Detecting and Measuring Security Risks of Hosting-Based Dangling Domains

MINGMING ZHANG, Tsinghua University, China
XIANG LI, Tsinghua University, China
BAOJUN LIU*, Tsinghua University and Quan Cheng Laboratory, China
JIANYU LU, QI-ANXIN Technology Research Institute, China
YIMING ZHANG*, Tsinghua University, China
JIANJUN CHEN, Tsinghua University and Zhongguancun Laboratory, China
HAIXIN DUAN, Tsinghua University and Quan Cheng Laboratory, China
SHUANG HAO, University of Texas at Dallas, USA
XIAOFENG ZHENG, Tsinghua University and QI-ANXIN Technology Research Institute, China

Public hosting services provide convenience for domain owners to build web applications with better scalability and security. However, if a domain name points to released service endpoints (e.g., nameservers allocated by a provider), adversaries can take over the domain by applying the same endpoints. Such a security threat is called "hosting-based domain takeover". In recent years, a large number of domain takeover incidents have occurred; even well-known websites like the subdomains of *microsoft.com* have been impacted. However, until now, there has been no effective detection system to identify these vulnerable domains on a large scale. In this paper, we fill this research gap by presenting a novel framework, HostingChecker, for detecting domain takeovers. Compared with previous work, HostingChecker expands the detection scope and improves the detection efficiency by: (i) systematically identifying vulnerable hosting services using a semi-automated method; and (ii) effectively detecting vulnerable domains through passive reconstruction of domain dependency chains. The framework enables us to detect the subdomains of Tranco sites on a daily basis. We evaluate the effectiveness of HostingChecker and eventually detect 10,351 subdomains from Tranco Top-1M apex domains vulnerable to domain takeover, which are over 8× more than previous findings. Furthermore, we conduct an in-depth security analysis on the affected vendors, like Amazon and Alibaba, and gain a suite of new insights, including flawed implementation of domain ownership validation. Following responsible disclosure processes, we have reported issues to the security response centers of affected vendors, and some (e.g., Baidu and Tencent) have adopted our mitigation.

CCS Concepts: • **Networks** → **Cloud computing**; • **Security and privacy** → **Network security**.

Additional Key Words and Phrases: public hosting service; domain takeover

*Corresponding Authors: Baojun Liu (lbj@tsinghua.edu.cn) and Yiming Zhang (zhangyiming@tsinghua.edu.cn).

Authors' addresses: Mingming Zhang, zmm18@mails.tsinghua.edu.cn, Tsinghua University, Beijing, China; Xiang Li, Tsinghua University, Beijing, China; Baojun Liu, Tsinghua University and Quan Cheng Laboratory, Beijing, China; Jianyu Lu, QI-ANXIN Technology Research Institute, Beijing, China; Yiming Zhang, Tsinghua University, Beijing, China; Jianjun Chen, Tsinghua University and Zhongguancun Laboratory, Beijing, China; Haixin Duan, Tsinghua University and Quan Cheng Laboratory, Beijing, China; Shuang Hao, University of Texas at Dallas, Richardson, Texas, USA; Xiaofeng Zheng, Tsinghua University and QI-ANXIN Technology Research Institute, Beijing, China.

2476-1249/2023/3-ART9
https://doi.org/10.1145/3579440

ACM Reference Format:
Mingming Zhang, Xiang Li, Baojun Liu, Jianyu Lu, Yiming Zhang, Jianjun Chen, Haixin Duan, Shuang Hao, and Xiaofeng Zheng. 2023. Detecting and Measuring Security Risks of Hosting-Based Dangling Domains. *Proc. ACM Meas. Anal. Comput. Syst.* 7, 1, Article 9 (March 2023), 28 pages. https://doi.org/10.1145/3579440

1 INTRODUCTION

Domain names serve to identify Internet resources and are adopted in multiple mainstream security paradigms, such as digital certificate application [40] and email trustworthiness validation [33, 46]. Several high-profile breaches, however, have shown the increasing prevalence and power of domain takeover attacks. Adversaries can exploit the domains that are outside of their authority for cybercrimes such as malware distribution, phishing, and certificate forging [29, 48, 52, 60]. Dangling DNS records, in which domain names are resolved to an expired domain or a discontinued service, are the major cause of domain takeover [48]. Attackers can compromise domains of prominent companies (e.g., Microsoft [2]), media organizations (e.g., American News [3]), and political websites (e.g., Donald Trump [28]) by exploiting these dangling DNS records. We refer to domains with dangling DNS records as "dangling domains" in this paper.

Hosting-based dangling domains ($D_{vulhost}$). A major source of dangling domains is discontinued hosting services. Large organizations are increasingly hosting domains on public platforms, such as cloud storage and content delivery networks (CDN), to improve scalability and security. The platforms allocate ephemeral resources, named "service endpoints", to serve customers' domain names. Meanwhile, customers should prove their domain ownership and resolve their domains to the allocated endpoints by configuring DNS records (e.g., `CNAME` and `NS`). This process is known as "custom domain connection". If a service is discontinued, the endpoints allocated for it will be released and can be acquired by other customers. The released endpoints could be further exploited by adversaries to initiate domain takeover attacks if the platform fails to correctly verify the domain ownership and domain owners forget to purge obsolete DNS records.

Motivation. Although research efforts [29, 48, 58] have been devoted to mitigating the security threats of domain takeover, incidents are still on the rise, increasing by 25% from 2020 to 2021 [34], particularly when public services are involved. It motivates us to explore why domain takeovers occur ceaselessly. In the end, we find two research gaps in detecting $D_{vulhost}$. First, there is no generic method for discovering third-party hosting services. The service types and domain hosting strategies offered by public providers are highly diverse [12], with no uniform features for identifying vulnerable services. Therefore, previous studies only focused on user-reported cases [36, 43] with ad hoc analysis [58]. Second, an efficient detection system is absent for quickly digging out $D_{vulhost}$ across enormous domains in the wild. Because Tranco Top-1M domains[1] have tens of millions of subdomains, timely detecting $D_{vulhost}$ among them is challenging. Prior detection methods mainly relied on active DNS lookups, which are rather slow. This brings down detection efficiency [48].

Our study. We introduce HostingChecker, a novel framework that can assist in spotting hosting services and efficiently detect vulnerable domains hosted on discontinued services (Section 4). It overcomes the aforementioned challenges in the following ways. For the first challenge, we design a semi-automated service discoverer to expand the detection scope. We observe that public hosting services share general domain features that can be automatically mined from passive DNS (PDNS) data. Customers must resolve (i.e., point) their domains to the service endpoint domains to launch a hosting service, which is referred to as the "domain dependencies". Meanwhile, endpoints of the same service use identical naming conventions,[2] which we term "endpoint patterns". To this end, we identify endpoint domain candidates with high domain dependencies and extract endpoint

[1]Tranco is a research-oriented top site ranking list that mainly consists of apex domains. https://tranco-list.eu/methodology
[2]`<prefix>.<service>.<location>.amazonaws.com`, e.g., `alice.s3.us-east-1.amazonaws.com`.

patterns using a novel Domain Suffix Tree. For the second challenge, HostingChecker employs a passive method to efficiently detect vulnerable domains. Instead of actively performing DNS queries, it reconstructs domain resolution chains passively by leveraging a local PDNS dataset.

Experiments and findings. We implement HostingChecker on a PDNS dataset containing 101 billion DNS responses. We then conduct a large-scale and longitudinal measurement study, i.e., 101 rounds of measurements on the subdomains under Tranco Top-1M apex domains [15] from Dec 16, 2021, to Jul 28, 2022 (Section 5). HostingChecker takes approximately one day to process each measurement round. Compared to previous studies, HostingChecker has a higher detection efficiency and detects more vulnerable domains than other programs [36, 43, 48, 58]. Below, we highlight the major findings of our analysis:

(i) A holistic characterization of public hosting services. Utilizing HostingChecker, we discover 65 vulnerable services that can be leveraged for domain takeover, with 34 newly reported by us. Our discovered vulnerable hosting services cover 52 public vendors, including popular vendors such as Alibaba [4], Amazon [5], and Cloudflare [10]. Based on these services, we delve into their domain hosting strategies to gain more insights for mitigating domain takeover threats. Specifically, we find that public service providers employ diverse domain connection methods, e.g., by configuring a DNS `CNAME` record. However, most (i.e., 7 out of 9) methods are vulnerable and could be exploited for domain takeover. Although several providers have deployed domain ownership validation (DOV) strategies for defense, we discover 4 flawed implementations that can bypass DOV and affect the top 20 hosting vendors. In particular, some previously reported non-exploitable services [36] are found to be exploitable again. For responsible disclosure, we report vulnerabilities to affected vendors and receive confirmation from ten of them, including Amazon, Tencent, and Huawei.

(ii) A longitudinal measurement of $D_{vulhost}$ among high-profile domain names. Through a 7-month measurement study, we find that hosting-based domain takeover threats are still prevalent. In detail, 114,063 (1.0%) of all tested domains have been hosted on vulnerable services and 10,351 are $D_{vulhost}$ (8× more than previous findings). In addition, popular apex domains are particularly susceptible to such threats since they often deploy subdomains to public services. In our results, the discovered $D_{vulhost}$ belong to 2,096 popular apex domains, including reputable universities (e.g., Stanford and Rice) and companies (e.g., Baidu, Huawei, and Marriott). Through periodic and large-scale measurements, we find that such threats appear frequently and are long-lasting. Specifically, we observe that 270 new $D_{vulhost}$ emerge per week on average, and 60% $D_{vulhost}$ remain vulnerable for over 5 days (36.3% for over a month), leaving a substantial window of time for attacks. Moreover, by analyzing PDNS logs, we find that 45.5% $D_{vulhost}$ still receive queries from 70 million client IPs. Thus, timely detection is greatly needed to reduce the attack surface.

Contributions. In this study, we make the following contributions:

- *Detection system.* We present a novel and effective framework, HostingChecker, that can perform daily checks on tens of millions of subdomains. Because of its high efficiency and coverage, it can help the security community mitigate hosting-based domain takeover risks.
- *Extensive measurements.* We deploy HostingChecker on a large-scale PDNS dataset and conduct a 7-month longitudinal measurement on Top-1M's subdomains. We detect 10,351 $D_{vulhost}$, which is 8 times more than the previous study.
- *Systematic service inspection and threat analysis.* We present the holistic characteristics of public hosting services. We discover 65 vulnerable services, including 34 new ones, and new security flaws in hosting practices. In the end, we provide an in-depth understanding of the reasons for hosting-based domain takeover and discuss best practices to support the community in mitigating threats.

2 BACKGROUND

2.1 DNS Basics

Domain name space. A Fully Qualified Domain Name (FQDN) like `foo.example.com` is presented as a hierarchical tree, and each layer is organized within a DNS zone. The top of the hierarchy is the root zone. Below the root is the Top-Level Domain (TLD, e.g., `.com`). Under TLDs, Second-Level Domains (SLDs, e.g., `example.com`) are open for registration from registrars (e.g., GoDaddy [11]).

The DNS community generally uses public suffixes [13] or effective TLDs (eTLDs) to refer to the TLDs that are directly operated by registrars, such as `.com` and `.co.uk`. Besides, it uses apex domains, or apexes (i.e., eTLD+1), to represent the domains that registrants can apply for, such as `example.com`. Registrants can create any subdomains for the apexes they control, e.g., `foo.example.com`. For convenience, we refer to all FQDNs under an apex domain as its subdomains.

DNS resource records. In the domain name space, resource records (RRs) are the information entries associated with a certain domain name in DNS zone files. RRs associate domain names with their corresponding resolution results, which are referred to as the `data` fields of the RRs. The authoritative DNS (aDNS) servers of a domain are responsible for managing the DNS RRs and translating the domain name to other resources. Typically, domain owners can configure the aDNS servers and control the domain resolution process by setting DNS RRs. Common RRs include: `A` records (IPv4 addresses, e.g., `1.2.3.4`), `CNAME` records (domain alias, e.g., `alias.example.com`), `NS` records (authoritative name servers), and `MX` records (email servers).

2.2 Dangling DNS Records

Dangling DNS records are a collection of DNS RRs in which the targeted resources (i.e., the `data` fields) are invalid, having expired, been released, or never been deployed. Previous work has identified four categories of security-sensitive dangling records [48]:

- Dangling A records. They occur if some domains point to an IP address that can be acquired by any person. For example, the IP address is in a shared IP pool of public cloud instances (e.g., Amazon EC2 and Microsoft Azure) and is deprovisioned.
- Dangling CNAME records. They occur when a canonical domain in `CNAME` records expires or becomes available on a public hosting platform.
- Dangling NS and MX records. They are also unsafe when name servers and mail servers can be controlled due to expiration or troublesome service hosting. The threats caused by dangling NS and MX records are more severe since all domains delegated to these vulnerable servers can be taken over.

By exploiting these unsafe dangling records, adversaries can manipulate the targeted resources in the DNS RRs and take over the domains that are not under their control.

3 PUBLIC HOSTING SERVICE AND STUDY SCOPE

3.1 Public Hosting Service

Because of their scalability, reliability, and security, public hosting platforms have become an attractive option for deploying Internet applications. Statistics show that over 1.7M domains have deployed websites using hosting platforms [17]. We simplify the steps for hosting a custom domain on a platform in Figure 1(a). Assume a customer, Alice, subscribes to a hosting service and attempts to add her domain, `custom.alice.com`, to the platform (①). The platform must first validate that Alice is the domain owner (i.e., domain ownership validation). To enable this, Alice must configure a challenge token temporarily issued by the provider, such as by creating a `TXT` record on the authoritative server (②). After verifying the challenge record (③), the platform will allocate network

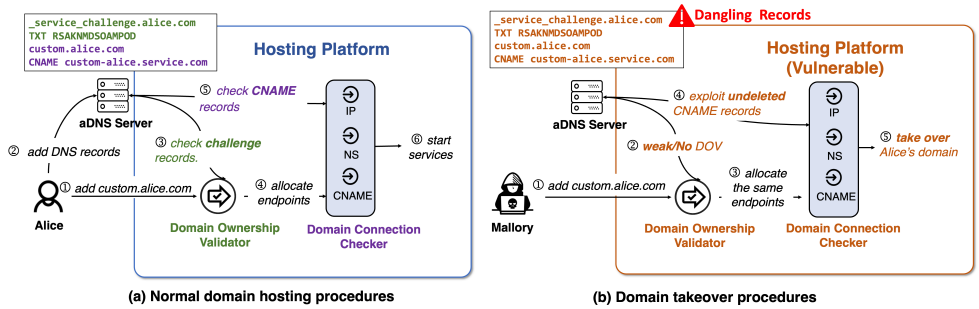


Fig. 1. Procedures of domain hosting and threat model of hosting-based domain takeover.

resources known as service endpoints to serve Alice's domain (④). Endpoints could be name servers, IP addresses, or subdomains under the apex domains controlled by the platform. Assuming the platform assigns a CNAME endpoint (e.g., `custom-alice.service.com`) to `custom.alice.com`, it will then begin delivering content using its resource pool (⑥) after confirming that the CNAME record exists (⑤). It indicates the domain connection is successful.

Domain ownership validation (DOV). The person who registers a domain name is the domain owner and can manage the domain's authoritative server. Customers can claim domain ownership on a hosting platform in two ways. (i) DNS-based verification: providers generate a challenge token and ask customers to configure it in a DNS record (e.g., CNAME or TXT). (ii) Web-based verification: providers ask customers to upload a file containing a challenge token to a certain directory on the website. Many Internet services now demand valid domain ownership, but there are no standard practices [55].

Domain connection. We refer to the processes ④ and ⑤ as domain connection for simplicity. Customers must update their DNS servers by adding DNS RRs pointing to the assigned endpoints. Common connection methods include (i) delegating custom domains to the given name servers via NS records, (ii) pointing the domains to CNAMEs managed by the provider, (iii) creating A records that point to the services' IP pool, or (iv) combining the aforementioned methods.

3.2 Our Study Scope: Hosting-based Domain Takeover

Threat model. We focus on the threat model that attackers try to take over victim domains by exploiting vulnerable public hosting services. In the following, we will refer to this threat as "hosting-based domain takeover" and relevant dangling domains as $D_{vulhost}$. Attackers can manipulate the endpoints of the victim domains by deploying a new service on vulnerable platforms.

Definition of vulnerable service. In practice, providers may combine DOV and domain connection into a single validation step, such as assigning unique endpoints to different customers rather than verifying the presence of challenge tokens via an additional step. They choose endpoints at random from a pool to ensure that each customer has a unique allocation. Then the customer who successfully connects a custom domain to the given endpoints is considered the domain owner.

In this situation, a hosting service is considered vulnerable to domain takeover if its DOV and domain connection methods are flawed. We categorize the flaws into three groups. First, there is no DOV for a hosted domain. Second, the employed DOV strategies can be bypassed. Third, the platform performs DOV by allocating random endpoints and requesting customers to point domains to them, however, attackers may apply the same endpoints as victims to pass DOV (this is known as endpoint collision). For example, if the platform chooses endpoints from a small candidate pool, attackers may perform endpoint collisions by simply creating many accounts. Attackers can claim

Table 1. Comparison of representative work.

Paper/Report	Service Discovery	DNS Lookup	# Vulnerable Services	Detection Efficiency	# $D_{vulhost}$
Liu et al. [48]	manual	active	9	low	467
Squarcina et al. [58]	[36, 48]	active	17	low	1,260
HostingChecker	PDNS	passive	65	high	**10,351**

ownership of any domain on a public platform and manipulate it by abusing the aforementioned weaknesses.

Attack steps. Figure 1(b) illustrates the threat model. If Alice unsubscribes from a third-party service without purging CNAME records, her custom domain, `custom.alice.com`, becomes dangling. Assuming the platform's DOV is misconfigured (②), an attacker, Mallory, can manipulate Alice's domain (①) by applying the same endpoints for domain connection (③). After passing the CNAME check (④), the provider begins offering services to Mallory (⑤). Mallory can then take over `custom.alice.com` and manipulate its resources, such as building phishing websites.

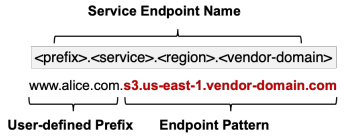
Real-world examples. Researchers discovered hundreds of Microsoft subdomains that might be taken over [2]. For example, if a custom domain name has a dangling `CNAME` record pointing to a non-existent subdomain under `azurewebsites.net`, an adversary can claim the ownership of that domain and launch any services via the Microsoft Azure portal platform.

3.3 Comparison with Related Work

According to published research, domain takeovers can occur due to three factors: (i) Weak security policies adopted by domain registrars/registries (e.g., Risky BIZness [20], domain sinkhole [30], and lame delegations [19]); (ii) Flaws in protocol design and implementation (e.g., orphan DNS servers [45], ghost domains [44], and phoenix domain [47]); (iii) Dangling DNS records. A dangling record occurs when a domain name points to an expired domain [48, 58, 59], a discontinued hosting service [48, 58], or a deprovisioned cloud instance [29].

Dangling DNS records caused by discontinued hosting services are the topic of this study, and the most relevant works are given in Table 1. Liu et al. proposed the threat model of domain takeover that abuse different types of dangling DNS records. They tested the threat on 1M apex domains and 57k subdomains of 14k apexes [48]. Squarcina et al. explored the security implications of related domains that are controlled by adversaries. They also empirically evaluated domain takeover threats over 50k apexes and associated 26M subdomains [58]. Both studies used a one-time measurement on a restricted number of target domains. Furthermore, they only performed ad hoc examinations on a few vulnerable services (9 and 17, respectively). They did not aim to systematically study the flaws in hosting provider policies that can be exploited to take over subdomains. Thus, the majority of observed dangling domains were caused by expired domains and deprovisioned cloud instances. Developing a practical detection system is also beyond the scope of their research. In addition to $D_{vulhost}$, Borgolte et al. investigated released cloud instances from two large cloud providers [29]. They discovered domains pointing to allocatable IP pools and demonstrated how expired domains might be exploited to issue fraudulent SSL certificates.

Our research varies from previous work in two major respects. First, we systematically explore the ecosystem of public hosting services in order to discover more exploitable services that can assist in the detection of $D_{vulhost}$. Existing work, however, was based solely on a few reported cases [36, 43]. Second, we build an efficient and scalable system for detecting $D_{vulhost}$ on a wide scale. Aside from being more efficient than prior techniques, HostingChecker provides greater coverage of detected vulnerable domains and fewer false positives.



Service Endpoint Examples

Services	Endpoint Names (endpoint patterns)	Suffix Level
Aliyun OSS	alice.storage.com.oss-cn-hongkong.aliyuncs.com	3
Amazon S3	a.b.c.d.s3.us-east-1.amazonaws.com ab-cd.s3.dualstack.us-gov-west-1.amazonaws.com	4 5
Github	abcd.github.io	2

Fig. 2. Examples of service endpoint names and endpoint patterns (red suffixes).

4 HOSTINGCHECKER: A DETECTION SYSTEM

4.1 Empirical Observations

Our high-level idea for detecting hosting-based domain takeover threats is straightforward: (i) identify hosting services, and (ii) discover vulnerable domains hosted on discontinued services.

As for the first task, after reviewing public hosting services reported in previous studies [48], we conclude that they exhibit special domain characteristics:

O1 **Similar endpoint patterns.** As illustrated in Figure 2, most services allow partially customizable endpoint domains [13, 21, 24]. Here, `<prefix>` is either a custom domain or a unique identifier (e.g., user-defined labels, or platform-generated labels that are relevant to user accounts); `<service>` and `<region>` represent service types and geolocation codes, which sometimes are merged into one label; `<vendor-domain>` are base domains managed by providers. We use an endpoint pattern to denote the longest endpoint suffix (i.e., `<service>.<region>.<vendor-domain>`) that is designated to a specific hosting service.

O2 **High domain dependency number (*DN*).** We claim that a custom domain depends on an endpoint domain if the former points to the latter via CNAME or NS RRs. In Figure 1, for example, `custom.alice.com` depends on `service.com`. Additionally, the domain dependency number (*DN*) of one endpoint pattern is defined as the number of apex custom domains that depend on the pattern. For example, $DN(service.com)$ is N in Figure 4. Given the large volume of customers, endpoint patterns of public hosting providers have a large *DN* since they can be depended on by numerous custom domains.

As for the second task, we find that vulnerable custom domains depend on the endpoint domains that exhibit the fingerprints (e.g., HTTP 404) of service discontinuation. Fetching web pages directly to validate fingerprints is error-prone, because many failures may trigger such notifications [64], and detectors may be unable to retrieve web contents if the endpoint domains become NXDOMAINs. To this end, we must inspect DNS settings (i.e., DNS records) to ensure that domains are hosted on services. Because PDNS has seen historical DNS lookups, we can reconstruct DNS resolution chains of detected domains. With this technique, we can enhance efficiency and reduce network resource consumption compared to active DNS resolution methods utilized in [48, 58].

The domain characteristics of public hosting services and the DNS chains of detected domains can be easily extracted from DNS traffic. It prompts us to automate our approach to hunting services and discovering vulnerable domains using PDNS logs.

4.2 System Workflow

In light of the above observations, we design a novel framework, HostingChecker, which leverages a PDNS dataset to perform service discovery and vulnerable domain identification. Because PDNS has been widely used in prior work [23, 29, 38, 39], we believe HostingChecker can be extended with any PDNS dataset. Figure 3 depicts the architecture of HostingChecker.

In Part 1, as a preparation component, we design a semi-automated service discoverer with four phases. First, the discoverer automatically harvests endpoint candidates that may belong

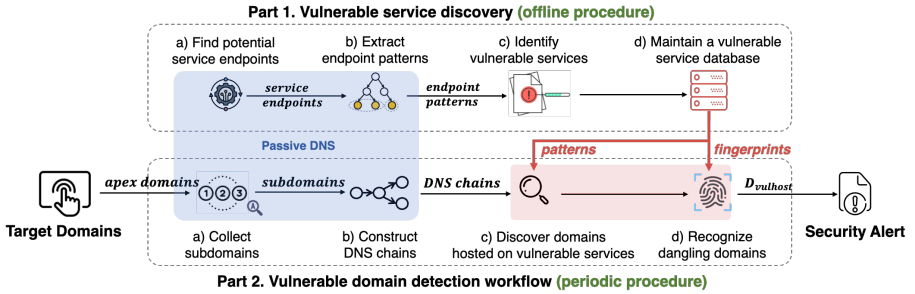


Fig. 3. Overview of HostingChecker modules. Step 1a, 1b, 2a, and 2b take PDNS as inputs and adopt the passive method, and step 3c performs active network probes. The output domain patterns and fingerprints of Part 1 are fed to the steps 2c and 2d.

to public hosting services based on O2 by calculating *DN*. Second, according to O1, it extracts endpoint patterns from these candidates by constructing a novel *Domain Suffix Tree*. In this phase, it traverses domains from the rightmost labels and merges those that have the same suffix. For example, `a.s3.service.com` and `b.s3.service.com` will be merged as `{a,b}.s3.service.com` and organized into a tree structure (Figure 5). The longest common suffixes are extracted as endpoint patterns. Third, it identifies service types and examines hosting policies to detect vulnerabilities. We can collect HTTP and DNS fingerprints indicating the discontinued vulnerable services during these steps. Fourth, all vulnerable endpoint patterns and the fingerprints are fed into a $D_{vulhost}$ detector.

The efficient $D_{vulhost}$ detector is illustrated in Part 2. It detects $D_{vulhost}$ by examining whether a domain name depends on a vulnerable and discontinued service. Given some apex domains to detect, the detector begins by gathering subdomains and DNS RRs from PDNS logs based on their query volume. Then it reconstructs the subdomains' DNS resolution chains by linking up DNS RRs. The detector checks all names in the chains to determine if they match the endpoint patterns obtained in Part 1, which indicates the domains are hosted on vulnerable services. Following that, it probes the hosted domains and inspects their HTTP and DNS fingerprints, which we have pre-acquired to detect a discontinued service. The system will generate a security warning if it finds a $D_{vulhost}$.

The system relies on PDNS as the only input in both modules, which can considerably enhance detection coverage and efficiency. The methodology and technical details are described below.

4.3 Discovering Vulnerable Services

Challenges and solutions. We attempt to recognize hosting services from DNS traffic by leveraging domain dependencies and special naming conventions. However, first, domain dependencies are not limited to public hosting services. For example, Internet corporations have self-built load balancers whose edge nodes likewise serve many domains and have similar naming patterns. Second, identical suffixes also appear in non-service domain names, like disposable domains used to convey a "one-time-signal" [31], which need to be discarded. Third, most providers do not publicly release their endpoint patterns that are extensively customized (e.g., containing service types and other identifiers). Thus, it is tough to find all service endpoint patterns by matching regular expressions.

To address the above challenges, we first observe that private load balancers used by providers have smaller *DNs* than public services, since they only serve the providers' domains under 1 or few apexes (Step 1). Second, because disposable domains are generated algorithmically (e.g., by using hash values of the same length) and are commonly queried once, we consider only domains with large query volumes to be service endpoints and exclude domains with fewer queries. Third, we

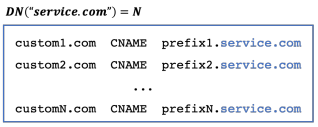


Fig. 4. Domain dependency (Step 1).

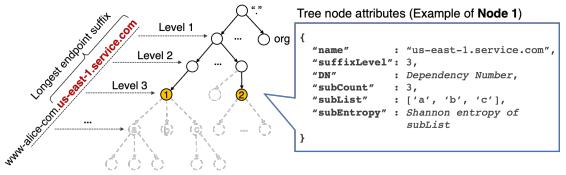


Fig. 5. Extract endpoint patterns via a domain suffix tree (Step 2).

propose a novel domain suffix tree to flexibly extract endpoint patterns (Step 2). The major steps are explained below.

Step 1: Finding service endpoint candidates. First, the discoverer extracts RRs (e.g., CNAME, NS, and MX) from PDNS with domain names in their data fields. In detail, it checks data field format and removes RRs containing wildcard domains, reverse DNS domains (e.g., 4.3.2.1.in-addr.arpa), and illegal strings according to the standard [50]. It also filters RRs of disposable domains with a low *Daily Query Volume* (*DQV*), the average query number of a domain within a day. The discoverer then generates *DNs* for all data fields. It chooses endpoint candidates by comparing *DNs* to a threshold, *minimum DN* (*MinDN*), that determines the popularity (i.e., the custom domain volume) of selected services. If *MinDN* is large, only impactful services with a high amount of customers can be evaluated; otherwise, the service vulnerabilities pose minimal security risks due to few customers.

Step 2: Extracting endpoint patterns via a domain suffix tree. Note that we are unaware of the domain suffix levels for service endpoints. Suffixes with the same levels may serve various services. "s3.us-east-2.amazonaws.com" and "elb.us-east-2.amazonaws.com", for example, respectively serve Amazon S3 and AWS Elastic Load Balancing. To flexibly extract the longest suffix patterns for each service's endpoints, i.e., endpoint patterns, we build a *Domain Suffix Tree* (*DST*) and extract the name groups that exhibit similar features.

Domain tree constructing. To begin with, we convert all endpoint names into a domain tree structure. Figure 5 depicts a tree diagram and a transformation example for the domain, "www-alice-com.us-east-1.service.com". The tree has "." as the root. Its child nodes in level 1 are TLDs (i.e., .com), followed by apex domains or eTLD+1 (i.e., service.com) in level 2, eTLD+2 in level 3, and so on. Referring to Figure 5, each $Node_i$ in node set $Node$ has six attributes: (i) name: the endpoint suffix name at the current level; (ii) suffixLevel: the label number of name; (iii) DN: dependency number of name; (iv) subCount: the children node number $k (k \in \mathbb{N})$; (v) subList: the children's name list L; (vi) subEntropy: the Shannon entropy [66] of all l_i in L, i.e., $H(L) = -\sum_{i=1}^{k} p_i log p_i$, where p_i denotes the proportion of each l_i, and an entropy value denotes the randomness of a children node.

Domain tree pruning. We traverse the domain tree and prune it from the bottom up using a depth-first search (DFS) algorithm. We first remove all leaf nodes that represent FQDNs of all endpoint domains. Then, for each $Node_i$ with a suffixLevel larger than 2 (i.e., subdomains beneath apexes), we merge $Node_i$'s children nodes if its DN, subCount and subEntropy reach thresholds. The threshold for each parameter is adjusted as per the node level, outcomes, and empirical experience. For example, the volume of domains pointing to "us-east-2.amazonaws.com" is larger than that of"s3.us-east-2.amazonaws.com", so the DN at level 4 should be smaller than at level 3. In addition, user-specific prefixes are highly diverse, whereas provider-specific labels are in a relatively small pool. As a result, subCount and subEntropy of the nodes representing the longest suffixes could be significantly larger than those of their children nodes, leading to a large deviation in value. In the end, we refer to the remaining tree as a *DST*, like the solid part in Figure 5. We extract all leaf nodes in *DST* as endpoint patterns and aggregate those with the same parents.

It should be noted that the operational experiences and concerns, such as parameter selection, can be found in Appendix B.

Step 3: Identifying services and checking service vulnerabilities. In our threat model (Section 3.2), attackers can only re-allocate the same endpoints as victims to perform domain takeovers if the service endpoint pool is small. Hence, highly randomized endpoint domains, such as those with high entropy hash values or algorithm-generated labels like DGA [65]), are not exploitable. Based on this requirement, the discoverer employs Stringlifier [18], a popular open-source ML model that identifies high-entropy or numeric strings, to automatically identify and filter out random domains from the extracted endpoint patterns. Using this method, we can significantly narrow down the candidate list, leaving only 995 endpoint patterns.

After that, we further determine the service types of the extracted endpoint patterns by directly accessing their webpages and digging through search engines. Then we inspect their domain connection and DOV policies, as mentioned in Section 3.1. We first review vendors' operational documentation and search for service setup tutorial videos to find the officially claimed domain validation policies. If no instructions are found, we will register two test accounts for each platform, connect one custom domain, and configure a service. We inspect whether account 1 (the attacker) can deploy the domain deployed by account 2 (the victim) that has been discontinued.

In our experiment, we systematically explore the current practices adopted by hosting services. Finally, we discover 65 vulnerable services and new vulnerabilities in domain connection policies. The holistic characterization about services can be found in Section 6.1.

Step 4: Maintaining a comprehensive database for vulnerable services. When a hosting service is discontinued, we observe that providers will reply to the client with distinctive responses. These responses could be used as fingerprints to determine service status and detect $D_{vulhost}$. During our research, we summarize three types of useful fingerprints:

- F1 DNS answers. Platforms may return customized DNS answers when a domain's service is cancelled. For example, platforms may resolve endpoint domains to a default alias (e.g., nx.aicdn.com) or a particular IP (e.g., 127.0.0.1). Alternatively, the endpoints become non-existent (i.e., NXDOMAINs) if a service is discontinued and caches have expired.
- F2 HTTP response headers. HTTP fingerprints are also essential for checking domain status. Hosting platforms use default HTTP errors (e.g., "404 Not Found") to signal that services are unavailable. To distinguish service discontinuation from other failures, we extract specialized HTTP headers from each service alongside status codes. For example, "<vendor-customized-header>: Please double-check that you are using the correct url. If so, make sure it matches your dashboard's custom domain..." only displays when a domain is not hosted on the service.
- F3 HTTP response bodies. Platforms could adopt default pages (e.g., an error page) with similar HTML structures or notification sentences to convey the service status. As with previous work [36], we can use typical contents (e.g., "This web app is stopped" and "NoSuchBucket") in web pages as fingerprints

In total, we keep a database of 110 DNS and HTTP fingerprints, which is by far the most comprehensive list. We will not present all fingerprints due to ethical concerns. These fingerprints, when combined with exploitable endpoint patterns, can be used as the basis for $D_{vulhost}$ detection.

4.4 Detecting Hosting-based Dangling Domains

Collecting subdomain names. To detect $D_{vulhost}$ in the wild, it is essential to ensure the coverage of collected subdomains and to keep track of newly appearing ones. However, prominent organizations (e.g., Amazon [57] and Microsoft [24, 49]) may create plenty of subdomains and

constantly change domains when updating businesses. In addition, given the hierarchical authorization structure, their domains may have many labels (e.g., `region.console.aws.amazon.com`). Based on these facts, it is difficult to exhaust subdomains using brute-force scans or zone transfer requests (AXFR), as earlier research did [48, 58]. Moreover, detection targets should not be biased towards domains with specific usages or scenarios (e.g., enabling SSL), so certain public sources (e.g., Censys, CT, and search engines) are insufficient for this need. In order to balance domain coverage and enumeration efficiency, HostingChecker collects subdomains from passive DNS traffic that contains frequently queried domains at the country level.

HostingChecker will preprocess the domains in PDNS by following steps. First, it removes strings that do not comply with domain format requirements [50], such as the ones that contain illegal characters, are not separated by dots, or do not end with eTLDs. Second, it removes any temporarily constructed domains, such as those used for experiments, using a metric called *Total Query Volume* (*TQV*). *TQV* is the number of queries toward a domain since it first occurred. We use *TQV* to reflect domain popularity. We notice *TQV* shows a long tail, with most domains receiving a modest number of queries and a handful receiving millions. Third, many subdomains containing random strings (e.g., `0vkcr4be.example.com`) have similar formats. They are generated in bulk using algorithms (e.g., DGA) and resolve to the same targets [25]. To identify such names, the system also employs Stringlifier as used in Section 4.3. It aggregates apex domains whose subdomains contain "`RANDOM_STRING`" labels and chooses one of them as the representative for subsequent tests. In addition, HostingChecker only needs to append the subdomains that are freshly observed in PDNS logs between two detections to the original list to keep track of new subdomains for given apexes.

Reconstructing DNS chains. In $D_{vulhost}$ detection, we consider checking DNS resolution chains necessary for determining whether a domain is hosted on a service. The reason is that detectors cannot retrieve web contents if target resources are unreachable (like F1 in Section 4.3). In this process, HostingChecker employs a passive method, i.e., reconstructing DNS chains using PDNS logs, to achieve high efficiency. It starts by extracting all DNS RRs from PDNS for a specific subdomain. If a `CNAME` record occurs, it recursively queries the `data` field in an RR until an `A`, `NS`, or `MX` record is found. As a result, one subdomain's DNS chains are represented as a directed resolution graph, as depicted in Figure 3. The reconstruction of DNS chains is finished locally on the database in parallel, without sending network queries, thus improving efficiency. In Appendix A, we present its pseudocode in Algorithm 2.

There are two key points in the reconstruction process. First, we must ensure the DNS records we extract from PDNS still exist on authoritative name servers. We employ *Daily Query Volume* (*DQV*), which represents the query volume of a DNS record for each day. It is a metric that measures whether a record can be actively resolved. Typically, we can predefine a minimum *DQV* to handle long-tail data in PDNS, such as records that are queried only once. Second, some DNS RRs (e.g., CNAMEs) may form a cyclic graph regarding domain dependency; however, they have different lifetimes. Hence, we need to divide the cyclic dependency graph into multiple directed acyclic graphs (DAGs) if the RRs have no overlap in lifetimes. Otherwise, we must break the iterative process when receiving a domain that has already been processed. Furthermore, we should terminate the iterative queries for `NS` and `MX` RRs since they cannot create alias names [37].

Inspecting domain status and recognizing $D_{vulhost}$. To confirm that a domain is hosted on public services, HostingChecker will check its DNS settings to verify whether its DNS chain matches any endpoint patterns in our database. It then requests the domains hosted on vulnerable services and compares their fingerprints to those we've already acquired. Eventually, domains whose fingerprints indicate discontinued services are classified as $D_{vulhost}$.

5 IMPLEMENTATION AND EVALUATION

5.1 System Development

We implement HostingChecker in Golang and use goroutines to achieve parallel execution. We deploy HostingChecker on a PDNS dataset managed by our industrial partner, a large security company. During the experiments, we can only access their internal dataset using an experimental token they provide at a limited rate, e.g., obtaining no more than 10k items per query for subdomains and 1k for DNS records. When HostingChecker is made available as a public service, it will run on the Spark framework and execute SQL queries on the raw PDNS data more efficiently. Moreover, HostingChecker can employ similar parameters (e.g., *DN*, *TQV*, and *DQV*) for different data platforms. Appendix B provides practical guidance on parameter settings and key considerations.

Passive DNS dataset. To build HostingChecker as a practical service, we use a large-scale PDNS dataset (like Farsight DNSDB [35] and Umbrella [9]). It is continuously collected from multiple ISPs' global public DNS resolvers for 114DNS, the largest DNS provider in China [6]. The raw data is saved on the internal servers of the organization and is only available to authorized employees. Researchers can access anonymized data after acquiring legal authority and agreeing to a privacy policy.

The PDNS consists of around 600 billion unique DNS queries every day from 70M clients for nearly 800M FQDNs. It covers 99.9% of popular domains in the Tranco Top 1M list. These FQDNs are distributed across an average of 13M SLDs per day and cover 99.9% of IANA TLDs. The DNS queries originate from telecom companies (e.g., China Telecom and Viettel Group), research institutions (e.g., MIT and NUS), and large providers (e.g., Alibaba and Google). Compared to the dataset used in previous work [9], our PDNS observes 4× more DNS requests per day than Umbrella and 6× more FQDNs than [29]. We recognize that the PDNS has regional biases because it is not collected from truly global vantage points. Nevertheless, these biases are inherent to all PDNS datasets. We believe it is sufficient to reflect DNS queries for the world's most popular domains.

To evaluate HostingChecker, we use PDNS logs spanning from Jan. 2021 to Jul. 2022. We extract 7 data fields, including query name, query type, answer data, timestamp of first-seen and last-seen (i.e., collecting time), client IP volume, and query volume aggregated by each FQDN. These fields are unrelated to a specific person's activities, and the dataset we process contains no sensitive information (e.g., client IPs). We follow the best practices to use shared data [22] under the supervision of authorized employees and senior security researchers.

5.2 Coverage

Subdomains. We collect 12,835,311 subdomains (before filtering by *TQV*) for Tranco top 1M domains and 1,977,645 for the top 1k from PDNS. In order to demonstrate domain coverage, we compare our results with those collected from other 34 data sources embedded by popular tools, i.e., `amass` [51] and `subfinder` [53]. However, it is widely acknowledged that enumerating subdomains of Top-1M apexes is impractical [48]. Thus, we sample 1k apexes from each of the top 1k, 10k, 100k, and 1M domain lists and use these 4k domains for comparison. Results show that our PDNS has a high coverage in both apex domains and subdomains (Figure 14). Details of data sources and domain coverage are presented in Appendix C. In addition, our subdomains are diverse in label numbers, including domains used in particular scenarios, such as DGA domains or misconfigured domains (Figure 15).

Vulnerable services. With the help of the service discoverer, our vulnerable service database (Table 2), which contains endpoint patterns and fingerprints, is by far the most thorough. Previous research, [48] and [58], analyzed domain takeover issues among 9 and 31 services, respectively (see Table 1). Aside from publications, the security community also maintains public lists [36, 43] of

vulnerable services reported by contributors. In comparison, we have maintained more exploitable services (with 34 newly discovered), which ensures that HostingChecker can behave better than other systems in threat coverage. We will provide more details about vulnerable services in Section 6.1.

5.3 Efficiency

To evaluate HostingChecker's efficiency, we deploy it on an 8-core Intel Xeon machine with 16GB RAM, running Ubuntu 18.04 LTS. We use 100 goroutines to detect Tranco Top-1M sites' subdomains. Results show that a complete detection takes 1 day and 24 minutes on average, and DNS reconstruction takes only 13.9 hours. It should be noted that the detection efficiency of HostingChecker can be optimized in real-world deployments. First, if HostingChecker runs regularly, administrators simply need to obtain fresh FQDNs from incremental data rather than traversing all historical PDNS traffic. Thus, the time required to collect subdomains can be shortened. Second, the system can execute directly on a high-performance platform and process DNS data locally rather than performing active network queries, so the time for building DNS chains is negligible. We believe HostingChecker is efficient enough to complete a daily inspection of all subdomains of the top 1M domains to timely detect potential threats.

We claim that HostingChecker outperforms existing methods in domain resolution efficiency. As a comparison experiment, we attempt to resolve 227,645 domains that are subdomains under the top 1k apexes obtained from our PDNS and Rapid7. We use a traditional DNS lookup tool (i.e., `dig`) and a state-of-the-art tool (i.e., `ZDNS` [41]) for active queries and specify the same parameters for each tool, including a 5-second timeout, 3 retries, and 5 levels of iteration depth, as well as default values for other parameters. HostingChecker takes a half-hour to reconstruct the DNS chains for these domains, according to the results. However, compared to HostingChecker, it takes almost 13× longer to use `dig` and 5× as long to use `ZDNS`. Though efficiency can be improved by increasing concurrency, active DNS resolution consumes more network resources and may raise ethical risks.

5.4 Detection Accuracy

Because there is no ground truth for vulnerable domains, we can only evaluate detection accuracy by manually verifying $D_{vulhost}$ samples and checking their fingerprints. Note that all items in the fingerprint database (introduced in Section 4.3) have been double-checked and proven exploitable by our analysts. Meanwhile, HostingChecker saves real-time fingerprints that are used to determine service status during detection. For evaluating detection accuracy, we re-probe the fingerprints of $D_{vulhost}$ samples and compare them to those captured in detection. If a $D_{vulhost}$'s fingerprint has changed, we investigate further by visiting its homepages and checking if the relevant service's policies have been updated.

The $D_{vulhost}$ samples are selected in two ways. First, we select the top 20 vendors based on service types, popularity, and domain connecting methods (as done in Section 6.1.4), and we check all 3,165 $D_{vulhost}$ related to these vendors. Second, we expand the tested domain list by randomly sampling 2,000 $D_{vulhost}$ hosted on less popular services.

At the evaluation time, we confirm 4,907 (95.0%) domains were vulnerable to domain takeover. We then check the DNS settings of these domains through PDNS logs, finding 46.4% of all confirmed $D_{vulhost}$ have removed dangling DNS records and eliminated the threats. It indicates that detection results only reflect the domain status at detection time. We further try to figure out the reasons for the remaining 5.0% false positives. We find they are mainly caused by the domain whitelist maintained by the platforms. In detail, providers may prohibit adding domains they control to their platforms; for example, we cannot add `balancer.wixdns.net` to Wix, despite the fact that such domains can trigger the same fingerprints. However, because each provider's whitelist is not public,

we need to gather whitelisted domains from future measurements and remove them from detection targets to reduce false positives.

5.5 Discussion

Limitations. First, our PDNS observes DNS queries toward resolvers located in China, which might be biased due to their geo-location. While we acknowledge this limitation, our evaluation shows that the dataset provides adequate coverage of domains (99.9% Tranco domains) and clients worldwide. We believe the results derived from the dataset are representative. Second, the detection system will unavoidably produce false negatives because not all services and domains can be observed from PDNS. However, in comparison to previous results, we have discovered more services and detected over eight times as many $D_{vulhost}$, which is sufficient to assist the community in mitigating threats. Third, HostingChecker is not a fully automated system. Manual inspection is required in checking whether vendors update service policies and confirming fingerprint correctness, which determines detection accuracy. We cannot precisely evaluate the update frequency of service policies and fingerprints. However, the service policies did not change during our half-year experiments, and we found that 31 reported services [36] remained vulnerable, implying that inspections do not need to be undertaken frequently. We put the automated fingerprint determination to future work and will collaborate with security companies to keep the fingerprints up to date.

Deployment scenarios and possible users. HostingChecker is highly scalable and adaptable to any large-scale PDNS dataset (e.g., Farsight DNSDB [35]) or (daily) active DNS measurement data (e.g., OpenINTEL [61] and RapidDNS [54]). We envision two user groups for deploying HostingChecker. First, large domain registrars and public DNS providers that host vast volumes of domains can gather DNS requests via their name servers. They can provide detection services for customers by deploying HostingChecker. Second, prominent organizations, particularly large Internet companies (e.g., Microsoft), may have numerous and complex subdomains. They can use HostingChecker to detect vulnerable domains in time to avoid potential risks.

6 LARGE-SCALE MEASUREMENT AND ANALYSIS

6.1 Systematic Analysis of Hosting Services

Utilizing the service discoverer, we analyze most of the deployed hosting services from a real-world traffic perspective. Additionally, we re-evaluate the popular services that have been reported as vulnerable in the security community [36, 43]. Below, we describe all vulnerable services and exploitable policies identified by HostingChecker.

6.1.1 Vulnerable services overview. As illustrated in Table 2, HostingChecker identifies 995 endpoint patterns and 165 hosting services, which belong to 88 public vendors. Out of the identified services, 65 (39.4%) are vulnerable and 34 are newly reported by us. These vulnerable services mainly cover cloud object storage (e.g., Alibaba OSS and Huawei OBS), CDN (e.g., Baidu and Cloudflare), and website builders (e.g., Wix). We find that website builders dominate the share of vulnerable services. They will pose a significant threat due to the fact that they are open to both large companies and individual users. In addition, other hosting services (e.g., email hosting and API management) are difficult to attack due to their sophisticated domain deployment policies. They either require unique validation records (e.g., DMARC [46]) or do not apply to individual users. We discover 4 vulnerable cases from these services, which is the lower bound result, and we leave other services for future investigation. In particular, service dependencies can make secure services exploitable again because hosting services can serve as backends for other production-line services [56]. Baidu Object Storage, for instance, is exploitable since it allows users to host custom

Table 2. Discovered vulnerable hosting services.

Categories	# Vendors		# Endpoint Patterns		# Services	
	All	Vulnerable (%*)	All	Vulnerable (%*)	All	Vulnerable (%*)
Cloud Storage	7	7 (100.0%)	130	118 (90.8%)	12	9 (75.0%)
CDN	25	7 (28.0%)	247	31 (12.6%)	44	8 (18.2%)
Website Builders	51	40 (78.4%)	156	105 (67.3%)	60	44 (73.3%)
Others	27	4 (14.8%)	462	4 (0.9%)	49	4 (8.2%)
Newly discovered	55	19 (34.5%)	920	183 (19.9%)	125	34 (27.2%)
All	88	52 (59.1%)	995	258 (25.9%)	165	65 (39.4%)

* % is the fraction of the *vulnerable* value to the *all* value. Take CDNs for example, 18.2% (8/44) CDN services are vulnerable, which belong to 28.0% (7/25) vendors and provide 12.6% (31/247) vulnerable patterns.

domains by deploying its CDN, but the CDN does not perform domain validation. Such service dependencies pose more threats, yet they are hard to detect in individual services.

6.1.2 Exploitable domain connecting methods. Public hosting providers employ diverse domain connection methods, but most (7/9) of them are exploitable for domain takeover. Based on endpoint types, we divide the current domain connection methods into three categories (Table 3):

- *Connecting with* `CNAME` *records.* When customers connect custom domains on a service, the provider allows them to point domains to the service endpoints by `CNAME` RRs. The endpoints comprise five types: M_1 are fixed domains, i.e., all custom domains are hosted at the same endpoints. M_2 allows users to define domain prefixes, such as in the case of Figure 2, even if the prefixes have been used by others before. M_3 are different from M_2 since customers can only define new prefixes that are never used, i.e., the vendor will keep historical account settings to prevent endpoint name conflicts. M_4 are selected from a candidate pool maintained by the provider. Domains belonging to M_5 contain random strings to avoid conflicts.
- *Changing nameservers.* Similar to `CNAMEs`, customers can also configure `NS` RRs to host their domains. This method contains two types: delegating domains to the fixed nameservers (M_6, similar to M_1), or the ones selected from an `NS` pool (M_7, similar to M_4). For instance, Cloudflare [10], randomly provides two nameservers per customer (e.g., `alice.ns.cloudflare.com` and `bob.ns.cloudflare.com`), and its pool contains thousands of candidates.
- *Connecting with A records.* Another method is directly pointing domains to the given IP addresses, such as fixed IPs (M_8) or randomly selected IPs from an address pool (M_9).

We find most of these methods exploitable, as listed in Table 3. First, fixed endpoints (i.e., M_1, M_6, and M_8) are easily exploited as all custom domains (including $D_{vulhost}$) are pointed to them. Second, customized endpoints (M_2) are also exploitable if providers do not check whether they have been used before. Attackers can directly exploit these two types of endpoints to manipulate the dangling domains. Besides, if providers select endpoints from a small pool (e.g., M_4, M_7, and M_9), attackers have the chance to allocate the same endpoints to victims by creating multiple requests. However, one-time CNAMEs (M_5 and M_3) are not vulnerable in theory, since they are generated randomly for each account or cannot be reused. In total, we find 105 services vulnerable due to the flawed connecting methods.

6.1.3 Flawed domain ownership validation. Although several providers implement domain ownership validation (DOV) policies to mitigate domain takeover attacks, their implementations could be flawed. In our experiments, we find that only 10 vendors have strict DOV policies. For usability

Table 3. Domain connecting methods for hosting D_{user}.

Method	Name	Method description	# Service	Exploitable
CNAME	M_1	Point D_{user} to *fixed* canonical domains	12	●
	M_2	Point D_{user} to *any* canonical domains customized by *any* users	70	●
	M_3	Point D_{user} to *new* canonical domains customized by *new* users	12	○
	M_4	Point D_{user} to the canonical domains allocated *from a candidate pool*	5	◐
	M_5	Point D_{user} to canonical domains containing newly generated *random labels*	47	○
NS	M_6	Point D_{user} to *fixed* nameservers	1	●
	M_7	Point D_{user} to the nameservers allocated *from a candidate pool*	5	◐
IP	M_8	Point D_{user} to *fixed* IPs	8	●
	M_9	Point D_{user} to the IPs allocated *from a candidate pool*	4	◐

- D_{user} denotes custom domains. And we do not summarize validation methods using special records (e.g., DMARC).
●: exploitable, ○: not exploitable, ◐: exploitable via endpoint collisions.

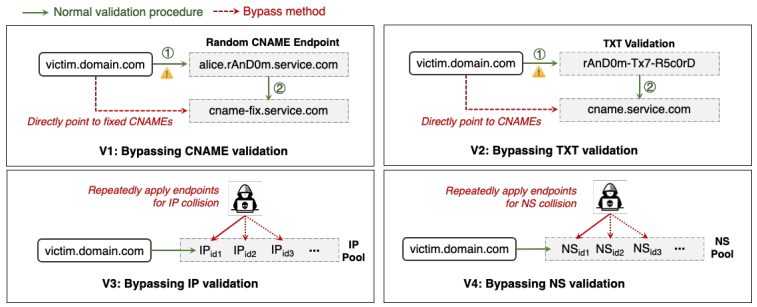


Fig. 6. Four threat models of bypassing flawed DOV. Each gray and dashed box represents a validation step.

reasons, most vendors prevent customers from connecting to the same endpoints by increasing the randomness of endpoint allocations. However, if the endpoint randomness is poor, attackers can achieve endpoint collisions by registering multiple accounts to bypass the validation. In detail, we uncover four new flaws (as depicted in Figure 6) that enable attackers to bypass DOV, affecting the top 20 service vendors listed in Table 4.

- *V1: Bypassing CNAME validation.* Among the top 20 vendors, 18 implement DOV by assigning a unique `CNAME` record to each customer, with 16 having implementation errors. As shown in Figure 6 (V1), the assigned `CNAME` (i.e., `alice.rAnD0m.service.com`) is unique, but the platform further resolves it to a fixed endpoint domain (i.e., `cname-fix.service.com`). The problem is that if someone directly resolves a custom domain to the fixed CNAME, the platform may also treat it as passing validation.
- *V2: Bypassing TXT validation.* Verifying a random `TXT` record is also common practice for DOV. For instance, Cloudflare requires a customer to create a `TXT` record with a random value for a domain, `cloudflare-verify.example.com`. However, it does not check whether a `TXT` record is correct and will still pass DOV if a custom domain directly points to a Cloudflare `CNAME` endpoint. Thus, attackers can exploit this vulnerability to take over arbitrary domains that are abandoned or not hosted on Cloudflare, as shown in Figure 6 (V2).
- *V3: Bypassing IP validation.* Non-reassignable IP endpoints ensure that clients without domain ownership cannot connect to the same IPs. However, some providers (e.g., WP Engine) select IPs from a small pool (M_8 and M_9). In this case, attackers can repeatedly apply IP endpoints

Table 4. Top 20 mainstream vendors and their exploitable services.

Category	Vendor	Service	Connecting method*	Vulnerable DOV				# $D_{vulhost}$
				V1	V2	V3	V4	
Cloud Strorage	Alibaba	OSS	M_2	✓	-	-	-	86
	Amazon	Elasticbeanstalk	M_2	✓	-	-	-	192
	Huawei	OBS	M_2	✓	-	-	-	178
	JD.COM	OBS	M_2	✓	-	-	-	51
CDN	Baidu	BOS, CDN, BCH	M_2	✓	-	-	-	1,309
	Cloudflare	CDN	M_2, M_7	✓	✓	-	-	543
	Fastly	CDN	M_2	✓	-	-	-	54
	Tencent	CDN	M_2	✓	-	-	-	119
Website Builder	Duda	Website Builder	M_1, M_8	✓	-	✓	-	10
	Jimdo	Website Builder	M_1, M_7, M_8	✓	-	✓	✓	5
	Medium	Blog	M_8	-	-	✓	-	3
	Netlify	Website Builder	M_1, M_2, M_7, M_8	✓	-	✓	✓	21
	Shopify	Website Builder	M_1, M_8	✓	-	✓	-	34
	Tilda	Website Builder	M_9	-	-	✓	-	4
	Tumblr	Blog	M_1, M_8	✓	-	✓	-	11
	Unbounce	Website Builder	M_5	✓	-	-	-	212
	Webflow	Website Builder	M_1, M_8	✓	-	✓	-	30
	Wix	Website Builder	M_4, M_7	✓	-	-	✓	26
	Wordpress	Website Builder	M_3, M_6, M_8	✗	-	✓	✓	27
	WP Engine	Website Builder	M_3, M_9	✗	-	✓	-	12

*: The exploitable domain connecting methods adopted by the services.
-: Indicate they have not deployed the validation methods involved in each vulnerability.

using different accounts until they control the same IPs as domain owners' and pass the validation, as the model in Figure 6 (V3).

- *V4: Bypassing NS validation.* Similar to V3, providers could provide a small NS pool (M_7), so attackers could apply the same NS endpoints. In addition, several platforms do not check if a customer has configured the NSes allocated for him/her as long as the configured ones are part of the NS pool. For instance, in Figure 6 (V4), customers should change the domain nameserver to NS_{id1} to start a service on Wix. However, if a domain's NS is set to NS_{id2}, it can still pass Wix's validation.

6.1.4 Current practices of hosting services. To investigate how mainstream vendors implement domain hosting policies, we select 20 vendors according to their popularity [8, 17], as illustrated in Table 4. They control around 70% market share [62, 63] and provide all service categories and domain connecting methods. For each service, we follow the checking steps mentioned in Section 4.3. After confirming the vulnerabilities, we will create a harmless proof of concept (as recommended in [1]) to disclose the issues to service providers and domain owners. Then, we consider ethical issues and delete all domains from the tested platforms. In conclusion, we confirm all of the top 20 hosting vendors are exploitable and find they can affect 2,927 custom domains. Each of the vendors involves at least one flawed domain validation method. Among these vendors, 12 are newly reported by us, and 3 were reported as secure before, but we find new flaws in their DOV implementations.

6.2 Threats on High-profile Domain Names

6.2.1 Measurement targets. To investigate the prevalence of $D_{vulhost}$, we use HostingChecker to detect Tranco Top-1M apex domains, together with 9,808 `.edu` and 7,198 `.gov` apex domains obtained from public lists [16, 32]. Table 5 illustrates our subdomain statistics and measurement results. In total, we collect 11,446,359 subdomains from PDNS for all apexes. We then conduct 101 rounds of measurements on these subdomains, spanning from Dec. 16, 2021 to Jul. 28, 2022. Below, we will discuss the main findings from the longitudinal measurements.

Table 5. Statistics on measurement results.

Source	# Apex	# Subdomain (FQDN)			# $D_{vulhost}$	
		A. All	B. Hosted (%)*[1]	C. Vul-hosted (%)*[2]	D. FQDN (%)*	Apex
Tranco	1,000	455,123	22,403 (4.9%)	11,440 (51.1%)	3,717 (32.5%)	34
	10,000	1,543,435	66,085 (4.3%)	28,490 (43.1%)	4,033 (14.2%)	110
	100,000	4,663,882	161,083 (3.5%)	65,277 (40.5%)	8,064 (12.4%)	418
	1,000,000	11,433,441	283,898 (2.5%)	114,002 (40.2%)	10,303 (9.0%)	2,058
.com	508,580	6,802,277	193,552 (2.8%)	96,131 (49.7%)	5,270 (5.5%)	1,406
.net	45,520	929,447	18,439 (2.0%)	11,626 (63.0%)	3,658 (31.5%)	56
.gov	7,198	24,713	2,218 (9.0%)	581 (26.2%)	5 (0.9%)	3
.edu	9,808	248,914	6,078 (2.4%)	3,054 (50.2%)	23 (0.8%)	16
Total	1,008,734	11,446,359	284,006 (2.5%)	114,063 (40.2%)	**10,351 (9.1%)**	**2,096**

[1] Hosted represents FQDNs hosted on public services.
[2] Vul-hosted represents FQDNs hosted on vulnerable public services.
* % is the fraction of the current value to the value in its adjacent left column. Take the first row for example, 4.9% (22,403/455,123, i.e., B/A) FQDNs are hosted on public services, 51.1% (11,440/22,403, i.e., C/B) hosted domains deploy vulnerable service, and 32.5% (3,717/11,440, D/C) vulnerable FQDNs are dangling.

6.2.2 Hosting-based dangling domains. A major finding is that the domain takeover threats caused by hosting services are still prevalent. According to the total numbers in Table 5, HostingChecker detects that 284,006 (2.5%) of all collected subdomains have ever been hosted at public services. We find 40.2% of the hosted domains use vulnerable services, and we term them "vul-hosted domains". After checking fingerprints, HostingChecker finally confirms that 10,351 (9.1%) of the vul-hosted ones have become $D_{vulhost}$ during our experiments. The $D_{vulhost}$ number is over eight times higher than prior results.

Vulnerable TLDs. We find FQDNs under generic TLDs account for the majority of $D_{vulhost}$, with 50.9% under `.com` and 35.3% under `.net` respectively. Moreover, we discover that vulnerable services are already deployed by a significant portion of FQDNs under critical TLDs, such as 26.2% of hosted `.gov` domains and 50.2% of hosted `.edu` domains. In particular, we detect 23 FQDNs under `.edu` and 10 under `.gov(.ccTLD)` were vulnerable during our experiments. By manually checking, we find most of them were hosted on Webflow, AWS Elastic Beanstalk, and Pantheon, which are online website builders. Two subdomains of `unc.edu` and `rice.edu` were continuously exploitable for over three months (the domain administrators have then solved the issues). This implies the need for more proper management of subdomains by the government and university administrators to prevent potential attacks.

Affected apex domains. Popular apex domains that prefer to host subdomains on third-party services are especially vulnerable to domain takeover attacks. The $D_{vulhost}$ we detected belong to 2,096 apex domains, of which 2,058 are ranked in top 1M and 34 even in top 1k. The most vulnerable subdomains are under `bcebos.com` (1,250 FQDNs), which belong to Baidu Object Storage. Such cloud storage services enable users to access stored resources via vendor domains whose prefixes can be customized.

By analyzing domain ranking, we find prominent organizations may create more subdomains that are challenging to manage. Table 5 summarizes the subdomain numbers in various domain ranking intervals, allowing us to approximate an average subdomain number for one apex. After estimating, we discover that an apex in the top 1k has an average of 455 (455,123 divided by 1,000) subdomains, while this number drops to 154 in the top 10k. In addition, according to column B, we calculate the fraction of domains hosted on public servers across all subdomains. It shows that higher ranked apexes have more hosted subdomains than lower ranked ones. Their massive subdomains and businesses may be a mojor cause, since they need reliable third-party platforms to

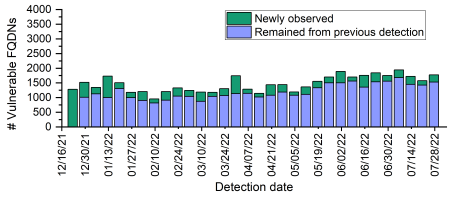


Fig. 7. Weekly cumulative $D_{vulhost}$ results.

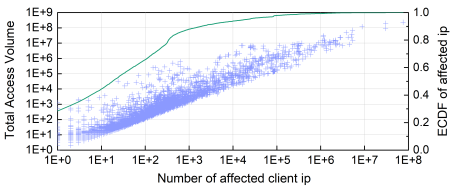


Fig. 8. Threat caused by discovered $D_{vulhost}$.

ensure security and stability. However, the services they select may have implementation flaws. Aaccording to the percentages in column C, for example, more than half of the services utilized by the top 1k domains are vulnerable, exposing their domains to threats (column D). In particular, in terms of domain ranking, the number distribution of $D_{vulhost}$ is aligned with that of hosted domains.

6.2.3 Indirect dangling domains and the expanded threats. Apart from customers' domains, vulnerable public services also endanger 18,253 domains that are not hosted on them. The reasons include: (i) domains depend on vulnerable customer domains; or (ii) their owners configure wildcard DNS RRs. For the first, if a domain A has a canonical name B that becomes a $D_{vulhost}$, attackers can indirectly control A after taking over B. We refer to domains like A as indirect dangling domains (indirect $D_{vulhost}$). In the measurements, we find 10 indirect $D_{vulhost}$ that point to 5 $D_{vulhost}$ via CNAME records. In order to assess the impact of indirect $D_{vulhost}$, we further reverse-parse domain names from PDNS that point to our discovered $D_{vulhost}$ and obtain an additional 18,057 affected FQDNs. Such domain dependency makes it difficult to prevent attacks because domain owners of indirect $D_{vulhost}$ are unaware of third-party services. As a supplement, we illustrate the threat scale of indirect $D_{vulhost}$ in Appendix D (Figure 16).

As for the second reason, we find there are unsafe configurations of wildcard domains. In order to use a hosting service, a customer may set wildcard domain records in DNS setups, such as pointing `*.example.com` to the service's CNAME endpoints. If the service uses a fixed endpoint (e.g., M_1 in Table 3) for validation, all subdomains in the wildcard domain will point to the same endpoint. In this case, attackers can claim arbitrary subdomains on the platform and manipulate them. After double-checking, we discover that at least 191 wildcard domains point to exploitable services.

6.2.4 Periodic detection findings. Figure 7 depicts the weekly cumulative results from our longitudinal measurements. Starting from the second week, we notice an average of 270 new $D_{vulhost}$ per week. There was an increase in the new $D_{vulhost}$ on January 13, 2022, as we added Baidu's services to the monitoring list. Meanwhile, the reason for the spike on March 31 is that some subdomains hosted on Wix expired, and administrators then renewed them that week. By conducting regular measurements, administrators can notify the owners of vulnerable names in time to foreclose on potential threats. Further, we can gain more insights into domain takeover issues from a longitudinal perspective.

Lifecycle of a hosting-based dangling domain. Routine detection can reflect how domains' status changes over time. Figure 9 depicts the lifecycle of a $D_{vulhost}$, where three time points are highlighted. T_h is when a custom domain is hosted on a vulnerable service. Then, if the service is discontinued, the domain can become a $D_{vulhost}$ at T_s. After that, its threat status may end at T_e for reasons of service renewals or DNS record updates. Based on these states, we define two durations to analyze the (dis)appearance of a $D_{vulhost}$.

Pre-vulnerable duration (δ_{pre}). To begin, we ask when a domain will become dangling after being hosted on a service. To answer the question, we propose a metric called pre-vulnerable duration,

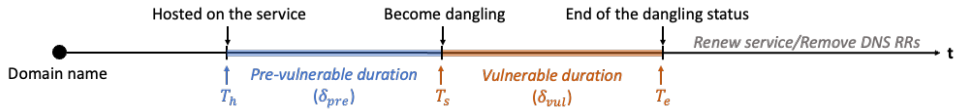


Fig. 9. Lifecycle of a $D_{vulhost}$.

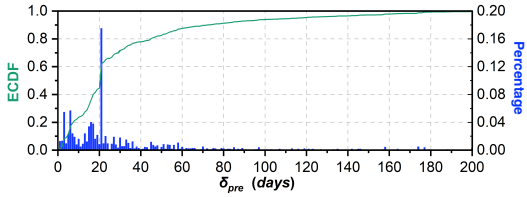


Fig. 10. Distribution of pre-vulnerable duration.

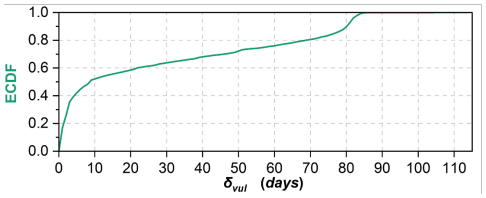


Fig. 11. Distribution of vulnerable duration.

δ_{pre}=T_s-T_h. In our experiments, we see the full δ_{pre} of 1,793 $D_{vulhost}$. As per Figure 10, we find that nearly 50% of domains have a δ_{pre} ranging from 10 to 30 days. A possible reason is that many services offer a short-term (e.g., 2 weeks) free trial for newcomers [7, 14]. A $D_{vulhost}$ arises if a customer starts a temporary trial on a public service but does not proceed with the subscription and forgets to purge DNS settings.

Vulnerable duration (δ_{vul}). Second, we can track each $D_{vulhost}$'s vulnerable duration, δ_{vul}=T_e-T_s, which represents the threat's time period. We divide $D_{vulhost}$ into four types depending on their δ_{vul}: We have observed both T_s and T_e of D_1 (77.0%), and only T_s of D_2 (22.8%); $D3$ (0.2%) is found to be vulnerable since our first scan, and ends its vulnerable state in the middle of measurements; $D4$ represents a domain that is continuously dangling; however, we discovered that no $D_{vulhost}$ belongs to this type. By analyzing the δ_{vul} of all detected $D_{vulhost}$, we find the threat can persist for a long time. We mainly use $D1$ for this analysis since we have recorded their entire threat duration. Figure 11 presents the empirical cumulative distribution of $D1$'s δ_{vul}. It shows only 17.0% $D_{vulhost}$ immediately disappeared within one day after being found dangling. However, nearly 60% have been vulnerable for a long time, i.e., $\geq$ 5 days, and 36.3% have even lasted longer than a month. This implies domain owners are rarely aware of such dangling domains and will probably not take proactive actions to fix them, leaving a wide time window for attackers.

Threat implication of $D_{vulhost}$. As further analysis, we attempt to quantify the real-world threat posed by $D_{vulhost}$, which remained vulnerable for several days. We use PDNS to estimate (i) the number of client IPs and (ii) the number of times the dangling domains were queried once they became vulnerable. We find 45.5% of all $D_{vulhost}$ still receive traffic from over 70 million client IPs, though their services were discontinued. It means that a large volume of clients are exposed to domain takeover threats. Figure 8 depicts the number of affected client IPs and the overall access volume during each δ per domain. It reveals that 43 $D_{vulhost}$ affect millions or even tens of millions of clients, and 364 have been accessed at least a million times. This significant impact is evidence that timely and comprehensive detection of $D_{vulhost}$ is imperative.

7 DISCUSSION

Best practices for hosting services. As per our analysis, we conclude that the following strategies are recommended practices that can prevent the threat. First, providers must perform strict domain ownership validation using standard procedures [40], rather than simply checking domain connections that use reassignable endpoints. For instance, providers can set one-time tokens or random values in the `TXT` records [42]. Second, providers must avoid potential endpoint collisions

for various accounts. When users deploy custom domains, providers should provide non-collidable endpoints from both the account and domain perspectives. They could choose at random from a large candidate pool. Third, providers should record all historical user-defined labels or hosted domains together with the relevant accounts, in case attackers can easily customize the same endpoint names as victims. Fourth, it is recommended to send security alerts to customers, including but not limited to service expiration and the use of wildcard DNS records.

Although the solutions are straightforward, our findings indicate that providers' domain hosting strategies vary. We contact hosting providers in order to understand the causes of poor implementations. First, they claim that implementing these validation policies in the production environment is complex and may affect the usability of services. To balance security and usability, they employ various simplified solutions (e.g., using a small endpoint pool), which we have shown can be bypassed. However, our research demonstrates the threat's impact remains pervasive and requires further mitigation. Second, some providers believe that solving this problem relies on deleting dangling records by customers. However, from an operational standpoint, it is unfeasible for the majority of users to be security-aware and manage their records well. Hence, we believe that public providers should take responsibility for minimizing security threats in order to improve ecosystem security. Along with validation during the domain connection phase, platforms should also perform critical checks in the exit phase (e.g., service destruction) and inform customers if configured resources are not purged.

Ethical considerations. We take full consideration of ethics based on the Menlo Report [27]. First, we use controlled accounts and domains to test public services. We follow responsible disclosure guidelines [1] when disclosing issues to vendors and domain owners. Second, similar to earlier papers [38], we use a PDNS dataset to explore real-world implications. With respect to people, we do not use personally sensitive information like client IPs, nor do we associate DNS traffic with specific individuals. We never investigate information exposed in domain strings. More data collection and processing details are discussed in Section 5. Third, for large-scale measurements, we deploy HostingChecker on infrastructure managed by a security company and limit the scanning rate to minimize the impact on real-world servers. Furthermore, since we do not perform active DNS queries, there is no influence on the DNS ecosystem.

Responsible disclosure. We have reported newly discovered vulnerabilities to affected vendors. So far, 10 service vendors have confirmed the issues. Amazon, Huawei, Duda, and Jimdo have acknowledged the issues and are working to find solutions. Baidu confirmed the vulnerability and awarded us a bounty. Baidu, Tencent, and Unbounce have added and SXL plans to add random `TXT` validation for DOV to mitigate the issue. Cloudflare and Netlify acknowledged the issue and decided there were no direct security implications for their vendors through the HackerOne platform (cite: hackeroneplatform). The other vendors asked for vulnerability details, and we are still waiting for further responses. In addition, since it is difficult to obtain the contacts of domain owners due to the anonymity of WHOIS data, we only report issues to the administrators of several popular domains. In the end, 16 domain owners have removed dangling DNS records after confirming our reports, including the examples present in the paper and subdomains under `aids.gov`, `stanford.edu`, `nobelprize.org`, and `asus.com`.

Detection service for research. To help detect dangling domains, we provide the system as a service for research purposes. Researchers that are interested in the service can gain access by visiting https://sites.google.com/view/hostingchecker. Considering ethics and avoiding abuse, we will only provide detailed detection reports to domain owners. After strict identity verification, domain owners can check whether their subdomains are hosted on flawed third-party services and thus can be taken over.

8 CONCLUSION

This paper introduces a highly efficient detection framework, HostingChecker, to comprehensively measure the prevalence of hosting-based dangling domains. Leveraging large-scale passive DNS traffic, HostingChecker can spotlight hosting services, harvest exhaustive subdomains, and efficiently reconstruct domain dependency chains. Through significant efforts to analyze hosting services, we identify a number of vulnerable services and feasible fingerprints for $D_{vulhost}$ detection. Periodic experiments and longitudinal measurements are conducted in order to monitor dangling domains in the wild. By leveraging HostingChecker, we find more vulnerable services and detect over 10k $D_{vulhost}$ that can be taken over. Our findings highlight the need for more research efforts on better service practices and remediation of domain takeover risks, especially for public hosting services.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and our shepherd, Gareth Tyson, for their valuable comments to improve the paper, and our industry partners for their support. This research is supported in part by the National Natural Science Foundation of China (62102218, U1836213, U19B2034, and 62272265). Yiming Zhang is partially supported by the Shuimu Tsinghua Scholar Program.

REFERENCES

[1] 2018. HackerOne: A Guide To Subdomain Takeovers. https://www.hackerone.com/application-security/guide-subdomain-takeovers.
[2] 2020. 670+ Subdomains of Microsoft are Vulnerable to Takeover. https://vullnerability.com/blog/microsoft-subdomain-account-takeover.
[3] 2020. American News Site's Subdomains Left Open for Takeover. https://www.wizcase.com/blog/cbslocal-vulnerabilty-research/.
[4] 2021. Alibaba Cloud. https://www.alibabacloud.com/.
[5] 2021. Amazon Web Services. https://aws.amazon.com/.
[6] 2022. *114 DNS.* https://www.114dns.com/
[7] 2022. 14 Day Trial Period Policy for Premium Plans. https://support.wix.com/en/article/14-day-trial-period-policy-for-premium-plans.
[8] 2022. *CDN Usage Distribution in the Top 1 Million Sites.* Retrieved May 20, 2022 from https://trends.builtwith.com/cdn
[9] 2022. Cisco Umbrella Passive DNS. https://docs.umbrella.com/investigate/docs/passive-dns.
[10] 2022. Cloudflare. https://www.cloudflare.com/.
[11] 2022. GoDaddy. https://www.godaddy.com/.
[12] 2022. Internet hosting service. https://en.wikipedia.org/wiki/Internet_hosting_service.
[13] 2022. *Public Suffix List.* https://publicsuffix.org/
[14] 2022. Shopify. https://shopify.com/.
[15] 2022. *Tranco List.* Retrieved Dec. 14, 2021 from https://tranco-list.eu/
[16] 2022. University Domains and Names Data List. https://github.com/Hipo/university-domains-list
[17] 2022. *Web Hosting Usage Distribution in the Top 1 Million Sites.* Retrieved May 20, 2022 from https://trends.builtwith.com/hosting
[18] Adobe. 2022. Stringlifier. https://github.com/adobe/stringlifier.
[19] Gautam Akiwate, Mattijs Jonker, Raffaele Sommese, Ian D. Foster, Geoffrey M. Voelker, Stefan Savage, and KC Claffy. 2020. Unresolved Issues: Prevalence, Persistence, and Perils of Lame Delegations. In *IMC '20: ACM Internet Measurement Conference, Virtual Event, USA, October 27-29, 2020.* ACM, 281–294. https://doi.org/10.1145/3419394.3423623
[20] Gautam Akiwate, Stefan Savage, Geoffrey M. Voelker, and KC Claffy. 2021. Risky BIZness: Risks Derived from Registrar Name Management. In *Proceedings of the 21st ACM Internet Measurement Conference* (Virtual Event) *(IMC '21).* Association for Computing Machinery, New York, NY, USA, 673–686. https://doi.org/10.1145/3487552.3487816
[21] Alibaba Cloud. 2022. Regions and endpoints. https://www.alibabacloud.com/help/en/doc-detail/31837.htm.
[22] Mark Allman and Vern Paxson. 2007. Issues and etiquette concerning use of shared measurement data. In *Proceedings of the 7th ACM SIGCOMM Internet Measurement Conference, IMC 2007, San Diego, California, USA, October 24-26, 2007,* Constantine Dovrolis and Matthew Roughan (Eds.). ACM, 135–140. https://doi.org/10.1145/1298306.1298327

[23] Eihal Alowaisheq, Siyuan Tang, Zhihao Wang, Fatemah Alharbi, Xiaojing Liao, and XiaoFeng Wang. 2020. Zombie Awakening: Stealthy Hijacking of Active Domains through DNS Hosting Referral. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM, 1307–1322. https://doi.org/10.1145/3372297.3417864

[24] Amazon. 2022. *AWS service endpoints.* Retrieved July 22, 2022 from https://docs.aws.amazon.com/general/latest/gr/rande.html

[25] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. 2017. Understanding the Mirai Botnet. In *Proceedings of the 26th USENIX Security Symposium (USENIX Security '17).*

[26] AWS Elastic Beanstalk. 2022. Your Elastic Beanstalk environment's Domain name. https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/customdomains.html.

[27] M. Bailey, E. Kenneally, D. Maughan, and D. Dittrich. 2012. The Menlo Report. *IEEE Security & Privacy* 10, 02 (mar 2012), 71–75. https://doi.org/10.1109/MSP.2012.52

[28] David Bisson. 2017. Hacker defaces Donald Trump fundraising site via subdomain takeover attack. (2017).

[29] Kevin Borgolte, Tobias Fiebig, Shuang Hao, Christopher Kruegel, and Giovanni Vigna. 2018. Cloud Strife: Mitigating the Security Risks of Domain-Validated Certificates. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018.* The Internet Society. http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018_06A-4_Borgolte_paper.pdf

[30] Guy Bruneau. 2010. DNS Sinkhole. *SANS White Paper* (Aug 2010). https://www.sans.org/white-papers/33523/

[31] Yizheng Chen, Manos Antonakakis, Roberto Perdisci, Yacin Nadji, David Dagon, and Wenke Lee. 2014. DNS Noise: Measuring the Pervasiveness of Disposable Domains in Modern DNS Traffic. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014.* IEEE Computer Society, 598–609. https://doi.org/10.1109/DSN.2014.61

[32] CISA. 2022. Official .gov Domain List. https://github.com/cisagov/dotgov-data

[33] Dave Crocker, Tony Hansen, and Murray Kucherawy. 2011. RFC 6376: DomainKeys Identified Mail (DKIM) Signatures (Internet Standard). https://tools.ietf.org/html/rfc6376. (2011).

[34] Detectify. 2022. Subdomain takeovers are on the rise and are getting harder to monitor. https://blog.detectify.com/2022/03/22/subdomain-takeover-on-the-rise-detectify-research/. (2022).

[35] DNSDB 2022. *Passive DNS historical internet database: Farsight DNSDB.* Retrieved July 18, 2022 from https://www.farsightsecurity.com/solutions/dnsdb/

[36] EdOverflow. 2022. Can I take over XYZ. https://github.com/EdOverflow/can-i-take-over-xyz. Last accessed: May. 12, 2022.

[37] Robert Elz and Randy Bush. 1997. Clarifications to the DNS Specification. RFC 2181. https://doi.org/10.17487/RFC2181

[38] Farsight Security. 2022. Research using Farsight DNSDB. Retrieved Apr 20, 2022 from https://www.farsightsecurity.com/research/

[39] Pawel Foremski, Oliver Gasser, and Giovane C. M. Moura. 2019. DNS Observatory: The Big Picture of the DNS. In *Proceedings of the Internet Measurement Conference, IMC 2019, Amsterdam, The Netherlands, October 21-23, 2019.* ACM, 87–100. https://doi.org/10.1145/3355369.3355566

[40] CA/Browser Forum. 2022. Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates, Version 1.8.4. https://cabforum.org/baseline-requirements-documents/.

[41] GitHub. 2020. *ZDNS: Fast CLI DNS Lookup Tool.* Retrieved July 14, 2022 from https://github.com/zmap/zdns

[42] Github Pages. 2022. Verifying your custom domain for GitHub Pages. https://docs.github.com/en/pages/configuring-a-custom-domain-for-your-github-pages-site/verifying-your-custom-domain-for-github-pages.

[43] HackerOne 2022. *Hacktivity: subdomain takeover.* Retrieved May. 12, 2022 from https://hackerone.com/hacktivity?querystring=subdomain%20takeover

[44] Jian Jiang, Jinjin Liang, Kang Li, Jun Li, Hai-Xin Duan, and Jianping Wu. 2012. Ghost Domain Names: Revoked Yet Still Resolvable. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012.* The Internet Society. https://www.ndss-symposium.org/ndss2012/ghost-domain-names-revoked-yet-still-resolvable

[45] Andrew J Kalafut, Minaxi Gupta, Christopher A Cole, Lei Chen, and Nathan E Myers. 2010. An empirical study of orphan DNS servers in the internet. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement.* 308–314.

[46] Murray Kucherawy and Elizabeth Zwicky. 2015. RFC 7489: Domain-based Message Authentication, Reporting, and Conformance (DMARC) (Informational). https://tools.ietf.org/html/rfc7489. (2015).

[47] Xiang Li, Baojun Liu, Xuesong Bai, Mingming Zhang, Qifan Zhang, Zhou Li, Haixin Duan, and Qi Li. 2023. Ghost Domain Reloaded: Vulnerable Links in Domain Name Delegation and Revocation. In *Proceedings of the 30th Annual*

Network and Distributed System Security Symposium (NDSS '23). https://doi.org/10.14722/ndss.2023.23005
[48] Daiping Liu, Shuai Hao, and Haining Wang. 2016. All Your DNS Records Point to Us: Understanding the Security Threats of Dangling DNS Records. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM, 1414–1425. https://doi.org/10.1145/2976749.2978387
[49] Microsoft. 2022. *Reference list of Azure domains (not comprehensive)*. Retrieved July 22, 2022 from https://docs.microsoft.com/en-us/azure/security/fundamentals/azure-domains
[50] Mockapetris, Paul V. 1987. RFC 1034: Domain Names - Concepts And Facilities (Standard). *RFC* (1987). https://datatracker.ietf.org/doc/html/rfc1034
[51] OWASP. 2022. amass. https://owasp.org/www-project-amass/.
[52] PCWorld. 2015. Lenovo, Google websites hijacked by DNS attacks. https://www.pcworld.com/article/432090/like-google-in-vietnam-lenovo-tripped-up-by-a-dns-attack.html.
[53] projectdiscovery. 2022. subfinder. https://github.com/projectdiscovery/subfinder.
[54] RapidDNS. 2022. Rapid DNS Information Collection. https://rapiddns.io/.
[55] Shivan Kaul Sahib, Shumon Huque, and Paul Wouters. 2022. *Survey of Domain Verification Techniques using DNS*. Internet-Draft draft-ietf-dnsop-domain-verification-techniques-00. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-sahib-domain-verification-techniques/03/ Work in Progress.
[56] Said Jawad Saidi, Srdjan Matic, Oliver Gasser, Georgios Smaragdakis, and Anja Feldmann. 2022. Deep Dive into the IoT Backend Ecosystem. *CoRR* abs/2209.09603 (2022). https://doi.org/10.48550/arXiv.2209.09603 arXiv:2209.09603
[57] SecurityTrails. 2022. *amazon.com subdomains*. Retrieved July 22, 2022 from https://securitytrails.com/list/apex_domain/amazon.com
[58] Marco Squarcina, Mauro Tempesta, Lorenzo Veronese, Stefano Calzavara, and Matteo Maffei. 2021. Can I Take Your Subdomain? Exploring Same-Site Attacks in the Modern Web. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2917–2934. https://www.usenix.org/conference/usenixsecurity21/presentation/squarcina
[59] Shir Tamari and Ami Luttwak. 2021. A New Class of DNS Vulnerabilities Affecting Many DNS-as-Service Platforms. https://i.blackhat.com/USA21/Wednesday-Handouts/us-21-A-New-Class-Of-DNS-Vulnerabilities-Affecting-Many-DNS-As-Service-Platforms.pdf.
[60] The Wall Street Journal. 2015. Cybercriminals Are Misappropriating Businesses' Web Addresses. https://www.wsj.com/articles/now-cybercriminals-are-misappropriating-businesses-web-addresses-1426120840.
[61] Roland van Rijswijk-Deij, Mattijs Jonker, Anna Sperotto, and Aiko Pras. 2016. A High-Performance, Scalable Infrastructure for Large-Scale Active DNS Measurements. *IEEE J. Sel. Areas Commun.* 34, 6 (2016), 1877–1888. https://doi.org/10.1109/JSAC.2016.2558918
[62] W3Techs. 2022. Usage statistics of content management systems. https://w3techs.com/technologies/overview/content_management. Last accessed: Oct. 6, 2022.
[63] W3Techs. 2022. Usage statistics of web hosting providers. https://w3techs.com/technologies/overview/web_hosting. Last accessed: Oct. 6, 2022.
[64] Wikipedia. 2022. HTTP 404. https://en.wikipedia.org/wiki/HTTP_404#Soft_404_errors.
[65] Wikipedia contributors. 2022. Domain generation algorithm — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Domain_generation_algorithm&oldid=1068669787 [Online; accessed 13-October-2022].
[66] Wikipedia contributors. 2022. Entropy (information theory) — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Entropy_(information_theory)&oldid=1115105228 [Online; accessed 16-October-2022].

A ALGORITHMS ADOPTED BY HOSTINGCHECKER

We present the algorithms of HostingChecker below. Algorithm 1 is the pseudocode for finding service endpoint candidates in Section 4.3 (Step 1). Algorithm 2 is the pseudocode for reconstructing domain dependency chains from passive DNS traffic, used in Section 4.4.

B CONSIDERATIONS OF SYSTEM PARAMETER SELECTION

DNS chain reconstruction. In this module, HostingChecker needs to set thresholds for Total Query Volume (*TQV*) of FQDNs and Daily Query Volume (*DQV*) of DNS records, and the selection depends only on the popularity and activity of the domain names. For example, if the minimum *TQV* is set to 1, all FQDNs appearing in the historical traffic should be extracted. However, the lookup volume of some FQDNs shows a long tail, such as temporarily constructed domains for scanning purposes, unintentionally accessed domains with typos, or disposable domains that are

Algorithm 1 Search for service endpoint candidates

Input: *PassiveRRs*, *Min*dqv, *MinDN*
Output: *Endpoints*

```
Endpoints ← ∅;
DomainTuples ← ∅;
for each RR ∈ PassiveRRs do
    qname, rtype, rdata, dqv ← UNPACK(RR)
    if rtype ∈ [CNAME, NS, MX] and ISLEGAL(rdata) and dqv ≥ Mindqv then
        qapex, rapex ← GETAPEX(qname, rdata);
        DomainTuples ← [qname, qapex, rdata, rapex]
        if qapex is new for rapex then rapex.dn += 1
        end if
    end if
end for
for each [qname, qapex, rdata, rapex] ∈ DomainTuples do
    if rapex.dn ≥ MinDN then Endpoints ← rdata
    end if
end for
```

Algorithm 2 Activity-based DNS Chain Reconstruction

Input: fqdn, depth, *MaxRecursive*, *Min*dqv
Output: *DnsChain*

```
procedure GETCHAINS(fqdn, depth)
    if depth > MaxRecursive then
        return ∅;
    end if
    chain.Domain ← fqdn;
    chain.Chains ← ∅;
    RRSet ← GET_PDNS_RRSET(fqdn);
    for each RR ∈ RRSet do
        rtype, rdata, dqv ← UNPACK(RR);
        if ISNOTCHECKED(rdata) and ISLEGAL(rdata) and dqv ≥ Mindqv then
            if rtype == CNAME then
                currChain ← GETCHAINS(rdata, depth + 1);
            else if rtype ∈ [NS, MX, A] then
                currChain.Domain ← rdata;
                currChain.Chains ← ∅
            end if
        end if
        Append currChain to chain.Chains;
    end for
    return chain;
end procedure
DnsChain ← GETCHAINS(fqdn, 1)
```

automatically created on demand. According to the existing work [31], the FQDNs that receive less than 10 lookups overall can be assumed to be long-tail data. Besides, the number of subdomains we can obtain tends to be stable when the minimum *TQV* is not less than 10. Therefore, we set *TQV* to 10 in our measurements, i.e., each domain name should have been visited at least 10 times in total. It's worth noting that this is only the lower bound for subdomain filtering, which ensures that we have obtained valid subdomains that appeared consistently in the historical traffic.

In addition, we only select data from the latest day of each detection to reduce the latency of detection results. Thus, we creatively use *DQV* to filter DNS records to reconstruct resolution

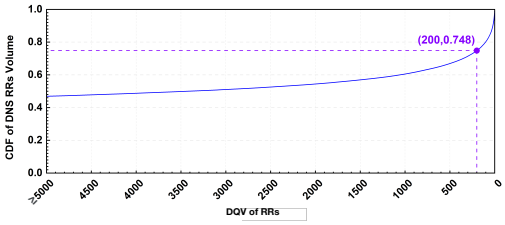


Fig. 12. ECDF of DNS RRs volume with DQVs.

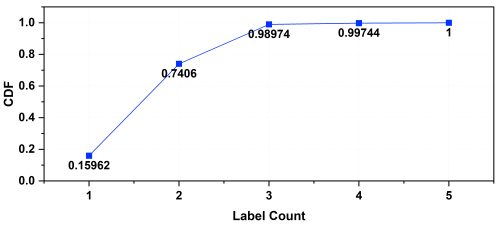


Fig. 13. Label count of domains in the PSL.

chains. Here, *DQV* is set for filtering out infrequent DNS records, which is similar to *TQV* for subdomains. We empirically set the minimum *DQV* of DNS RRs to 200 and assume the RRs being visited less than 200 times as the long tail. This is because more than 75% RRs have a DQV of over 200 (see Figure 12).

Service discovery. For obtaining potential service endpoints, HostingChecker also employs a parameter, *DQV*, which is the same as in the DNS Chain Reconstruction module. The difference is that the service discoverer uses it to filter only CNAME RRs. Besides, as introduced in Section 4.3, domain names of service endpoints often have common domain suffixes controlled by hosting providers. We heuristically selected the CNAMEs that are depended on by over 1,000 domains (i.e., *DN* is larger than 1k), since service providers with more than 1,000 customers may have a greater impact on the ecosystem.

Next, for extracting common patterns of obtained endpoint names, a few attributes (shown in Figure 5) are used to construct and prune the domain suffix tree. First, it appears from the public suffix list (PSL) that the label levels of public suffixes are generally no more than 5, as depicted in Figure 13, and over 98% suffixes have ≤ 3 labels. Thus, we typically merge all nodes whose suffix level is ≥ 5. Second, the service discoverer will prune the domain name tree from the bottom up by the following attributes: (1) DN: It represents the dependency number of the current level of suffix. We assume that if the DN of a particular level of the suffix is large, then the suffix is more likely to be part of the common pattern. In contrast, if the DN suddenly becomes smaller by a certain level, then it is more likely to be an endpoint name assigned to a specific user. (2) subCount: In practical terms, service endpoint patterns typically contain over 2 labels, i.e., the shortest suffixes are SLDs, and thus we leave all nodes at level ≤ 2. Besides, according to the published endpoint names [21, 26], we set different limitations for subCount in different levels. For example, the region labels' count (mostly in the third or fourth label from the suffixes) is set to no more than 50, the service types may not be more than 20, but the user-defined prefixes can be more than a thousand. (3) subEntropy: As there are usually a few endpoint patterns for a service, the subEntropy of nodes in the suffix can be within a limited range. However, the prefixes of service endpoint names (e.g., 'alice-prefix' in Figure 5) are user-specified, so the subEntropy of the last level of suffixes can increase abruptly compared to the previous levels. To this end, we will merge the sub-nodes of $Node_{level_{i+1}}$ if the subEntropy of $Node_{level_{i+1}}$ is larger than that of $Node_{level_i}$.

C DOMAIN COVERAGE FOR 34 DATA SOURCES

We use amass and subfinder to perform a comparison experiment for subdomain collection. We employ 27 and 7 data sources ranging from different technique methods that allow collecting domains for free, and the data sources are listed in Table 6. We depict the distribution of covered apex domains of our samples (4k in total) and the obtained subdomains from different data sources in Figure 14. It proves that our dataset exhibits high coverage from the perspective of apex domains

Table 6. Other data sources for subdomain collection.

Tool	Techniques	Data Sources
amass	Web Archives	ArchiveIt, Arquivo, CommonCrawl, UKWebArchive, Wayback
	Scraping	Ask, Baidu, Bing, DNSDumpster, DuckDuckGo, Gists, HackerOne HyperStat, PKey, RapidDNS, Searchcode, Searx, SiteDossier, Yahoo
	Certificates	CertSpotter, Digitorus
	APIs	FullHunt, Maltiverse, Mnemonic, SonarSearch, Sublist3rAPI, URLScan
subfinder	WHOIS	AlienVault
	Certificates	Crtsh
	APIs	HackerTarget, ThreatCrowd, ThreatMiner, AnubisDB
	Scraping	Riddler

and subdomains. Results show that only Rapid7 (i.e., Project Sonar) can obtain more subdomains than ours, which however only covers 68.6% of tested apex domains. Domains in SSL certificates that are obtained from `crt.sh` can cover a high percentage (85.3%) of apex domains. However, it seems not all their subdomains have enabled SSL settings, i.e., the collected subdomains are fewer than ours. In addition, Figure 15 illustrates that our subdomain results are diverse in domain label numbers, which can include the domains used in special scenarios, such as back-end APIs, DGA domains, or misconfigured domains.

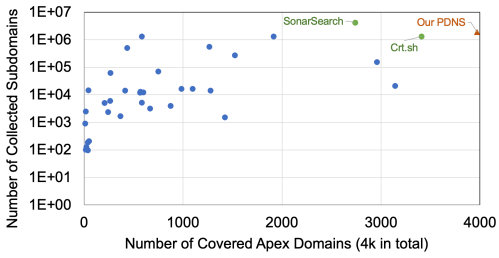


Fig. 14. Volume of obtained subdomain names and covered base domains using different data sources.

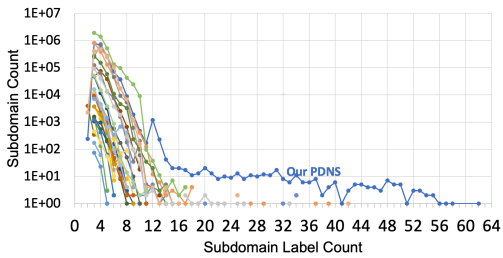


Fig. 15. Label count of collected subdomains using different data sources.

D DOMAIN AND SERVICE DEPENDENCY

We find that some providers may expose a large scale of user domains to security threats, and the domain names that indirectly depend on these services could also be vulnerable. We present examples showing the threat scale caused by some vulnerable domains in Figure 16.

Received 11 August 2022; revised 20 October 2022; accepted 9 December 2022

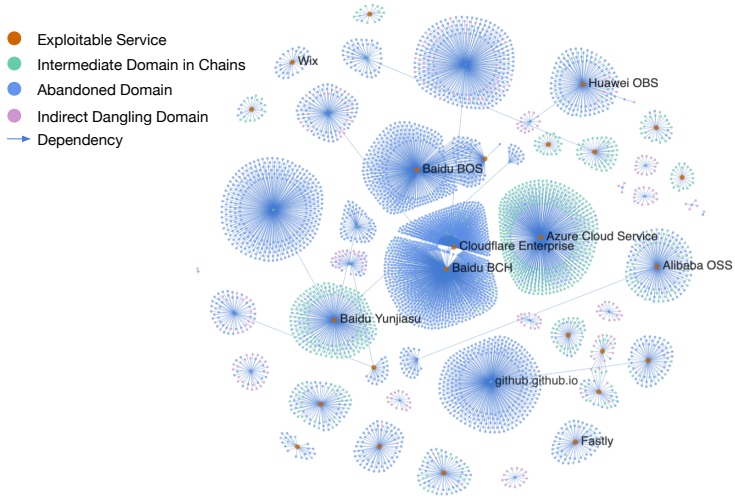


Fig. 16. Domain dependencies can cause a cascading effect of domain takeover threats.