Adding DNS-SD Service Discovery Records. http://www.dns-sd.org/serverstaticsetup.html.

[3] AFP File Server Security. https://developer.apple.com/library/content/documentation/Networking/Conceptual/AFP/AFPSecurity/AFPSecurity.html.

[4] Amazon Echo. www.amazon.com/echo.

[5] American Fuzzy Lop. http://lcamtuf.coredump.cx/afl.

[6] An Overview of XMPP. https://xmpp.org/about/technology-overview.html.

[7] Analysis result website. http://tinyurl.com/PacketInjectionVulnerability.

[8] Annual Day In The Life of the Internet (DITL) collection. https://www.dns-oarc.net/oarc/data/ditl.

[9] Apple Filing Protocol Concepts. https://developer.apple.com/library/content/documentation/Networking/Conceptual/AFP/Concepts/Concepts.html.

[10] Apple HomeKit. https://www.apple.com/shop/accessories/all-accessories/homekit.

[11] ARM TrustZone: SoC and CPU System-Wide Approach to Security. https://www.arm.com/products/security-on-arm/trustzone.

[12] ASCII Table and Description. http://www.asciitable.com/.

[13] Asterisk custom communications for VoIP. http://www.asterisk.org/.

[14] August Smart Lock. https://august.com/.

[15] Autodiscover for Exchange. https://msdn.microsoft.com/en-us/library/office/jj900169(v=exchg.150).aspx.

[16] Automount NFS in OS X. https://yourmacguy.wordpress.com/2012/06/29/osx-automount/.

[17] Autoware: Open-source software for urban autonomous driving. https://github.com/CPFL/Autoware.

[18] Baidu Apollo: An open autonomous driving platform. http://apollo.auto/.

[19] Baïkal: Cal and CardDAV server based on sabre/dav. http://sabre.io/baikal/.

[20] BestWhois service. https://www.whoisxmlapi.com/terms-of-service.php.

[21] Bonjour API Architecture. https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/NetServices/Articles/programming.html.

[22] Bonjour: Apple's implementation of zero-configuration networking protocols. https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/NetServices/Introduction.html.

[23] Bonjour service types used in Mac OS X. https://developer.apple.com/library/content/qa/qa1312/_index.html.

[24] Building an Enterprise Root Certification Authority in Small and Medium Businesses. https://msdn.microsoft.com/en-us/library/cc875810.aspx.

[25] Calendaring Extensions to WebDAV (CalDAV). https://tools.ietf.org/html/rfc4791.

[26] CardDAV: vCard Extensions to Web Distributed Authoring and Versioning (WebDAV). https://tools.ietf.org/html/rfc6352.

[27] Chromes startup random DNS queries tracked in, and polluting users Google Web History. https://bugs.chromium.org/p/chromium/issues/detail?id=47262.

[28] Cohda Wireless OBU and RSU. http://cohdawireless.com/Products/Hardware.aspx.

[29] Comcast's IPv6 Information Center. http://www.comcast6.net/.

[30] Comparison between NTLM and Kerberos. https://highfromtea.wordpress.com/tag/ntlmssp/.

[31] Configure Email Accounts with Outlook. https://support.marcaria.com/hc/en-us/articles/215526083-Configure-Email-Accounts-with-Outlook.

[32] Configure web-site for access with and without the 'www' domain name prefix. http://support.simpledns.com/kb/a87/configure-web-site-for-access-with-and-without-the-www-domain-name-prefix.aspx.

[33] Configuring Pivotal Cloud Foundry SSL Termination for vSphere Deployments. https://docs.pivotal.io/pivotalcf/1-7/opsguide/ssl-term.html.

[34] Configuring the Commerce Server Network. https://msdn.microsoft.com/en-us/library/aa545742(v=cs.70).aspx.

[35] Connected Vehicle Applications. https://www.its.dot.gov/pilots/cv_pilot_apps.htm.

[36] Connected Vehicle Pilot Deployment Program Phase 1, Concept of Operations (ConOps) Tampa (THEA). https://ntl.bts.gov/lib/57000/57000/57032/FHWA-JPO-16-311.pdf.

[37] CV application: MMITSS-AZ 1.0. https://www.itsforge.net/index.php/community/explore-applications/for-search-results#/30/63.

[38] Demystifying Driving's Dilemma Zone. https://www.insidescience.org/news/demystifying-drivings-dilemma-zone.

[39] DNS-Based Authentication of Named Entities (DANE). https://tools.ietf.org/html/rfc6698.

[40] DNS-Based Service Discovery. https://tools.ietf.org/html/rfc6763.

[41] DNS Long-Lived Queries. https://tools.ietf.org/html/draft-sekar-dns-llq-01.

[42] Download RubyGems. https://rubygems.org/pages/download.

[43] DSRC: The Future of Safer Driving. https://www.its.dot.gov/factsheets/dsrc_factsheet.htm.

[44] DuerOS for Apollo. http://apollo.auto/platform/dueros.html.

[45] Dynamic Updates in the Domain Name System (DNS UPDATE). https://tools.ietf.org/html/rfc2136.

[46] Econolite NEMA Traffic Control. https://www.econolite.com/products/traffic-cabinets/nema.

[47] Edge Server environmental requirements in Skype for Business Server 2015. https://technet.microsoft.com/en-us/library/mt346415.aspx.

[48] ejabberd: robust, massively scalable and extensible XMPP server. https://www.ejabberd.im/.

[49] End-to-End Deep Learning for Self-Driving Cars. https://devblogs.nvidia.com/deep-learning-self-driving-cars.

[50] File Transfer Protocol (FTP). https://tools.ietf.org/html/rfc959.

[51] FTP Security Extensions. https://tools.ietf.org/html/rfc2228.

[52] General Motors: OnStar In-Vehicle Safety and Security. https://www.onstar.com/us/en/home/.

[53] Google Assistant. https://assistant.google.com.

[54] Google open resolver IP addresses. https://developers.google.com/speed/public-dns/docs/using.

[55] Growing Number of ECUs Forces New Approach to Cars Electrical Architecture. http://www.newelectronics.co.uk/electronics-technology/growing-number-of-ecus-forces-new-approach-to-car-electrical-architecture/45039.

[56] Hackers Create 'Ghost' Traffic Jam to Confound Smart Traffic Systems. https://www.theregister.co.uk/2018/03/07/hackers_create_ghost_traffic_jam_to_confound_smart_traffic_systems/.

[57] Hacking Time Machine. https://dreness.com/blog/archives/48.

[58] Half the Web Is Now Encrypted. That Makes Everyone Safer. https://www.wired.com/2017/01/half-web-now-encrypted-makes-everyone-safer/.

[59] Home routers come under attack from new DNS redirection tool. http://www.enyo.de/fw/notes/the-great-corp-renaming.html.

[60] How Drive.ai Is Mastering Autonomous Driving With Deep Learning. https://spectrum.ieee.org/cars-that-think/transportation/self-driving/how-driveai-is-mastering-autonomous-driving-with-deep-learning.

[61] HTTP Authentication: Basic and Digest Access Authentication. https://tools.ietf.org/html/rfc2617.

[62] HTTP Extensions for Distributed Authoring – WEBDAV. https://tools.ietf.org/html/rfc2518.

[63] HTTP Over TLS. https://tools.ietf.org/html/rfc2818.

[64] Hypertext Transfer Protocol – HTTP/1.1. https://tools.ietf.org/html/rfc2616.

[65] IANA Service Name and Transport Protocol Port Number Registry. http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml.

[66] IBM Knowledge Center: LDAP and SSL configuration example. http://www.ibm.com/support/knowledgecenter/SSPFMY_1.3.3/com.ibm.scala.doc/config/iwa_config_ldap_exmpl_c.html.

[67] ICANN: Mitigating the Risk of DNS Namespace Collisions Phase One. https://www.icann.org/news/announcement-2-2014-06-10-en.

[68] ICANN: Proposal to Mitigate Name Collision Risks. https://www.icann.org/public-comments/name-collision-2013-08-05-en.

[69] ICANN Study: Name Collision in the DNS. https://www.icann.org/en/system/files/files/name-collision-02aug13-en.pdf.

[70] Intel Software Guard Extensions (Intel SGX). https://software.intel.com/en-us/sgx.

[71] Internet Printing Protocol/1.1: Encoding and Transport. https://tools.ietf.org/html/rfc2910.

[72] Intra-Site Automatic Tunnel Addressing Protocol (ISATAP). https://tools.ietf.org/html/rfc5214.

[73] Kerberos: The Network Authentication Protocol. http://web.mit.edu/kerberos/.

[74] kpasswd MIT Kerberos Documentation. https://web.mit.edu/kerberos/krb5-1.13/doc/user/user_commands/kpasswd.html.

[75] Let's Encrypt Certificate Authority. https://letsencrypt.org/.

[76] Lightweight Directory Access Protocol (LDAP): The Protocol. https://tools.ietf.org/html/rfc4511.

[77] Linux ISATAP Setup. http://www.litech.org/isatap/.

[78] Max OS X Xgrid. http://www.apple.com/server/macosx/technology/xgrid.html.

[79] Microsoft Key Management Services (KMS). http://help.unc.edu/help/microsoft-key-management-services-kms/.

[80] Microsoft TechNet: SRV Resource Records. https://technet.microsoft.com/en-us/library/cc961719.aspx.

[81] Mirai: What you need to know about the botnet behind recent major DDoS attacks. https://www.symantec.com/connect/blogs/mirai-what-you-need-know-about-botnet-behind-recent-major-ddos-attacks.

[82] MMITSS Final ConOps: Concept of Operations. http://www.cts.virginia.edu/wp-content/uploads/2014/05/Task2.3._CONOPS_6_Final_Revised.pdf.

[83] Name Server API? https://developer.dnsimple.com/v1/nameservers/.

[84] Naming an internal (private) Active Directory LAN. http://arstechnica.com/civis/viewtopic.php?f=17&t=394734.

[85] Network Time Protocol Version 4: Protocol and Algorithms Specification. https://tools.ietf.org/html/rfc5905.

[86] Number of Smartphone Users Worldwide from 2014 to 2020 (in Billions). https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/.

[87] Off-path Packet Injection Vulnerability Detection Results. http://tinyurl.com/PacketInjectionVulnerability.

[88] One Single Malicious Vehicle Can Block "Smart" Street Intersections in the US. https://www.bleepingcomputer.com/news/technology/one-single-malicious-vehicle-can-block-smart-street-intersections-in-the-us/.

[89] Open Source Application Development Portal (OSADP). https://itsforge.net/.

[90] OpenAFS. http://www.openafs.org/.

[91] Openssl: How to generate a CSR with interactively requested alternative theme names? https://www.enmimaquinafunciona.com/pregunta/13352/openssl-como-generar-un-csr-con-nombres-de-alternativa-tema-solicitados-interactivamente-sans.

[92] Page Description Language. http://printwiki.org/Page_Description_Language.

[93] Picture Transfer Protocol (PTP). http://www.imaging.org/ist/resources/standards/ptp-standards.cfm.

[94] Post Office Protocol - Version 3. https://tools.ietf.org/html/rfc1939.

[95] PTV Vissim. http://vision-traffic.ptvgroup.com/en-us/products/ptv-vissim.

[96] Required DNS Records for Automatic Client Sign-In. https://technet.microsoft.com/en-us/library/bb663700(v=office.12).aspx.

[97] REST Resource Naming Guide. http://restfulapi.net/resource-naming/.

[98] RFC 5214. https://tools.ietf.org/html/rfc5214.

[99] RFC 5424. https://tools.ietf.org/html/rfc5424.

[100] RIPE 72 discussion, 05/23/2016: Alert (TA16144A) WPAD Name Collision Vulnerability. https://ripe72.ripe.net/presentations/49-wpad-vulnerability-RIPE-CPH.pdf.

[101] Root server distribution. http://root-servers.org.

[102] Samsung SmartThings. https://www.smartthings.com.

[103] Savari StreetWAVE RSU. http://savari.net/technology/road-side-unit.

[104] Second Level Domain (SLD). http://icannwiki.com/SLD.

[105] Security Mechanism Agreement for the Session Initiation Protocol (SIP). https://tools.ietf.org/html/rfc3329.

[106] Server Message Block Overview. https://technet.microsoft.com/en-us/library/hh831795(v=ws.11).aspx.

[107] Session Traversal Utilities for NAT (STUN). http://www.voip-info.org/wiki/view/STUN.

[108] SFTP - The Modern FTP. https://www.ssh.com/ssh/sftp/.

[109] Simple Mail Transfer Protocol. https://tools.ietf.org/html/rfc2821.

[110] SIP: Session Initiation Protocol. https://tools.ietf.org/html/rfc3261.

[111] Smart Traffic Lights Cause Jams When Fed Spoofed Data. https://nakedsecurity.sophos.com/2018/03/08/smart-traffic-lights-cause-jams-when-fed-spoofed-data/.

[112] Special Interest Group of BA – BAAAAA! (Foosball) and Bug. https://sites.google.com/site/sigbaaaaa.

[113] SSLsplit: Transparent SSL/TLS Interception. https://www.roe.ch/SSLsplit.

[114] STAC: Static Taint Analysis for C. http://code.google.com/p/tanalysis.

[115] Static Analysis vs Dynamic Analysis in Software Testing. http://www.testingexcellence.com/static-analysis-vs-dynamic-analysis-software-testing.

[116] Static Content Subdomain. https://halfelf.org/2015/static-content-subdomain.

[117] Study: Single Connected Car Can Trick Smart Traffic Lights Into Causing Intersection Clogging. https://www.trendmicro.com/vinfo/ph/security/news/internet-of-things/connected-car-can-trick-smart-traffic-lights-causing-intersection-clogging.

[118] The BitTorrent Protocol. http://www.morehawes.co.uk/the-bittorrent-protocol.

[119] The Case Against DNSSEC. http://www.circleid.com/posts/070814_case_against_dnssec/.

[120] The DNS Operations, Analysis, and Research Center (DNS-OARC). https://www.dns-oarc.net/.

[121] The Heartbleed Bug. http://heartbleed.com.

[122] The Remote Framebuffer Protocol. https://tools.ietf.org/html/rfc6143.

[123] Top Level Domain (TLD). http://icannwiki.com/TLD.

[124] Tutorial: Run Your Own Gem Server. http://guides.rubygems.org/run-your-own-gem-server/.

[125] US-CERT: Domain Name Collision Bug Could Result in MitM Attacks. https://www.scmagazine.com/us-cert-domain-name-collision-bug-could-result-in-mitm-attacks/article/528184/.

[126] US-CERT: Leaked WPAD Queries Could Expose Corporate to MitM Attacks. https://securityaffairs.co/wordpress/47716/hacking/wpad-queries-mitm.html.

[127] US-CERT Technical Alert (TA16-144A): WPAD Name Collision Vulnerability. https://www.us-cert.gov/ncas/alerts/TA16-144A.

[128] US Department of Transportation hopes to mandate V2V communications. https://www.cnet.com/roadshow/news/us-department-of-transportation-hopes-to-mandate-v2v-communications.

[129] U.S. DoT Connected Vehicle Pilot Deployment Program. https://www.its.dot.gov/pilots/.

[130] USDOT: 20 Questions About Connected Vehicles. https://www.its.dot.gov/cv_basics/cv_basics_20qs.htm.

[131] USDOT: Connected Vehicles. https://www.its.dot.gov/cv_basics/index.htm.

[132] USDOT: Multi-Modal Intelligent Traffic Safety System (MMITSS). https://www.its.dot.gov/research_archives/dma/bundle/mmitss_plan.htm.

[133] USDOT: Security Credential Management System (SCMS). https://www.its.dot.gov/factsheets/pdf/CV_SCMS.pdf.

[134] Using Digest Authentication as a SASL Mechanism. https://tools.ietf.org/html/rfc2831.

[135] Vehicle-Infrastructure Integration (VII) Initiative: Benefit-Cost Analysis. https://www.pcb.its.dot.gov/connected_vehicle/508/Library/Library-RRs-Institutional/VII%20BCA%20Report%20Ver2-3.htm.

[136] Verisign White Paper: Enterprise Remediation for WPAD Name Collision Vulnerability. https://www.verisign.com/assets/Enterprise_Remediation_for_WPAD_Name_Collision_Vulnerability.pdf.

[137] WannaCry: What you need to know about the WannaCry Ransomware. https://www.symantec.com/blogs/threat-intelligence/wannacry-ransomware-attack.

[138] Waymo Uses Machine Learning to Help Self-Driving Cars See Through Snow. https://www.cnet.com/roadshow/news/waymo-machine-learning-self-driving-cars-snow/.

[139] Web Authentication Proxy Configuration Example. http://www.cisco.com/c/en/us/support/docs/wireless-mobility/wlan-security/116052-config-webauth-proxy-00.html.

[140] When Domain Names Attack: the WPAD Name Collision Vulnerability. http://nakedsecurity.sophos.com/2016/05/25/when-domain-names-attack-the-wpad-name-collision-vulnerability.

[141] WHOIS database. http://whois.icann.org/en.

[142] WPAD Name Collision Bug Opens Door for MitM Attackers. https://www.helpnetsecurity.com/2016/05/24/wpad-name-collision-bug.

[143] WPAD Name Collision Flaw Allows MitM Attacks. https://www.securityweek.com/wpad-name-collision-flaw-allows-mitm-attacks.

[144] Zero Configuration Networking (Zeroconf). http://www.zeroconf.org.

[145] Setting up Web Proxy Autodiscovery Protocol (WPAD) using DNS. http://tektab.com/2012/09/26/setting-up-web-proxy-autodiscovery-protocol-wpad-using-dns, 2012.

[146] The New gTLD Program. https://newgtlds.icann.org/en/about/program, 2013.

[147] Use a Custom TLD for Local Development. http://blog.bfontaine.net/2013/08/26/use-a-custom-tld-for-local-development, 2013.

[148] What is a Windows Domain and How Does It Affect My PC? http://www.howtogeek.com/194069/what-is-a-windows-domain-and-how-does-it-affect-my-pc, 2014.

[149] Centralized Zone Data Service. https://czds.icann.org/en, 2015.

[150] Root Zone Database. http://www.iana.org/domains/root/db, 2015.

[151] New delegated TLD strings. http://newgtlds.icann.org/en/program-status/delegated-strings, 2017.

[152] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, et al. Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice. In *ACM CCS*, 2015.

[153] M. Amoozadeh, A. Raghuramu, C.-N. Chuah, D. Ghosal, H. M. Zhang, J. Rowe, and K. Levitt. Security Vulnerabilities of Connected Vehicle Streams and Their Impact on Cooperative Driving. In *IEEE Communications Magazine*, 2015.

[154] R. Anderson and S. Fuloria. Who Controls the Off Switch? In *IEEE Smart Grid Communications (SmartGridComm)*, 2010.

[155] M. Andrysco, D. Kohlbrenner, K. Mowery, R. Jhala, S. Lerner, and H. Shacham. On Subnormal Floating Point and Abnormal Timing. In *IEEE Symposium on Security and Privacy*, 2015.

[156] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Protocol Modifications for the DNS Security Extensions. RFC4035, 2005.

[157] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Resource Records for the DNS Security Extensions. RFC4034, 2005.

[158] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel. Flowdroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps. In *PLDI*, 2014.

[159] X. Bai, L. Xing, N. Zhang, X. Wang, X. Liao, T. Li, and S.-M. Hu. Staying Secure and Unprepared: Understanding and Mitigating the Security Risks of Apple ZeroConf. In *IEEE Symposium on Security and Privacy*, 2016.

[160] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications. In *IEEE Symposium on Security and Privacy*, 2008.

[161] T. Bao, Y. Zheng, Z. Lin, X. Zhang, and D. Xu. Strict Control Dependence and its Effect on Dynamic Information Flow Analyses. In *ACM ISSTA*, 2010.

[162] K. Bhargavan, D. Obradovic, and C. A. Gunter. Formal Verification of Standards for Distance Vector Routing Protocols. *Journal of the ACM*, 2002.

[163] S. Bishop, M. Fairbairn, M. Norrish, P. Sewell, M. Smith, and K. Wansbrough. Rigorous Specification and Conformance Testing Techniques for Network Protocols, as Applied to TCP, UDP, and Sockets. *SIGCOMM*, 2005.

[164] W. Burghout and J. Wahlstedt. Hybrid Traffic Simulation With Adaptive Signal Control. In *Transportation Research Record: Journal of the Transportation Research Board*, 2007.

[165] N. Carlini, P. Mishra, T. Vaidya, Y. Zhang, M. Sherr, C. Shields, D. Wagner, and W. Zhou. Hidden Voice Commands. In *USENIX Security Symposium*, 2016.

[166] N. Carlini and D. Wagner. Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. In *ACM Workshop on Artificial Intelligence and Security*, 2017.

[167] N. Carlini and D. Wagner. Towards Evaluating the Robustness of Neural Networks. In *IEEE Symposium on Security and Privacy*, 2017.

[168] S. Castro, D. Wessels, M. Fomenkov, and K. Claffy. A Day at the Root of the Internet. In *ACM SIGCOMM Computer Communication Review*, 2008.

[169] R. Chang, G. Jiang, F. Ivancic, S. Sankaranarayanan, and V. Shmatikov. Inputs of Coma: Static Detection of Denial-of-Service Vulnerabilities. In *CSF*, 2009.

[170] P. Chapman and D. Evans. Automated Black-box Detection of Side-channel Vulnerabilities in Web Applications. In *ACM CCS*, 2011.

[171] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, et al. Comprehensive Experimental Analyses of Automotive Attack Surfaces. In *USENIX Security*, 2011.

[172] Q. A. Chen, E. Osterweil, M. Thomas, and Z. M. Mao. MitM Attack by Name Collision: Cause Analysis and Vulnerability Assessment in the New gTLD Era. In *IEEE Symposium on Security and Privacy*, 2016.

[173] Q. A. Chen, Z. Qian, Y. Jia, Y. Shao, and Z. M. Mao. Static Detection of Packet Injection Vulnerabilities – A Case for Identifying Attacker-controlled Implicit Information Leaks. In *ACM CCS*, 2015.

[174] Q. A. Chen, Z. Qian, and Z. M. Mao. Peeking into Your App without Actually Seeing It: UI State Inference and Novel Android Attacks. In *USENIX Security*, 2014.

[175] Q. A. Chen, M. Thomas, E. Osterweil, Y. Cao, J. You, and Z. M. Mao. Client-side Name Collision Vulnerability in the New gTLD Era: A Systematic Study. In *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS)*, 2017.

[176] Q. A. Chen, Y. Yin, Y. Feng, Z. M. Mao, and H. X. Liu. Exposing Congestion Attack on Emerging Connected Vehicle based Traffic Signal Control. In *Proceedings of the 25th Network and Distributed System Security Symposium (NDSS)*, 2018.

[177] S. Chen, R. Wang, X. Wang, and K. Zhang. Side-channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow. In *IEEE Symposium on Security and Privacy*, 2010.

[178] S. Chen, R. Wang, X. Wang, and K. Zhang. Side-channel Leaks in Web Applications: A Reality Today, a Challenge Tomorrow. In *IEEE Symposium on Security and Privacy*, 2010.

[179] S. Cheshire and M. Krochmal. Multicast DNS. RFC6762, 2013.

[180] K.-T. Cho and K. G. Shin. Fingerprinting Electronic Control Units for Vehicle Intrusion Detection. In *USENIX Security Symposium*, 2016.

[181] D. Clark, S. Hunt, and P. Malacaria. A Static Analysis for Quantifying Information Flow in a Simple Imperative Language. In *Journal of Computer Security*, 2007.

[182] C. Cowan, C. Pu, D. Maier, J. Walpole, and P. Bakke. StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks. In *USENIX Security*, 1998.

[183] A. Cui and S. J. Stolfo. A Quantitative Analysis of the Insecurity of Embedded Network Devices: Results of a Wide-area Scan. In *ACM ACSAC*, 2010.

[184] X. d. C. de Carnavalet and M. Mannan. Killed by Proxy: Analyzing Client-end TLS Interception Software. In *ISOC NDSS*, 2016.

[185] L. De Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS*, 2008.

[186] C. Deccio. Whats in a Name (Collision)? Modeling and Quantifying Collision Potential. In *Workshop and Prize on Root Causes and Mitigation of Name Collisions (WPNC)*, 2014.

[187] D. Dominic, S. Chhawri, R. M. Eustice, D. Ma, and A. Weimerskirch. Risk Assessment for Cooperative Automated Driving. In *ACM CPS-SP Workshop*, 2016.

[188] G. Doychev, D. Feld, B. Köpf, L. Mauborgne, and J. Reineke. CacheAudit: A Tool for the Static Analysis of Cache Side Channels. In *Usenix Security*, 2013.

[189] G. Doychev, B. Köpf, L. Mauborgne, and J. Reineke. Cacheaudit: A Tool for the Static Analysis of Cache Side Channels. In *USENIX Security*, 2013.

[190] H. Duan, N. Weaver, Z. Zhao, M. Hu, J. Liang, J. Jiang, K. Li, and V. Paxson. Hold-on: Protecting Against On-path DNS Poisoning. In *Workshop on Securing and Trusting Internet Names*, 2012.

[191] E. Dumazet. Kernel discussion on ACK flag set. `http://comments.gmane.org/gmane.linux.network/253369`, 2012.

[192] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer, et al. The Matter of Heartbleed. In *ACM Internet measurement conference*, 2014.

[193] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. Analysis of the HTTPS Certificate Ecosystem. In *ACM Internet measurement conference*, 2013.

[194] D. Eastlake 3rd and A. Panitz. Reserved Top Level DNS Names. RFC2606, 1999.

[195] M. Emmelmann, B. Bochow, and C. Kellum. Vehicular Networking: Automotive Applications and Beyond. John Wiley & Son, 2010.

[196] S. Fahl, M. Harbach, T. Muders, and M. Smith. Why Eve and Mallory love Android: An analysis of SSl (in) security on Android. In *ACM CCS*, 2012.

[197] Y. Feng, K. L. Head, S. Khoshmagham, and M. Zamanipour. A Real-time Adaptive Signal Control In A Connected Vehicle Environment. In *Elsevier Transportation Research Part C: Emerging Technologies*, 2015.

[198] P. Gauthier, J. Cohen, M. Dunsmuir, and C. Perkins. The Web Proxy Auto-Discovery Protocol. *Internet draft, IETF*, 1999.

[199] Gavron, Ehud. A Security Problem and Proposed Correction With Widely Deployed DNS Software. RFC1535, 1993.

[200] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. The Most Dangerous Code in the World: Validating SSL Certificates in Non-browser Software. In *ACM CCS*, 2012.

[201] A. Ghafouri, W. Abbas, Y. Vorobeychik, and X. Koutsoukos. Vulnerability of Fixed-time Control of Signalized Intersections to Cyber-tampering. In *IEEE Resilience Week (RWS)*, 2016.

[202] B. Ghena, W. Beyer, A. Hillaker, J. Pevarnek, and J. A. Halderman. Green Lights Forever: Analyzing the Security of Traffic Infrastructure. In *Usenix WOOT*, 2014.

[203] C. Gibler, J. Crussell, J. Erickson, and H. Chen. AndroidLeaks: Automatically Detecting Potential Privacy Leaks in Android Applications on a Large Scale. In *TRUST*, 2012.

[204] Y. Gilad and A. Herzberg. Off-Path Attacking the Web. In *USENIX WOOT*, 2012.

[205] Y. Gilad and A. Herzberg. When Tolerance Causes Weakness: The Case of Injection-Friendly Browsers. In *WWW*, 2013.

[206] J. A. Goguen and J. Meseguer. Security Policies and Security Models. In *IEEE Symposium on Security and Privacy*, 1982.

[207] S. Goldberg, M. Naor, D. Papadopoulos, L. Reyzin, S. Vasant, and A. Ziv. NSEC5: Provably Preventing DNSSEC Zone Enumeration. In *ISOC NDSS*, 2015.

[208] M. I. Gordon, D. Kim, J. Perkins, L. Gilham, N. Nguyen, and M. Rinard. Information-flow Analysis of Android Applications in DroidSafe. In *NDSS*, 2015.

[209] D. Gullasch, E. Bangerter, and S. Krenn. Cache Games–Bringing Access-based Cache Attacks on AES to Practice. In *IEEE Symposium on Security and Privacy*, 2011.

[210] Y. Guo. *Improving Application QoE with Flow-Level, Interface-Level, and Device-Level Parallelism*. PhD thesis, University of Michigan, 2017.

[211] T. Halvorson, M. F. Der, I. Foster, S. Savage, L. K. Saul, and G. M. Voelker. From .academy to .zone: An Analysis of the New TLD Land Rush. In *ACM IMC*, 2015.

[212] T. Halvorson, K. Levchenko, S. Savage, and G. M. Voelker. XXXtortion? Inferring Registration Intent in the .XXX TLD. In *ACM WWW*, 2014.

[213] T. Halvorson, J. Szurdi, G. Maier, M. Felegyhazi, C. Kreibich, N. Weaver, K. Levchenko, and V. Paxson. The BIZ Top-Level Domain: Ten Years Later. In *Passive and Active Measurement*, 2012.

[214] M. Hind, M. Burke, P. Carini, and J.-D. Choi. Interprocedural Pointer Alias Analysis. *TOPLAS*, 21(4):848–894, 1999.

[215] S. Jana and V. Shmatikov. Memento: Learning Secrets from Process Footprints. In *IEEE Symposium on Security and Privacy*, 2012.

[216] N. Jovanovic, C. Kruegel, and E. Kirda. Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities. In *IEEE Symposium on Security and Privacy*, 2006.

[217] B. S. Kaliski Jr. and A. Mankin. US Patent Application 20150256424: Name Collision Risk Manager. `http://www.freepatentsonline.com/y2015/0256424.html`.

[218] M. G. Kang, S. McCamant, P. Poosankam, and D. Song. DTA++: Dynamic Taint Analysis with Targeted Control-Flow Propagation. In *NDSS*, 2011.

[219] J. B. Kenney. Dedicated short-range communications (DSRC) standards in the United States. In *Proceedings of the IEEE*, 2011.

[220] A. Kieyzun, P. J. Guo, K. Jayaraman, and M. D. Ernst. Automatic creation of SQL injection and cross-site scripting attacks. In *ICSE*, 2009.

[221] D. King, B. Hicks, M. Hicks, and T. Jaeger. Implicit flows: Can't Live with 'em, Can't Live Without 'em. *Information Systems Security, Lecture Notes in Computer Science*, 5352:56–70, 2008.

[222] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO*, 1996.

[223] B. Könings, C. Bachmaier, F. Schaub, and M. Weber. Device Names in the Wild: Investigating Privacy Risks of Zero Configuration Networking. In *IEEE International Conference on Mobile Data Management*, 2013.

[224] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, et al. Experimental Security Analysis of a Modern Automobile. In *IEEE Symposium on Security and Privacy*, 2010.

[225] N. Kothari, R. Mahajan, T. Millstein, R. Govindan, and M. Musuvathi. Finding Protocol Manipulation Attacks. In *SIGCOMM*, 2011.

[226] A. Laszka, B. Potteiger, Y. Vorobeychik, S. Amin, and X. Koutsoukos. Vulnerability of Transportation Networks to Traffic-signal Tampering. In *ACM ICCPS*, 2016.

[227] C. Lever, R. Walls, Y. Nadji, D. Dagon, P. McDaniel, and M. Antonakakis. Domain-Z: 28 Registrations Later. In *IEEE Symposium on Security and Privacy*, 2016.

[228] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. Last-Level Cache Side-Channel Attacks are Practical. In *IEEE Symposium on Security and Privacy*, 2015.

[229] J. Liu, Y. Xiao, S. Li, W. Liang, and C. P. Chen. Cyber Security and Privacy Issues in Smart Grids. In *IEEE Communications Surveys & Tutorials*, 2012.

[230] G. Lowe. Quantifying Information Flow. In *IEEE Workshop on Computer Security Foundations*, 2002.

[231] C. Lozoya, P. Martí, M. Velasco, J. M. Fuertes, and E. X. Martin. Resource and Performance Trade-offs in Real-time Embedded Control Systems. In *Real-Time Systems*, 2013.

[232] X. Luo, P. Zhou, E. W. Chan, W. Lee, R. K. Chang, and R. Perdisci. HTTPOS: Sealing Information Leaks with Browser-side Obfuscation of Encrypted Flows. In *NDSS*, 2011.

[233] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. In *arXiv preprint arXiv:1706.06083*, 2017.

[234] A. Malhotra, I. E. Cohen, E. Brakke, and S. Goldberg. Attacking the Network Time Protocol. In *NDSS*, 2016.

[235] H. C. Manual. Highway Capacity Manual. In *Transportation Research Board*, 2000.

[236] S. Mazloom, M. Rezaeirad, A. Hunter, and D. McCoy. A Security Analysis of an In-Vehicle Infotainment and App Platform. In *Usenix WOOT*, 2016.

[237] C. Meyer, J. Somorovsky, E. Weiss, J. Schwenk, S. Schinzel, and E. Tews. Revisiting SSL/TLS Implementations: New Bleichenbacher Side Channels and Attacks. In *USENIX Security*, 2014.

[238] P. Mockapetris and K. J. Dunlap. Development of the Domain Name System. In *ACM SIGCOMM*, 1988.

[239] Mockapetris, Paul. Domain Names - Implementation and Specification. rfc1035, 2004.

[240] B. Muller. Whitepaper: Improved DNS Spoofing Using Node Re-delegation. `https://www.sec-consult.com/fxdata/seccons/prod/downloads/whitepaper-dns-node-redelegation.pdf`.

[241] V. Narayanan and Y. Xie. Reliability Concerns in Embedded System Designs. In *IEEE Computer*, 2006.

[242] G. C. Necula, S. McPeak, S. P. Rahul, and W. Weimer. CIL: Intermediate Language and Tools for Analysis and Transformation of C prhttps://www.readcube.com/homeograms. In *CC*, 2002.

[243] E. Osterweil and D. McPherson. New gTLD Security and Stability Considerations. Technical Report 1130007 version 1, 2013. `http://techreports.verisignlabs.com/docs/tr-1160018-1.pdf`.

[244] E. Osterweil, D. McPherson, and L. Zhang. The Shape and Size of Threats: Defining a Networked System's Attack Surface. In *IEEE ICNP*, 2014.

[245] E. Osterweil, M. Thomas, A. Simpson, and D. McPherson. New gTLD Security, Stability, Resiliency Update: Exploratory Consumer Impact Analysis. Technical report, 2013. `http://techreports.verisignlabs.com/docs/tr-1130008-1.pdf`.

[246] E. Osterweil and L. Zhang. Interadministrative Challenges in Managing DNSKEYs. *IEEE Security and Privacy*, 7(5):44–51, 2009.

[247] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks. In *IEEE Symposium on Security and Privacy*, 2016.

[248] L. Pedrosa, A. Fogel, N. Kothari, R. Govindan, R. Mahajan, and T. Millstein. Analyzing protocol implementations for interoperability. In *NSDI*, 2015.

[249] K. Pei, Y. Cao, J. Yang, and S. Jana. Deepxplore: Automated Whitebox Testing of Deep Learning Systems. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2017.

[250] Z. Qian and Z. M. Mao. Off-Path TCP Sequence Number Inference Attack – How Firewall Middleboxes Reduce Security. In *IEEE Symposium on Security and Privacy*, 2012.

[251] Z. Qian, Z. M. Mao, and Y. Xie. Collaborative TCP Sequence Number Inference Attack: How to Crack Sequence Number Under A Second. In *ACM CCS*, 2012.

[252] A. Quach, Z. Wang, and Z. Qian. Investigation of the 2016 Linux TCP Stack Vulnerability at Scale. In *ACM SIGMETRICS*, 2017.

[253] Ramaiah, Anantha and Stewart, R and Dalal, Mitesh. Improving TCP's Robustness to Blind In-Window Attacks. RFC5961, 2010.

[254] D. A. Ramos and D. R. Engler. Under-Constrained Symbolic Execution: Correctness Checking for Real Code. In *USENIX Security*, 2015.

[255] A. Rane, C. Lin, and M. Tiwari. Raccoon: Closing Digital Side-Channels through Obfuscated Execution. In *USENIX Security*, 2015.

[256] S. Rasthofer, S. Arzt, and E. Bodden. A machine-learning Approach for Classifying and Categorizing Android Sources and Sinks. In *NDSS*, 2014.

[257] T. Reps, S. Horwitz, and M. Sagiv. Precise Interprocedural Dataflow Analysis via Graph Reachability. In *POPL*, 1995.

[258] A. Rountev, M. Sharp, and G. Xu. IDE Dataflow Analysis in the Presence of Large Object-oriented Libraries. In *CC*, 2008.

[259] M. Sagiv, T. Reps, and S. Horwitz. Precise Interprocedural Dataflow Analysis with Applications to Constant Propagation. *Theoretical Computer Science*, 167(1):131–170, 1996.

[260] S. Sen and K. L. Head. Controlled Optimization of Phases at an Intersection. In *Transportation science*, 1997.

[261] U. Shankar, K. Talwar, J. S. Foster, and D. Wagner. Detecting Format String Vulnerabilities with Type Qualifiers. In *USENIX Security*, 2001.

[262] Y. Shao, J. Ott, Q. A. Chen, Z. Qian, and Z. M. Mao. Kratos: Discovering Inconsistent Security Policy Enforcement in the Android Framework. In *Proceedings of the 23rd Network and Distributed System Security Symposium (NDSS)*, 2016.

[263] M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter. Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-art Face Recognition. In *ACM CCS*, 2016.

[264] A. Sharma, D. Bullock, and S. Peeta. Estimating Dilemma Zone Hazard Function at High Speed Isolated Intersection. In *Elsevier Transportation research part C: emerging technologies*, 2011.

[265] A. Simpson. Detecting Search Lists in Authoritative DNS. In *Workshop and Prize on Root Causes and Mitigation of Name Collisions (WPNC)*, 2014.

[266] S. Sivakorn, G. Argyros, K. Pei, A. D. Keromytis, and S. Jana. HVLearn: Automated Black-box Analysis of Hostname Verification in SSL/TLS Implementations. In *IEEE Symposium on Security and Privacy*, 2017.

[267] S. Son and V. Shmatikov. The Hitchhiker's Guide to DNS Cache Poisoning. In *Security and Privacy in Communication Networks*. Springer, 2010.

[268] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena. BitBlaze: A New Approach to Computer Security via Binary Analysis. In *Information systems security*, 2008.

[269] S. Stamm, Z. Ramzan, and M. Jakobsson. Drive-by pharming. In *Information and Communications Security*, pages 495–506, 2007.

[270] M. Thomas, Y. Labrou, and A. Simpson. The Effectiveness of Block Lists to Prevent Collisions. In *Workshop and Prize on Root Causes and Mitigation of Name Collisions (WPNC)*, 2014.

[271] O. Tripp, M. Pistoia, S. J. Fink, M. Sridharan, and O. Weisman. TAJ: Effective Taint Analysis of Web Applications. In *PLDI*, 2009.

[272] T. Urbanik, A. Tanaka, B. Lozner, E. Lindstrom, K. Lee, S. Quayle, S. Beaird, S. Tsoi, P. Ryus, D. Gettman, et al. Signal Timing Manual. In *Transportation Research Board*, 2015.

[273] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, And Tools (2nd Edition)*. Addison Wesley, 2006.

[274] V. Varadarajan, T. Ristenpart, and M. Swift. Scheduler-based Defenses Against Cross-VM Side-channels. In *Usenix Security*, 2014.

[275] G. Wassermann and Z. Su. Static Detection of Cross-site Scripting Vulnerabilities. In *Proceedings of the 30th international conference on Software engineering*, 2008.

[276] W. Whyte, A. Weimerskirch, V. Kumar, and T. Hehn. A Security Credential Management System for V2V Communications. In *IEEE Vehicular Networking Conference (VNC)*, 2013.

[277] R. P. Wilson and M. S. Lam. Efficient Context-sensitive Pointer Analysis for C Programs. In *PLDI*, 1995.

[278] M. Y. Wong and D. Lie. IntelliDroid: A Targeted Input Generator for the Dynamic Analysis of Android Malware. In *NDSS*, 2016.

[279] M. Wu, W. Ma, and L. Li. Characterize Dilemma Zone and Minimize Its Effect at Coordinated Signalized Intersections. In *Elsevier Procedia-Social and Behavioral Sciences*, 2013.

[280] Y. Xie and A. Aiken. Saturn: A Scalable Framework for Error Detection using Boolean Satisfiability. *TOPLAS*, 2007.

[281] F. Yamaguchi, N. Golde, D. Arp, and K. Rieck. Modeling and Discovering Vulnerabilities with Code Property Graphs. In *IEEE Symposium on Security and Privacy*, 2014.

[282] Y. Yarom and K. E. Falkner. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. *USENIX Security*, 2014.

[283] G. Zhang, C. Yan, X. Ji, T. Zhang, T. Zhang, and W. Xu. DolphinAttack: Inaudible Voice Commands. In *ACM CCS*, 2017.

[284] K. Zhang, Z. Li, R. Wang, X. Wang, and S. Chen. Sidebuster: Automated Detection and Quantification of Side-channel Leaks in Web Application Development. In *ACM CCS*, 2010.

[285] K. Zhang and X. Wang. Peeping Tom in the Neighborhood: Keystroke Eavesdropping on Multi-User Systems. In *USENIX Security*, 2009.

[286] Y. Zhang, C. Fu, and L. Hu. Yellow Light Dilemma Zone Researches: A Review. In *Journal of traffic and transportation engineering*, 2014.

[287] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Cross-VM side channels and their use to extract private keys. In *ACM CCS*, 2012.

[288] Y. Zhang and M. K. Reiter. Düppel: Retrofitting commodity operating systems to mitigate cache side channels in the cloud. In *ACM CCS*, 2013.

[289] C. Zheng, S. Zhu, S. Dai, G. Gu, X. Gong, X. Han, and W. Zou. Smartdroid: An Automatic System for Revealing UI-based Trigger Conditions in Android Applications. In *ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, 2012.

[290] Y. Zheng and X. Zhang. Path Sensitive Static Analysis of Web Applications for Remote Code Execution Vulnerability Detection. In *ICSE*, 2013.

[291] X. Zhou, S. Demetriou, D. He, M. Naveed, X. Pan, X. Wang, C. A. Gunter, and K. Nahrstedt. Identity, Location, Disease and More: Inferring Your Secrets from Android Public Resources. In *ACM CCS*, 2013.

[292] Z. Zhou, Z. Qian, M. K. Reiter, and Y. Zhang. Static Evaluation of Noninterference using Approximate Model Counting. In *IEEE Symposium on Security and Privacy*, 2018.