Client-side Name Collision Vulnerability in the New gTLD Era: A Systematic Study

Qi Alfred Chen, Matthew Thomas†, Eric Osterweil†, Yulong Cao, Jie You, Z. Morley Mao
University of Michigan, †Verisign Labs
alfchen@umich.edu,{mthomas,eosterweil}@verisign.com,{yulongc,jieyou,zmao}@umich.edu

ABSTRACT

The recent unprecedented delegation of new generic top-level domains (gTLDs) has exacerbated an existing, but fallow, problem called name collisions. One concrete exploit of such problem was discovered recently, which targets internal namespaces and enables Man in the Middle (MitM) attacks against end-user devices from anywhere on the Internet. Analysis of the underlying problem shows that it is not specific to any single service protocol, but little attention has been paid to understand the vulnerability status and the defense solution space at the service level. In this paper, we perform the first systematic study of the robustness of internal network services under name collision attacks.

We first perform a measure study and uncover a wide spectrum of services affected by the name collision problem. We then collect their client implementations and systematically analyze their vulnerability status under name collision attacks using dynamic analysis. Out of the 48 identified exposed services, we find that nearly all (45) of them expose vulnerabilities in popular clients. To demonstrate the severity, we construct exploits and find a set of new name collision attacks with severe security implications including MitM attacks, internal or personal document leakage, malicious code injection, and credential theft. We analyze the causes, and find that the name collision problem broadly breaks common security assumptions made in today's service client software. Leveraging the insights from our analysis, we propose multiple service software level solutions, which enables the victim services to actively defend against name collision attacks.

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 [102]. 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 [44, 102]. 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 [79, 87]. 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 [41], 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 [44, 87, 95, 102], 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 paper, 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 [87], 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

CCS'17, Oct. 30–Nov. 3, 2017, Dallas, TX, USA.

978-1-4503-4946-8/17/10…$15.00
DOI: 10.1145/3133956.3134084

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) [14], 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 [87, 95] 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.

2 BACKGROUND

2.1 The Name Collision Problem

In DNS, a domain name is a set of dot-separated labels that form a tree structure with the DNS root located at the top. The last two labels, for example `.com` and `example` in `www.example.com`, are called the TLD (top-level domain) and SLD (second-level domain). In the DNS ecosystem, the public DNS namespace is for the resolution of domain names on the public Internet, and the Internet Corporation for Assigned Names and Numbers (ICANN) is the authoritative administrator for its DNS root. ICANN delegates the management of the TLDs to specific TLD registry operators. Outside of the public DNS namespace, a local area network can also setup an internal DNS namespace using private domain names. This is common practice for corporate networks to control internal data access and resolution. In an internal namespace, the network administrators provision internal DNS zones and configure their internal resolvers to query these servers instead of the public namespace. To prevent confusion between internal and public namespaces, the administrators usually use TLD strings that have not been delegated in the public namespace as the internal TLDs (iTLDs).

In late 2013, ICANN launched the New gTLD Program [83], which has delegated more than 1,000 new gTLDs in three years, making it the largest expansion of the public namespace ever. As a side effect, many popular iTLD strings were also delegated. Thus, the internal domain names using previously undelegated iTLD strings can now be registered in the public namespace. Meanwhile, it has long been known that internal namespace queries are leaked to the public namespace every day [87, 100, 102]. Previously, leaked queries would not resolve in the public namespace as the iTLD was not delegated. However, with the expanded public namespace, these leaked queries can now be answered by malicious registrants owning the colliding domains in the public namespace. Such resolution of a domain name in an unintended namespace is known as the name collision problem [44, 87, 102]. By exploiting these query leaks, this problem has become an attack vector.

An exploit instance: the WPAD name collision attack. WPAD is a protocol designed for end devices to automatically configure web proxies. It is primarily used in internal networks where clients are restricted from communicating to the public Internet. The proxy configuration process uses a special DNS query, which prepends the label `wpad` to the internal domain name. If the query resolves, the client device retrieves and applies the proxy configuration provided by the server. Due to the sensitive nature of this protocol, all WPAD queries should never leave the internal network. However, as characterized by previous work [87, 102], millions of WPAD queries are observed in the public namespace every day. Due to the name collision problem, malicious registrants can register vulnerable domains with WPAD queries leaks, and web traffic of Internet users from all over the world can be automatically redirected to the attacker's MitM proxy. Chen et al. recently characterized the severity and vulnerability status of this attack, and demonstrated a real threat to users [87].

2.2 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 [27], 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) [88]. When used with mDNS, DNS-SD can provide Zeroconf [82], which can discover services on nearby devices in local link without setting up unicast DNS servers. A popular Zeroconf implementation is the Apple Bonjour [14], which is built-in with the latest macOS [15].

This paper 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 [41]. In this paper, we refer to the service names in the IANA registry *registered* names and others as *non-registered* names.

2.3 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 pre-installed in popular OSes or browsers. For an internal network, the network administrators typically use self-signed local CAs [16], 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 [47] and DIGEST-MD5 [80]. In this paper, we perform a systematic vulnerability analysis to understand whether the clients with server authentication support are robust enough under the name collision threat model.

3 CLIENT-SIDE NAME COLLISION VULNERABILITY

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

3.1 Threat Model

As covered in §2, internal DNS namespace queries are observed to be leaked into the public namespace. Among them, as we later characterize in §4, 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 paper, 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 [76], (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. 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), 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 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.

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

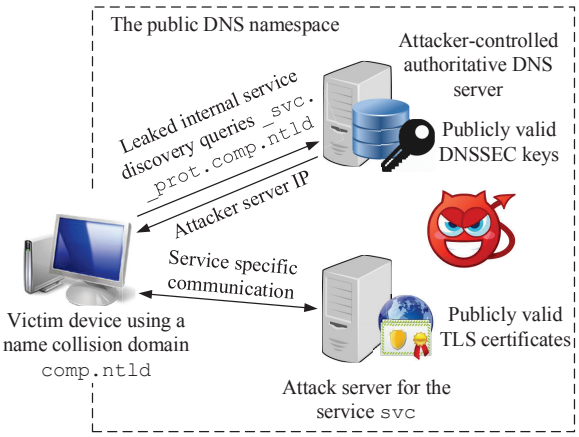

Figure 1: The generalized name collision attack threat model.

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 [87].

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 [49]. 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 [93].

3.2 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 §4, 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 paper, 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 paper, 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, we detail the analysis results and findings.

4 EXPOSED SERVICE CHARACTERIZATION

In this section, we measure the exposed internal network services (defined in §3.2), and characterize their functionality.

4.1 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 [5]. 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 [86]. 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 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.

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 [99]. 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 paper, we consider the delegated new gTLDs as of March 4, 2017, which include 1,216 new gTLDs in total [84].

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). In our study, our main focus is the services officially registered in the IANA registry [41]. These are services that are widely used in industry, e.g., `sip` and `ldap`, and their IANA registration entries have

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 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 6 in Appendix.

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 paper, 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. To overcome this challenge, we use an automated script to conservatively rule out service name candidates without sufficient information for our study. Details are in Appendix §A.

4.2 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. With the help of the automatic script (§4.1), we choose 30 service names with top popularity for subsequent analysis. More details are in Appendix §A.

In total, we form an exposed service dataset of 80 services with highest levels of service query exposure today. Table 1 characterizes 68 of them according to their designed functionality and potential security implications under name collision attacks. Table 6 in Appendix includes 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 4 in Appendix). In Table 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 1, in addition to the previously-studied WPAD service [87], the name collision problem actually affects a wide spectrum of internal network services with 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 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 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.

5.1 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 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

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

Table 2: Vulnerability analysis results for the collected client implementations of the exposed services.

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 2 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 1. For the remaining ones, we were unable to obtain valid software for our study (details in Table 5 in the Appendix).

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. [41]

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 (§3.1), 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 [73].

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 [72].

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 [49]. 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.

5.2 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. 2, these clients use the traditional service discovery methods via SRV or A/AAAA queries as introduced in §2.2. 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

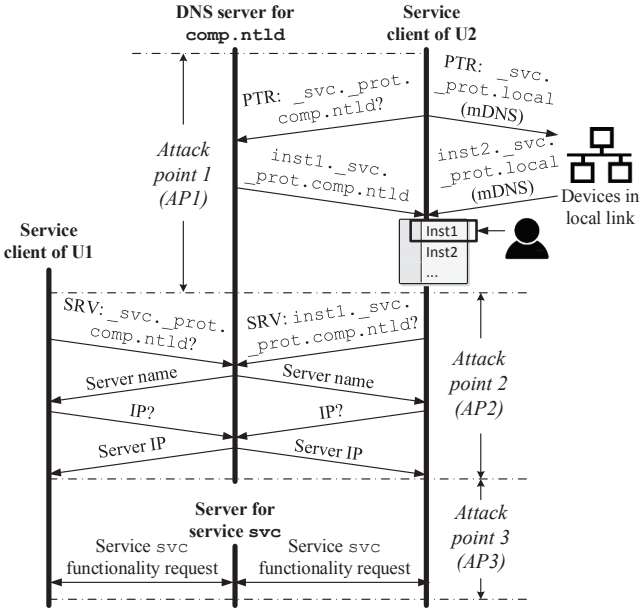

Figure 2: Illustration of usage scenario U1 and U2 of DNS-based service discovery (§5.2) in our service client collection.

queries to retrieve a list of server names for the user to choose as shown on the right of Fig. 2, which conforms to the Zeroconf usage of the DNS-SD standard [82]. Even though Zeroconf is mostly designed for discovering nearby devices without unicast DNS server support [14], 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. 2.

5.3 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 2.

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 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 [46, 68, 91]. Also for services such as `sip` and `ftp`, server authentication is mentioned but the implementation is not enforced [34, 66]. 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 2, 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 §5.4, we have more discussions on this problem cause.

V3. Mix local-link and unicast DNS domain discovery. As discussed in §5.2, clients in U2 are mainly designed to discover servers at nearby devices in the local link using mDNS queries [14] 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 [13], 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. 2, 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 §6, 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 [21]. 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) [6] 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 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 §3.1, 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 paper, 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 §7 we propose a set of defense strategies at the service client software design level.

6 EXPLOITATION CASE STUDY

To demonstrate the severity of the identified vulnerabilities, we construct realistic attack scenarios in our analysis framework. Table 3 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 [87], 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. Due to the space limit, this section only describes the attacks with potentially more interesting attack strategies; the others are in Appendix §B and §C.

6.1 MitM Attack

In the previous work [87], 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. 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) [46]. It is an IPv6 transition mechanism to enable a client to use IPv6 in an internal network that only

Section	Client software	Service name	Vulnerable design/imp. choice	Exploitation
§6.1	Microsoft Windows 10	isatap	V1	MitM attack (web traffic & DNS traffic)
§6.2	RubyGems 2.6.12	rubygems	V1, V2	Malicious code injection
§6.3	macOS 10.12 printer discovery	ipp	V1, V3	Document leakage
§6.4	PSI 0.15, macOS 10.12 and iOS 10.3 Contacts, Calendar	xmpp-client, carddav, carddavs, caldav, caldavs	V1, V2, V4	Credential theft
	macOS 10.12 Terminal	ftp, ssh, sftp-ssh, telnet	V1, V3, V4	
Appendix §B	Linphone 3.10.2, PSI 0.15	sip, xmpp-client	V1, V2, V4	Phishing calls & messages
Appendix §C	macOS 10.12 and iOS 10.3 Contacts, Calendar	carddav, carddavs caldav, caldavs	V1, V2, V4	Phishing name card & calendar event injection

Table 3: Exploitation case studies for the identified client-side name collision vulnerabilities. V1 to V4 are detailed in §5.3.

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 [25]. 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 [51]. 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. [20]. 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 [35], 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 [75] and can thus read and modify arbitrary DNS responses.

6.2 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 [78]. 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 §5.3, 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.

6.3 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. 2). 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 [7]). 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.

6.4 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 in Appendix §B, §C.

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 [7], 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` [7]. 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.

7 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. Leveraging the insights in §5.4, in this section we propose a set of service client software level defense strategies, which complements the previously-proposed DNS ecosystem level solutions [43, 87, 95]. Compared to other levels, service clients are the direct victims of name collision attacks. Considering the amount of time required for policy making and solution deployment to tens to hundreds of sites in the DNS ecosystem [87], it is necessary to provide nearer-term solutions to these service clients so that they are able to proactively protect themselves. More importantly, as we show below, service clients are the actual issuers of the exploited leaked queries and thus can leverage the knowledge of the intended namespace to fundamentally and more effectively prevent name collision attacks. Note that our analysis also helps extend the previous DNS ecosystem level solutions to the general form of name collision attacks, which is in Appendix §D.

Integrate and enforce server authentication. As discussed in §5.4 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 §5.4 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 subject 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) [26]. 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 [101]. 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 (§6.2), 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 (§6.3), the client software can disable the use of invisible characters to prevent the attacker from manipulating printer name display.

8 RELATED WORK

Name collision attacks. After the launch of the New gTLD Program, some studies have been performed to understand its impact on the DNS ecosystem [94, 96], including a few expressing concerns about the name collision problem. A preliminary study by ICANN found that the potential for name collisions with the proposed new gTLDs was substantial [44]. Osterweil et al. used the leaked queries to quantify the name collision risk [102]. Compared to these work on analyzing the name collision problem, our work focuses on studying the vulnerability status and defense solutions at the service level.

Recently, Chen et al. demonstrated the first concrete exploitation of name collisions, the WPAD name collision attack, and performed a systematic study of the problem cause and the vulnerability status [87]. In comparison, our work generalizes the name collision threat model, and performs security analysis on a much larger and more diverse set of exposed internal network services.

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 [89], or off the path but physically inside the targeted network [103]. Compared to these attacks with tight attack placement and timing requirements, the name collision attack studied in this paper 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 [98]. 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 focuses on vulnerability analysis at the service software design level.

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 [97]. Xing et al. studied the major Zeroconf frameworks, and found popular apps such as AirDrop are vulnerable to MitM attacks [85]. 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 §3.1). 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.

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 [90, 92]. In comparison, our paper 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 [21, 37]. This paper characterizes the use of these weak methods in the exposed service clients, which is found to be a common vulnerable design choice under name collision attacks.

9 CONCLUSION

In this paper, 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.

ACKNOWLEDGMENTS

We would like to thank Danny McPherson, Burt Kaliski, Tomofumi Okubo, Jeremy Erickson, Yihua Guo, Yunhan Jack Jia, Yuru Shao, Yikai Lin, David Ke Hong, Shichang Xu, and the anonymous reviewers for providing valuable feedback on our work. The University of Michigan authors were supported in part by the National Science Foundation under grants CNS-1318306 and CNS-1526455 and by ONR grant N00014-14-1-0440.

REFERENCES

[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] An Overview of XMPP. https://xmpp.org/about/technology-overview.html.
[5] Annual Day In The Life of the Internet (DITL) collection. https://www.dns-oarc.net/oarc/data/ditl.
[6] Apple Filing Protocol Concepts. https://developer.apple.com/library/content/documentation/Networking/Conceptual/AFP/Concepts/Concepts.html.
[7] ASCII Table and Description. http://www.asciitable.com/.
[8] Asterisk custom communications for VoIP. http://www.asterisk.org/.
[9] Autodiscover for Exchange. https://msdn.microsoft.com/en-us/library/office/jj900169(v=exchg.150).aspx.
[10] Automount NFS in OS X. https://yourmacguy.wordpress.com/2012/06/29/osx-automount/.
[11] Baïkal: Cal and CardDAV server based on sabre/dav. http://sabre.io/baikal/.
[12] BitTorrent Protocol. http://www.morehawes.co.uk/the-bittorrent-protocol.
[13] Bonjour API Architecture. https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/NetServices/Articles/programming.html.
[14] Bonjour: Apple's implementation of zero-configuration networking protocols. https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/NetServices/Introduction.html.
[15] Bonjour service types used in Mac OS X. https://developer.apple.com/library/content/qa/qa1312/_index.html.
[16] Building an Enterprise Root Certification Authority in Small and Medium Businesses. https://msdn.microsoft.com/en-us/library/cc875810.aspx.
[17] Calendaring Extensions to WebDAV (CalDAV). https://tools.ietf.org/html/rfc4791.
[18] CardDAV: vCard Extensions to Web Distributed Authoring and Versioning (WebDAV). https://tools.ietf.org/html/rfc6352.
[19] Chromes startup random DNS queries tracked in, and polluting users Google Web History. https://bugs.chromium.org/p/chromium/issues/detail?id=47262.
[20] Comcast's IPv6 Information Center. http://www.comcast6.net/.
[21] Comparison between NTLM and Kerberos. https://highfromtea.wordpress.com/tag/ntlmssp/.
[22] Configure Email Accounts with Outlook. https://support.marcaria.com/hc/en-us/articles/215526083-Configure-Email-Accounts-with-Outlook.
[23] 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.
[24] Configuring Pivotal Cloud Foundry SSL Termination for vSphere Deployments. https://docs.pivotal.io/pivotalcf/1-7/opsguide/ssl-term.html.
[25] Configuring the Commerce Server Network. https://msdn.microsoft.com/en-us/library/aa545742(v=cs.70).aspx.
[26] DNS-Based Authentication of Named Entities (DANE). https://tools.ietf.org/html/rfc6698.
[27] DNS-Based Service Discovery. https://tools.ietf.org/html/rfc6763.
[28] DNS Long-Lived Queries. https://tools.ietf.org/html/draft-sekar-dns-llq-01.
[29] Download RubyGems. https://rubygems.org/pages/download.
[30] Dynamic Updates in the Domain Name System (DNS UPDATE). https://tools.ietf.org/html/rfc2136.
[31] Edge Server environmental requirements in Skype for Business Server 2015. https://technet.microsoft.com/en-us/library/mt346415.aspx.
[32] ejabberd: robust, massively scalable and extensible XMPP server. https://www.ejabberd.im/.
[33] File Transfer Protocol (FTP). https://tools.ietf.org/html/rfc959.
[34] FTP Security Extensions. https://tools.ietf.org/html/rfc2228.
[35] Google open resolver IP addresses. https://developers.google.com/speed/public-dns/docs/using.
[36] Hacking Time Machine. https://dreness.com/blog/archives/48.
[37] HTTP Authentication: Basic and Digest Access Authentication. https://tools.ietf.org/html/rfc2617.
[38] HTTP Extensions for Distributed Authoring (WEBDAV). https://tools.ietf.org/html/rfc2518.
[39] HTTP Over TLS. https://tools.ietf.org/html/rfc2818.
[40] Hypertext Transfer Protocol – HTTP/1.1. https://tools.ietf.org/html/rfc2616.
[41] IANA Service Name and Transport Protocol Port Number Registry. http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml.
[42] 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.
[43] ICANN: Mitigating the Risk of DNS Namespace Collisions Phase One. https://www.icann.org/news/announcement-2-2014-06-10-en.
[44] ICANN Study: Name Collision in the DNS. https://www.icann.org/en/system/files/files/name-collision-02aug13-en.pdf.
[45] Internet Printing Protocol/1.1: Encoding and Transport. https://tools.ietf.org/html/rfc2910.
[46] Intra-Site Automatic Tunnel Addressing Protocol (ISATAP). https://tools.ietf.org/html/rfc5214.
[47] Kerberos: The Network Authentication Protocol. http://web.mit.edu/kerberos/.
[48] kpasswd - MIT Kerberos Documentation. https://web.mit.edu/kerberos/krb5-1.13/doc/user/user_commands/kpasswd.html.
[49] Let's Encrypt Certificate Authority. https://letsencrypt.org/.
[50] Lightweight Directory Access Protocol (LDAP): The Protocol. https://tools.ietf.org/html/rfc4511.
[51] Linux ISATAP Setup. http://www.litech.org/isatap/.
[52] macOS Xgrid. http://www.apple.com/server/macosx/technology/xgrid.html.
[53] Microsoft Key Management Services (KMS). http://help.unc.edu/help/microsoft-key-management-services-kms/.
[54] Microsoft TechNet: SRV Resource Records. https://technet.microsoft.com/en-us/library/cc961719.aspx.
[55] Name Server API? https://developer.dnsimple.com/v1/nameservers/.
[56] Network Time Protocol Version 4: Protocol and Algorithms Specification. https://tools.ietf.org/html/rfc5905.
[57] OpenAFS. http://www.openafs.org/.
[58] 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.
[59] Page Description Language. http://printwiki.org/Page_Description_Language.
[60] Picture Transfer Protocol (PTP). http://www.imaging.org/ist/resources/standards/ptp-standards.cfm.
[61] Post Office Protocol - Version 3. https://tools.ietf.org/html/rfc1939.
[62] Required DNS Records for Automatic Client Sign-In. https://technet.microsoft.com/en-us/library/bb663700(v=office.12).aspx.
[63] REST Resource Naming Guide. http://restfulapi.net/resource-naming/.
[64] RFC 5214: Intra-Site Automatic Tunnel Addressing Protocol (ISATAP). https://tools.ietf.org/html/rfc5214.
[65] RFC 5424. https://tools.ietf.org/html/rfc5424.
[66] Security Mechanism Agreement for the Session Initiation Protocol (SIP). https://tools.ietf.org/html/rfc3329.
[67] Server Message Block Overview. https://technet.microsoft.com/en-us/library/hh831795(v=ws.11).aspx.
[68] Session Traversal Utilities for NAT (STUN). http://www.voip-info.org/wiki/view/STUN.
[69] SFTP - The Modern FTP. https://www.ssh.com/ssh/sftp/.
[70] Simple Mail Transfer Protocol. https://tools.ietf.org/html/rfc2821.
[71] SIP: Session Initiation Protocol. https://tools.ietf.org/html/rfc3261.
[72] SSLsplit - Transparent SSL/TLS Interception. https://www.roe.ch/SSLsplit.
[73] Static Analysis vs Dynamic Analysis in Software Testing. http://www.testingexcellence.com/static-analysis-vs-dynamic-analysis-software-testing.
[74] Static Content Subdomain. https://halfelf.org/2015/static-content-subdomain.
[75] The Case Against DNSSEC. http://www.circleid.com/posts/070814_case_against_dnssec/.
[76] The DNS Operations, Analysis, and Research Center (DNS-OARC). https://www.dns-oarc.net/.
[77] The Remote Framebuffer Protocol. https://tools.ietf.org/html/rfc6143.
[78] Tutorial: Run Your Own Gem Server. http://guides.rubygems.org/run-your-own-gem-server/.
[79] US-CERT Technical Alert (TA16-144A): WPAD Name Collision Vulnerability. https://www.us-cert.gov/ncas/alerts/TA16-144A.
[80] Using Digest Authentication as a SASL Mechanism. https://tools.ietf.org/html/

rfc2831.
[81] Web Authentication Proxy Configuration Example. http://www.cisco.com/c/en/us/support/docs/wireless-mobility/wlan-security/116052-config-webauth-proxy-00.html.
[82] Zero Configuration Networking (Zeroconf). http://www.zeroconf.org.
[83] The New gTLD Program. https://newgtlds.icann.org/en/about/program, 2013.
[84] New delegated TLD strings. http://newgtlds.icann.org/en/program-status/delegated-strings, 2017.
[85] 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 S&P*, 2016.
[86] S. Castro, D. Wessels, M. Fomenkov, and K. Claffy. A Day at the Root of the Internet. volume 38, pages 41–46. ACM, 2008.
[87] 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 S&P*, 2016.
[88] S. Cheshire and M. Krochmal. Multicast DNS. rfc6762, 2013.
[89] 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.
[90] 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.
[91] P. Gauthier, J. Cohen, M. Dunsmuir, and C. Perkins. The Web Proxy Auto-Discovery Protocol. *Internet draft, IETF*, 1999.
[92] 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.
[93] S. Goldberg, M. Naor, D. Papadopoulos, L. Reyzin, S. Vasant, and A. Ziv. NSEC5: Provably Preventing DNSSEC Zone Enumeration. In *ISOC NDSS*, 2015.
[94] 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.
[95] B. S. Kaliski Jr. and A. Mankin. US Patent Application 20150256424: Name Collision Risk Manager. http://www.freepatentsonline.com/y2015/0256424.html.
[96] A. R. Kang, S. H. Jeong, S. Y. Ko, K. Ren, and A. Mohaisen. Transparency in the New gTLD Era: Evaluating the DNS Centralized Zone Data Service. In *IEEE HotWeb*, 2016.
[97] 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.
[98] C. Lever, R. Walls, Y. Nadji, D. Dagon, P. McDaniel, and M. Antonakakis. Domain-Z: 28 Registrations Later. In *IEEE S&P*, 2016.
[99] Mockapetris, Paul. Domain Names - Implementation and Specification. rfc1035, 2004.
[100] A. Mohaisen and K. Ren. Leakage of. onion at the DNS Root: Measurements, Causes, and Countermeasures. 2017.
[101] E. Osterweil, D. McPherson, and L. Zhang. The Shape and Size of Threats: Defining a Networked System's Attack Surface. In *ICNP*, 2014.
[102] 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.
[103] S. Son and V. Shmatikov. The Hitchhiker's Guide to DNS Cache Poisoning. In *Security and Privacy in Communication Networks*. Springer, 2010.

APPENDIX

A Non-registered Service Name Analysis

Automatic labeling script. In non-registered service measurement, 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 §4.2 a large portion of them are actually irrelevant, e.g., random strings potentially sent by Chrome for infobar customization [19].

To effectively identify valid non-registered service names from the extremely large candidate string set (78.5 million from our measurement in §4.2), 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 [31, 42]. 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.

Analysis results. Fig. 3 shows the automatic labeling results. As shown, for the top 50 server string candidates, the majority (60%) of them are popular names that at least have some 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 paper, 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.

B Exploit Case Study: 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 remember 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 [8], and ejabberd for PSI [32]. During the account logon, Linphone sends password in MD5 hash without server authentication, which is the designed SIP authentication method [71]. For PSI, the authentication process by default uses DIGEST-MD5, which provides server authentication [37]. 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 (§5.3 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

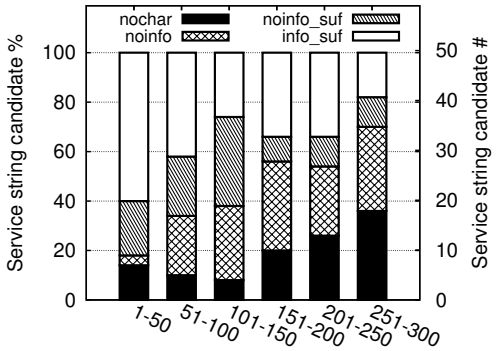

Figure 3: Automatic labeling results for the top 300 non-registered service string candidates.

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 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.

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

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.

C Exploit Case Study: 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 [11]. 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 [37]. 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.

Service name	Service description and documentation
wpad (N)	Web Proxy Auto-Discovery (WPAD) protocol, used by web clients to locate web proxies [91]
isatap (N)	Intra-Site Automatic Tunnel Addressing Protocol (ISATAP), used by dual-stack (IPv6/IPv4) clients to automatically tunnel IPv6 packets in IPv4 networks [64]
proxy (N)	Popular first label for a web proxy server [81]
vlmcs (N)	Microsoft Key Management Services (KMS), used by Microsoft clients to automatically activate volume license editions of Microsoft Windows and Office [53]
ntp	Network Time Protocol (NTP), used by clients to synchronize computer clocks in the Internet [56]
ns* (N), alt* (N)	Popular first label for a DNS name server [55, 58]
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 [12]
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 [28]
www* (N), api (N), static (N), share (N), cf (N)	Popular first labels for a server hosting web content, web elements, and web operations [10, 23, 24, 63, 74]
http, https	Hypertext Transfer Protocol, used by web clients to browse web content [39, 40]
stun	Session Traversal Utilities for NAT (STUN), used by clients to get the IP address and port allocated to it by a NAT [68]
ptp	Picture Transfer Protocol (PTP), used by clients to transfer images from digital cameras [60]
dpap	Digital Photo Access Protocol (DPAP), used by iPhoto clients to share photos starting in iPhoto 4.0 [15]
kerberos	The Kerberos service, used by clients to perform network authentications [47]
rubygems	Used by RubyGems, the package manager in Ruby to help clients download Ruby coding libraries [29]
gc (N)	Used by clients to locate a Microsoft Global Catalog (GC) server in a domain [54]
ldap, ldaps	Lightweight Directory Access Protocol (LDAP), used by clients to access directory services [50]
carddav, carddavs	vCard Extensions to WebDAV (CardDAV), used by clients to access, manage, and share contact information [18]
caldav, caldavs	Calendaring Extensions to WebDAV (CalDAV), used by clients to access, manage, and share calendaring information [17]
dns-update	Dynamic Updates in the DNS, used by DNS clients to add or delete resource records in DNS zones [30]
afs3-vlserver	Used by clients to access the Andrew distributed file system (AFS) [57]

Service name	Service description and documentation
adisk	Used by Apple Time Machine clients to perform automatic disk discovery [36]
smb	Server Message Block (SMB), used by clients to share file over a network [67]
afpovertcp	Apple Filing Protocol Over TCP, used by clients to share file over a network [6]
ftp, sftp-ssh	File Transfer Protocol (FTP), used by clients to transfer file over a network [33, 69]
webdav	HTTP Extensions for Distributed Authoring (WebDAV), used by web clients to manage remote web content [38]
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 [77]
ssh	Secure Shell (SSH) protocol, used by clients to access a remote computer [15]
eppc	Used by clients to send remote Apple events [15]
telnet	Used by clients to access a remote computer [15]
kpasswd	Used by clients to change Kerberos passwords [48]
airport	Used by clients to configure a AirPort base station [15]
servermgr	Used by macOS clients to manage macOS servers [15]
autodiscover (N), outlook (N)	Exchange Autodiscover service, used by clients to automatically configure Microsoft Exchange [9]
mail*	Used by clients to locate POP3 or SMTP mail servers [22]
pop3	Post Office Protocol (POP), used by clients to locate POP mail servers [61]
smtp	Simple Mail Transfer Protocol (SMTP), used by clients to locate SMTP mail servers [70]
sip, sips, sipinternaltls (N), sipinternal (N), sipexternal (N)	Session Initiation Protocol (SIP), used by clients to create, modify, and terminate Internet telephone call sessions [62, 71]
xmpp-server, xmpp-client	Extensible Messaging and Presence Protocol (XMPP), used by clients to manage sessions for messaging, network availability, and request-response interactions [4]
printer	Used by client to locate network printers [15]
riousbprint	Used by the AirPort base station to share USB printers [15]
pdl-datastream	Used by client to locate network printers supporting Page Description Language (PDL) [59]
ipp	Internet Printing Protocol (IPP), used by clients to locate network printers supporting IPP [45]
scanner	Used by macOS clients to locate network scanners [15]
ica-networking	Used by macOS Image Capture app to share cameras [15]
xgrid	Used by macOS clients to locate Apple xGrid agents for distributed computing [52]
syslog	The Syslog protocol, used by clients to send and receive event notification messages [65]

Table 6: Descriptions and documentations of the exposed internal network services. N denotes non-registered service.

D 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 [87] to the general form of name collision attacks in this paper.

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) [87]. 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 paper. 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 becomes 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.