Can I Take Your Subdomain? Exploring Same-Site Attacks in the Modern Web

Marco Squarcina, Mauro Tempesta, and Lorenzo Veronese, *TU Wien;*
Stefano Calzavara, *Università Ca' Foscari Venezia & OWASP;* Matteo Maffei, *TU Wien*

https://www.usenix.org/conference/usenixsecurity21/presentation/squarcina

This paper is included in the Proceedings of the
30th USENIX Security Symposium.

August 11–13, 2021

978-1-939133-24-3

Open access to the Proceedings of the
30th USENIX Security Symposium
is sponsored by USENIX.

Can I Take Your Subdomain? Exploring Same-Site Attacks in the Modern Web

Marco Squarcina[1] Mauro Tempesta[1] Lorenzo Veronese[1] Stefano Calzavara[2] Matteo Maffei[1]
[1] *TU Wien* [2] *Università Ca' Foscari Venezia & OWASP*

Abstract

Related-domain attackers control a sibling domain of their target web application, e.g., as the result of a subdomain takeover. Despite their additional power over traditional web attackers, related-domain attackers received only limited attention from the research community. In this paper we define and quantify for the first time the threats that related-domain attackers pose to web application security. In particular, we first clarify the capabilities that related-domain attackers can acquire through different attack vectors, showing that different instances of the related-domain attacker concept are worth attention. We then study how these capabilities can be abused to compromise web application security by focusing on different angles, including cookies, CSP, CORS, postMessage, and domain relaxation. By building on this framework, we report on a large-scale security measurement on the top 50k domains from the Tranco list that led to the discovery of vulnerabilities in 887 sites, where we quantified the threats posed by related-domain attackers to popular web applications.

1 Introduction

The Web is the most complex distributed system in the world. Web security practitioners are well aware of this complexity, which is reflected in the threat modeling phase of most web security analyses. When reasoning about web security, one has to consider multiple angles. The *web attacker* is the baseline attacker model that everyone is normally concerned about. A web attacker operates a malicious website and mounts attacks by means of standard HTML and JavaScript, hence any site operator in the world might act as a web attacker against any other service. High-profile sites are normally concerned about *network attackers* who have full control of the unencrypted HTTP traffic, e.g., because they operate a malicious access point. Both web attackers and network attackers are well known to web security experts, yet they do not capture the full spectrum of possible threats to web application security.

In this paper we are concerned about a less known attacker, referred to as *related-domain attacker* [9]. A related-domain attacker is traditionally defined as a web attacker with an extra twist, i.e., its malicious website is hosted on a sibling domain of the target web application. For instance, when reasoning about the security of `www.example.com`, one might assume that a related-domain attacker controls `evil.example.com`. The privileged position of a related-domain attacker endows it, for instance, with the ability to compromise cookie confidentiality and integrity, because cookies can be shared between domains with a common ancestor, reflecting the assumption underlying the original Web design that related domains are under the control of the same entity. Since client authentication on the Web is mostly implemented on top of cookies, this represents a major security threat.

Despite their practical relevance, related-domain attackers received much less attention than web attackers and network attackers in the web security literature. We believe there are two plausible reasons for this. First, related-domain attackers might sound very specific to cookie security, i.e., for many security analyses they are no more powerful than traditional web attackers, hence can be safely ignored. Moreover, related-domain attackers might appear far-fetched, because one might think that the owner of `example.com` would never grant control of `evil.example.com` to untrusted parties.

Our research starts from the observation that both previous arguments have become questionable, and this is the right time to take a second look at the threats posed by related-domain attackers, which are both relevant and realistic. A key observation to make is that a related-domain attacker shares the same *site* of the target web application, i.e., sits on the same registrable domain. The notion of site has become more and more prominent for web security over the years, going well beyond cookie confidentiality and integrity issues. For example, the Site Isolation mechanism of Chromium ensures that pages from different sites are always put into different processes, so as to offer better security guarantees even in presence of bugs in the browser [44]. Moreover, major browsers are now changing their behavior so that cookies are only attached to same-site requests by default, which further differentiates related-domain attackers from web attackers. In the rest of

the paper, we discuss other (normally overlooked) examples where the privileged position of related-domain attackers may constitute a significant security threat. Finally, many recent research papers showed that *subdomain takeover* is a serious and widespread security risk [8,33]. Large organizations owning a huge number of subdomains might suffer from incorrect configurations, which allow an attacker to make subdomains resolve to a malicious host. This problem also received attention from the general media [40] and the industry [7]. Though these studies proved that related-domain attackers are a realistic threat, they never quantified their impact on web application security at scale.

Contributions

In the present paper, we perform the first scientific analysis of the dangers represented by related-domain attackers to web application security. In particular:

1. We introduce a fine-grained definition of related-domain attacker that captures the capabilities granted to such attackers according to the position they operate and the associated web security threats. In particular, we systematize the attack vectors that an attacker can exploit to gain control of a domain, and we present the attacks that can be launched from that privileged position, discussing the additional gain with respect to a traditional web attacker (§3).

2. We implement a toolchain to evaluate the dangers that related-domain attackers can pose to web application security. Our toolchain builds on top of an analysis module for subdomain takeover, which significantly improves over previous results [33]. We use the output of this module to perform automated web application security analyses along different angles, including cookies, CSP, CORS, postMessage, and domain relaxation (§4).

3. We report on experimental results established through our toolchain. In particular, we enumerate 26M subdomains of the top 50k registrable domains from the Tranco list and discover practically exploitable vulnerabilities in 887 domains, including major websites like `cnn.com`, `nih.gov`, `harvard.edu`, and `cisco.com`. We also study the security implications of 31 third-party service providers and dynamic DNS and present a novel subdomain hijacking technique that resulted in a bug bounty of $1,000. Importantly, we quantify for the first time the impact of these vulnerabilities on web application security, concluding that related-domain attackers have an additional gain compared to web attackers that goes beyond well-studied issues on cookies (§5).

We have responsibly disclosed the identified vulnerabilities to the respective site operators. For space reasons, the results of the notification process are shown in Appendix A.

Table 1: Main DNS record types.

Record Type	Description
`A`	Returns the IPv4 address of a domain
`AAAA`	Returns the IPv6 address of a domain
`CNAME`	Maps an alias name to the canonical domain name
`NS`	Defines the authoritative DNS record for a domain
`CAA`	Specifies the allowed certificate authorities for a domain

2 Background

DNS Resolution. DNS is a protocol that stands at the core of the Internet [36]. It translates mnemonic domain names to IP addresses used by the underlying network layer to identify the associated resources. The translation process, called *DNS resolution*, is done transparently to applications. For instance, when a browser attempts to visit a fully qualified domain name (FQDN), such as `www.example.com`, the local resolver forwards the request to one of the DNS servers designated by the operating system. In case the DNS server has no information on the requested domain name, it initiates the recursive resolution from the root DNS server until the *authoritative* DNS server for the domain is reached, following the *subdomain* hierarchy of the DNS system. Eventually, the authoritative DNS server returns to the client a set of Resource Records (RRs) with the format: *name*, *TTL*, *class*, *type*, *data*. A list of relevant DNS record types is summarized in Table 1.

DNS also supports *wildcard* RRs with the label `*`, such as `*.example.com`. Wildcard RRs are not matched if an explicit RR is defined for the requested name. In general, wildcard RRs have a lower priority than standard RRs [31]. For instance, given a wildcard `A` record `*.example.com` and an `A` record for `a.example.com`, requests to `b.example.com` and `c.b.example.com` are resolved by the wildcard, while requests to `a.example.com` are matched by the corresponding `A` record. Notice that `c.a.example.com` is not resolvable.

Public Suffix List. While DNS defines the hierarchical structure of domain names, the Public Suffix List (PSL) is a catalog of domain suffixes controlled by registrars [38]. In contrast to Top-Level Domains (TLDs) that are defined in the Root Zone Database [27], such as `.com`, `.org`, `.net`, the suffixes listed in the PSL are called *effective TLDs* (eTLDs) and define the boundary between names that can be registered by individuals and private names. A domain name having just one label at the left of a public suffix is commonly referred to as *registrable domain*, *eTLD+1*, or *apex domain*. Domains sharing the same eTLD+1 are said to belong to the same *site*.

Cookies are scoped based on the definition of site, i.e., subdomains of the same site can share cookies (*domain cookies*) by setting their `Domain` attribute to a common ancestor. This attribute can never be set to a member of the PSL: for instance, since `github.io` is in the PSL, `foo.github.io` is not allowed to set cookies for `github.io`. This means that there is no way to share cookies between different GitHub Pages hosted sites.

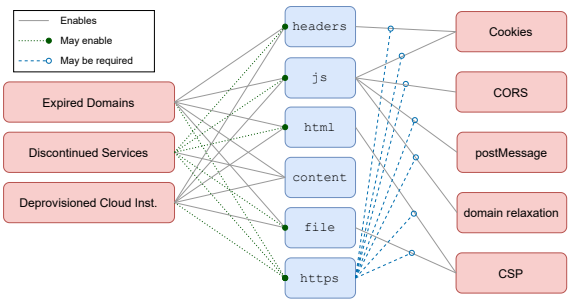


Figure 1: Summary of related-domain attacker instances for dangling DNS records.

3 The Related-Domain Attacker

We revise the threat model of the related-domain attacker in light of the directions that the Web has taken in recent years. In particular, we systematize for the first time the different attack vectors that can be exploited to escalate to a related-domain position. We also factorize the related-domain attacker into a set of capabilities and we express prerequisites of web attacks in terms of them, as presented below and summarized in Figure 1 for the most common subdomain takeover vulnerabilities [33]. This systematization allows for a quantification of the related-domain attacker problem, which we conduct in §5 by a large-scale measurement in the wild.

3.1 Threat Model

In its original definition, the related-domain attacker is a web attacker who operates a malicious website that is hosted on a *related domain* of the target website [9]. Two domains are *related* if they share a suffix that is not included in the PSL. For instance, consider the target site `example.com`: all its subdomains are related to the target, as well as being related to each other. Network attackers are traditionally considered out of scope, given that they could mount *person-in-the-middle* attacks via, e.g., ARP spoofing and DNS cache poisoning, which allow to easily control the IP address of any hostname accessed by the victim [14].

Subdomain takeovers are often caused by DNS misconfigurations [8, 33], with consequences ranging from altering the content of a page to full host control. Additionally, organizations frequently assign a subdomain of their corporate domain to their users, who could maliciously take advantage of this implicit trust. Vulnerable web applications can also be infiltrated to increase the privileges of attackers interested in exploiting their related domains.

As we elaborate in the following, the attack vector exploited to acquire a related-domain position is not a detail, but has an impact on the *capabilities* granted to the attacker. While full control of the host grants the attacker the ability to configure the web server to host arbitrary content, other attack scenarios only grant more limited power. For example, exploiting a reflected XSS on a subdomain of a company poses several restrictions on the actions that can be undertaken by the attacker. This motivates the need for a new, fine-grained definition of related-domain attacker, which precisely characterizes its power based on the acquired capabilities. In §3.2, we map concrete attack vectors to the set of capabilities (see Table 2) that the attacker may acquire when escalating to a related-domain position. In §3.3, we link such capabilities to web security threats, giving rise to a granular framework that defines different instances of the related-domain attacker.

Table 2: Capabilities of the related-domain attacker.

Capability	Description
`headers`	access and modify HTTP headers
`js`	arbitrary JavaScript code execution
`html`	alter the markup of the website with the exclusion of JavaScript
`content`	alter the textual content of the website with the exclusion of embed tags, frames and JavaScript code
`file`	host arbitrary files
`https`	operate a website under HTTPS with a valid certificate

Note: `js` subsumes both `html` and `content`, since it is possible to edit the DOM by using JavaScript. Similarly, `html` subsumes `content`.

3.2 Abusing Related Domains

We provide a comprehensive characterization of the attack vectors that can be exploited to acquire a related-domain position and identify the set of associated capabilities. While some of these attack vectors have been already analyzed in the literature in isolation (e.g., dangling DNS records [33] and domain shadowing [7, 34]), it is the first time they are systematized to cover the possible abuses which enable escalation to a related-domain position. Furthermore, we introduce a novel attack vector that exploits DNS wildcards, and we point out concrete instances of roaming services, hosting providers, and dynamic DNS services which are vulnerable to the threats described in this work.

3.2.1 Dangling DNS Records

Dangling DNS records refer to records in the authoritative DNS servers of a domain that point to expired resources. These records should be purged right away after releasing the pointed resources. Unfortunately, this practice is often overlooked, resulting in dangling DNS records to persist indefinitely. Possible reasons include lack of communication between the person who releases the resource and the domain owner or when the pointed resource expires automatically after a certain period of time, passing unnoticed. A dangling DNS record is considered vulnerable if an unintended party can take control of the expired resource [33].

Expired Domains. A DNS CNAME record maps a domain name (*alias*) to another one called canonical name. If the canonical name is expired, a third party can simply register the domain and serve arbitrary content under the alias domain.

Attackers exploiting this vulnerability have full control of the host and generally can rely on all the capabilities listed in our framework. One exception is `https` in presence of a CAA DNS record [25]: this record defines a list of Certificate Authorities (CAs) which are allowed to issue certificates for a given domain, possibly preventing attackers to rely on automated CAs like Let's Encrypt [2].

Discontinued Services. Third-party services are widely used to extend the functionalities of a website. Domain owners can integrate rich platforms by making them accessible under a subdomain of their organization, e.g., `blog.example.com` could show a blog hosted by WordPress and `shop.example.com` could be an e-shop run by Shopify. To map a (sub)domain to a service, an integrator typically has (i) to configure a DNS record for the (sub)domain, such as `A/AAAA`, `CNAME` or `NS`, to point to a server controlled by the service provider, and (ii) to claim the ownership of the (sub)domain in the account settings of the service. If the service provider does not verify the domain ownership explicitly, i.e., a DNS record pointing to the service is the only condition required to claim the ownership of a (sub)domain, an attacker could map to their account any unclaimed (sub)domain with a valid DNS record in place [33].

In addition, we observe that dangling records can also occur due to the presence of DNS wildcard. Consider, for example, a site operator configuring a DNS wildcard such as `*.example.com` pointing to a service provider IP to enable multiple websites to be hosted under subdomains of `example.com`. An attacker could bind a subdomain of their choice, e.g., `evil.example.com`, to a new account on the service provider. Surprisingly, we discovered that some service providers do not verify the ownership of a subdomain even if the parent domain has been already mapped to an existing account. In practice, this allows an attacker to claim `evil.proj.example.com` also in presence of a legitimate binding for `proj.example.com`. Even worse, we found that some service providers perform an automatic redirection of the `www`-prefixed subdomains to their parent domains without preventing the `www` subdomain from being associated to a different account. We report on this novel attack in §5.1.2.

Attackers' capabilities vary depending on the platform and range from altering the content of a single page to full host control. We refer to §5 for the result of a thorough security investigation conducted on 31 service providers.

Deprovisioned Cloud Instances. The ephemeral nature of resources allocated in Infrastructure as a Service (IaaS) environments is known to facilitate the spread of dangling DNS records. DNS records pointing to available IP addresses in the cloud can be abused by a determined attacker who rapidly allocates IP addresses in order to control the target of the dangling DNS record [8, 33]. Similarly to expired domains, the presence of a CAA DNS record in a parent domain could hinder the capability of obtaining a valid TLS certificate.

3.2.2 Corporate Networks and Roaming Services

Large organizations often assign fully qualified domain names (FQDNs) to devices in their network. This practice allows to statically reference resources in the network, irrespective of the assignment of IP addresses that may change over time. Although hosts might be inaccessible from outside of the organization network, internal users are put in a related-domain attacker position with full capabilities, excluding `https` that depends on the network configuration of the organization.

Institutions providing roaming services are similarly prone to the same issue. This is the case of eduroam, a popular international education roaming service that enables students and researchers to have a network connection provided by any of the participating institutions. As a novel insight, we discovered that system integrators at some local institutions are assigning eduroam users a subdomain of the main institution, such as `ip1-2-3-4.eduroam.example.com`, where `1.2.3.4` is a placeholder for the public IP assigned to the user connected to the eduroam network. This practice ultimately promotes any eduroam user to a related-domain attacker with full control of the host that is pointed by the DNS record. Firewall restrictions might hinder complete visibility on the Internet of the personal device of the user. Still, users' devices might be accessible within the institution network.

3.2.3 Hosting Providers and Dynamic DNS Services

Many service providers allow users to create websites under a specific subdomain, e.g., `<username>.github.io` on GitHub. Subdomains hosting user-supplied content are not related to each other if the parent domain is included in the PSL, as in the case of `github.io`. Unfortunately, several service providers that we reviewed did not include their domains in the PSL, turning any of their users into a related-domain attacker for all the websites hosted on the same platform.

A similar consideration applies to dynamic DNS providers. The race to offer a huge variety of domains under which users can create their custom subdomains, made it unfeasible for certain providers to maintain a list of entries in the PSL. The FreeDNS service [24] pictures well the problem, with 52,443 offered domains and a declared user base of 3,448,806 active users as of October 2020, who are in a related-domain attacker position to all the subdomains and domains of the network, since none of them has been added to the PSL.

While in the case of hosting and service providers, the capabilities granted to the attacker largely depend on the specific service (see §5.1.2 for more details), a dynamic DNS service allows users to point a DNS record to a host they fully control, capturing all the capabilities discussed in Table 2.

3.2.4 Compromised Hosts/Websites

Aside from scenarios in which attackers gain control of a resource that is either abandoned or explicitly assigned to them,

another way to obtain a related-domain attacker position is the exploitation of vulnerable hosts and websites. Intuitively, attackers achieving code execution on the vulnerable application have capabilities ranging from serving arbitrary content to full host control. If the exploited vulnerability is an XSS, attackers could take advantage of the ability to execute JavaScript code from a privileged position to escalate the attack against a more sensitive website.

Furthermore, attackers have been found employing a technique called *domain shadowing* [7, 34] to illicitly access the DNS control panel of active domains to distribute malware from arbitrary subdomains. Alowaisheq et al. recently discovered that stale `NS` records [5] could also be abused by attackers to take control of the DNS zone of a domain to create arbitrary DNS records. Controlling the DNS of a domain is the highest privileged setting for a related-domain attackers, since they can point subdomains to hosts they fully control and reliably obtain TLS certificates.

3.3 Web Threats

We identify for the first time a comprehensive list of web security threats posed by related-domain attackers, discussing in particular the scenarios where a related-domain attacker might have an advantage over traditional web attackers. While there exists ample literature on threats to cookies confidentiality and integrity posed by related-domain attackers [15, 62], in this work we focus on a complete account of how related-domain attackers affect web application security by exploring less-studied mechanisms.

3.3.1 Inherent Threats

Related-domain attackers sit on the same site of their target web application. This is weaker than sharing the same *origin* of the target, which is the traditional web security boundary, yet it suffices to abuse the trust put by browser vendors and end users on same-site content. We discuss examples below.

Trust of End Users. End users might trust subdomains of sites they are familiar with more than arbitrary external sites. For instance, attackers could exploit the residual trust associated with the subdomain's prior use [30] or deceive users into inserting their passwords provided by a password manager [56]. This is particularly dangerous on some mobile browsers, which display only the rightmost part of the domain due to the smaller display size, hence a long subdomain might erroneously look like the main site. Attackers could similarly abuse the trust inherited from the apex domain to use compromised subdomains for the distribution of malware or other types of dangerous content [34].

Site Isolation. Site Isolation is a browser architecture first proposed and implemented by the Google Chrome browser, which treats different sites as separate security principals requiring dedicate rendering processes [44]. These processes can access sensitive data for a single site only, which mitigates the leakage of cross-origin data via memory disclosure and renderer exploits, including attacks based on Spectre [29, 47]. As acknowledged in the original Site Isolation paper [44], "cross-origin attacks within a site are not mitigated", hence related-domain attackers can void the benefits of this security architecture.

Same Site Request Forgery. The introduction of *same-site cookies* [59] and the recent enforcement of this security feature by default on major browsers [20, 54] received high praise as an effective countermeasure against CSRF [26]. In the absence of other defenses [6], the restrictions introduced by same-site cookies are voided by a related-domain attacker who can mount a *same-site* request forgery attack just by including an HTML element pointing to the target website in one of their web pages.

3.3.2 Cookie Confidentiality and Integrity

Cookies can be issued with the `Domain` attribute set to an ancestor of the domain setting them, so as to share them with all its subdomains. For example, `good.foo.com` can issue a cookie with the `Domain` attribute set to `foo.com`, which is sent to both `good.foo.com` and `evil.foo.com`. Hence, related-domain attackers can trivially break *cookie confidentiality* and abuse of stolen cookies [62], e.g., to perform session hijacking. The `Domain` attribute poses risks to *cookie integrity* too: `evil.foo.com` can set cookies for `good.foo.com`, which can be abused to mount attacks like session fixation. Note that the integrity of host-only cookies is at harm too, because a related-domain attacker can mount *cookie shadowing*, i.e., set a domain cookie with the same name of a host-only cookie to confuse the web server [62].

Site operators can defend against such threats by careful cookie management. For example, they can implement (part of) the session management logic on top of host-only cookies, which are not disclosed to related-domain attackers. Moreover, they can use the `__Host-` prefix to ensure that security-sensitive cookies are set as host-only, thus ensuring their integrity against related-domain attackers.

Capabilities. The capabilities required by a related-domain attacker to break the confidentiality of a domain cookie depend on the flags enabled for it: if the cookie is `HttpOnly`, it cannot be exfiltrated via JavaScript and the `headers` capability is needed to sniff it; otherwise, just one between `headers` and `js` suffices. If the `Secure` flag is enabled, the cookie is sent only over HTTPS, hence the `https` capability is also required. As to integrity, all cookies lacking the `__Host-` prefix have low integrity against a related-domain attacker with the `headers` or `js` capabilities, since they are affected by cookie shadowing. There is one exception: cookies using the `__Secure-` prefix have low integrity only against related-domain attackers which additionally have the `https` capability, since these cookies can only be set over HTTPS.

3.3.3 Bypassing CSP

Content Security Policy (CSP) is a client-side defense mechanism originally designed to mitigate the dangers of content injection and later extended to account for different threats, e.g., click-jacking. CSP implements a whitelisting approach to web application security, whereby the browser behavior on CSP-protected web pages is restrained by binding *directives* to sets of *source expressions*, i.e., a sort of regular expressions designed to express sets of origins in a compact way. To exemplify, consider the following CSP:

```
script-src foo.com *.bar.com;
frame-ancestors *.bar.com;
default-src https:
```

This policy contains three directives, `script-src`, `frame-ancestors` and `default-src`, each bound to a set of source expressions like `foo.com` and `*.bar.com`. It allows the protected page to: (i) include scripts from `foo.com` and any subdomain of `bar.com`; (ii) be included in frames opened on pages hosted on any subdomain of `bar.com`; (iii) include any content other than scripts over HTTPS connections with any host.

Since the syntax of source expressions naturally supports the whitelisting of any subdomain of a given parent, related-domain attackers represent a major threat against the security of CSP. For example, if an attacker could get control of `vuln.bar.com`, then they would be able to bypass most of the protection put in place by the CSP above. In particular, the attacker would be able to exploit a content injection vulnerability on the CSP-protected page to load and execute arbitrary scripts from `vuln.bar.com`, thus voiding XSS mitigation. Moreover, the attacker could frame the CSP-protected page on `vuln.bar.com` to perform click-jacking attacks. To avoid these threats, site operators should carefully vet the subdomains included in their CSP whitelists.

Capabilities. A related-domain attacker requires the capability to upload arbitrary files on the website under its control to void the protection offered by CSP against content inclusion vulnerabilities, with the only notable exception of frame inclusion which requires only the `html` capability. For active contents [37], i.e., those that may have access to the DOM of the page, the attacker also needs the `https` capability if the target page is hosted over HTTPS. Regarding click-jacking protection, attackers only requires the `html` capability to include the target website on a page under their control.

3.3.4 Abusing CORS

Cross-Origin Resource Sharing (CORS) is the standard approach to relax the restrictions enforced by SOP on cross-origin communications, i.e., preventing JavaScript from reading the content of responses to cross-origin requests. Consider a service at `https://www.example.com`, which needs to fetch sensitive data from `api.example.com` via JavaScript: to enable CORS, `https://api.example.com` can inspect the `Origin` header of incoming requests to detect if they come from `https://www.example.com` and, in such a case, set a CORS header `Access-Control-Allow-Origin` with the value `https://www.example.com` in the response. As an additional layer of protection, the server must also set the `Access-Control-Allow-Credentials` header to `true` if the request includes credentials, e.g., cookies, since the associated response is more likely to include sensitive content.

Related-domain attackers can abuse CORS to bypass the security restrictions put in place by SOP when the aforementioned server-side authorization checks are too relaxed, i.e., read access is granted to arbitrary subdomains. For example, if `https://api.example.com` was willing to grant cross-origin access to any subdomain of `example.com` besides `www.example.com`, a related-domain attacker could get unconstrained access to its data. To avoid these threats, site operators should be careful in the security policy implemented upon inspection of the `Origin` header, e.g., restricting access just to a few highly trusted subdomains.

Capabilities. To exploit CORS misconfigurations, an attacker needs the `js` capability to issue requests via JavaScript APIs like `fetch` and access the content of the response. The `https` capability may be required depending on the CORS policy deployed by the site operator.

3.3.5 Abusing postMessage

The postMessage API supports cross-origin communication across windows (e.g., between frames or between a page and the popup opened by it). The sender can invoke the `postMessage` method of the target window to transmit a message, possibly restricting the origin of the receiver. The receiver, in turn, can use event handlers to listen for the `message` event and process incoming messages.

Despite its apparent simplicity, the postMessage API should be used with care, as shown by prior research [50, 51]. In particular, when sending confidential data, one should always specify the origin of the intended receiver in the `postMessage` invocation. When receiving data, instead, one should check the origin of the sender (via the `origin` property of the received message) and appropriately sanitize the content of the message before processing it.

Related-domain attackers can undermine web application security when site operators put additional trust in subdomains. In particular, related-domain attackers can try to abuse their position to void the aforementioned origin checks and communicate with inattentive receivers that might process messages in an unsafe way, e.g., messages are provided as input to `eval` or stored in a cookie, opening the way to session hijacking attacks. Site operators can defend against such attacks by carefully vetting authorized subdomains for communication between windows.

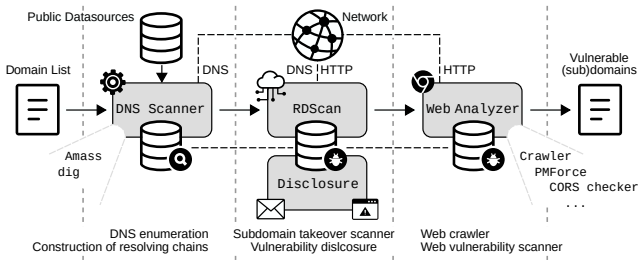


Figure 2: Vulnerability scanning pipeline.

Capabilities. An attacker requires scripting capabilities (`js`) to open a new tab containing the vulnerable page and communicate with it via the postMessage API. Similarly to CORS, `https` may be needed depending on the origin checking performed by the receiver.

3.3.6 Abusing Domain Relaxation

Domain relaxation is the legacy way to implement communication between windows whose domains share a common ancestor. Assume that a page at `a.example.com` opens a page at `b.example.com` inside a frame. Besides using the postMessage API as explained, the two frames can communicate by relaxing their `document.domain` property to a common ancestor. In this case, both frames can set such property to `example.com`, thus moving into a same-origin position.[1] After that, SOP does not enforce any isolation between the two frames, which can communicate by writing on each other's DOM. Note that `example.com` must explicitly set the `document.domain` property to `example.com` if it is willing to engage in the domain relaxation mechanism, although this is apparently a no-op.

Domain relaxation can be abused by related-domain attackers, who can look for pages which are willing to engage in such dangerous communication mechanism and abuse it. In particular, when the attacker moves into a same-origin position, SOP does not provide any protection anymore, which voids any confidentiality and integrity guarantee. Websites that are willing to communicate with a selected list of related domains should refrain from using this mechanism – which is deemed as insecure – and should implement cross-origin communication on top of the postMessage API.

Capabilities. Besides the `js` capability needed to perform the relaxation and access the DOM of the target page, attackers need to setup their attack page on the same protocol of the target, hence the `https` capability may also be required.

4 Analysis Methodology

We performed a large-scale vulnerability assessment to measure the pervasiveness of the threats reported in this work, first by identifying subdomains of prominent websites that can be abused by a related-domain attacker exploiting dangling DNS records, and second by evaluating the security implications on web applications hosted on related domains of the vulnerable websites. Our methodology is based on the pipeline summarized in Figure 2 and further described in this section.

4.1 DNS Data Collection

We enumerated the subdomains of the top 50k domains in the Tranco list [42] from March 2020.[2] The enumeration phase was based on `amass` [41], a state of the art information gathering tool backed by the OWASP project. The tool supports several techniques to maximize the chances of discovering subdomains of a target. In our configuration, we extracted subdomains using the following approaches: (i) fetch data from publicly available sources, such as Censys [17], certificate transparency logs [49], search engines, etc.; (ii) attempt DNS zone transfer to obtain the complete list of RRs defined for a certain DNS zone; (iii) inspect fields of TLS certificates, e.g., `Subject Alternative Name` and `Common Name`. To speed up the enumeration phase and lower the number of network requests, we avoided bruteforcing DNS resolvers against domain name wordlists. Similarly, we explicitly disabled the resolution of subdomain alterations.

We modified `amass` to compute the DNS *resolving chains* of all the domains obtained in the previous step. Similarly to [33], we define a resolving chain as a list of DNS RRs in which each element is the target of the previous one, starting from a DNS record of type `A/AAAA`, `CNAME` or `NS`. We ignore `MX` records because we focus on web attacks in this study. For `CNAME` and `NS` records, we recursively perform a DNS resolution until an `A/AAAA` RR is detected. Unterminated DNS resolving chains can occur in presence of a record pointing to an unresolvable resource or due to the abrupt termination of `amass` after reaching the execution timeout limit of 5 minutes. To ensure the correctness of the results, we recompute unterminated DNS resolving chains using the `dig` utility.

Starting from the set of 50k domains in the Tranco list, our framework identified 26 million valid subdomains. In a previous study, Liu et al. [33] used a relatively small wordlist of 20,000 entries to find possible subdomains of the Alexa top 10k list, 2,700 .edu domains, and 1,700 .gov domains. Compared to their work, our domain selection is penalized given that we do not restrict to specific TLD zones. For instance, .edu domains typically have a high number of subdomains in contrast to other categories (see §5.1.1). Nevertheless, our results outperform the findings of Liu et al. by discovering on average 13 times more subdomains.

[1] We assume here that the two frames share the same protocol and port.

[2] https://tranco-list.eu/list/ZKYG/1000000

4.2 RDScan

After populating a database with the DNS records of the discovered subdomains, the framework detects dangling records and verifies that all the preconditions to mount a subdomain takeover attack are met. By doing so, false positives are minimized in the analysis. This component, that we call *RDScan*, has three different modules that test for the presence of the vulnerable scenarios described in §3.2.1.

Expired Domains. The detection of expired domains is performed according to the following procedure: given a resolving chain that begins with a `CNAME` record, our tool checks if it points to an unresolvable resource and extracts the eTLD+1 of the canonical name at the end of the chain, that we call *apex* for brevity. Then, if the `whois` command on the apex domain does not return any match, RDScan queries GoDaddy to detect if the domain can be purchased. In this case, we consider the domain of the resolving chain, i.e., the alias of the first record of the chain, as vulnerable. Notice that we only tested domains that can be registered without special requirements, i.e., we did not consider .edu domains and other specific eTLDs not offered by the registrar.

Discontinued Services. The process of finding discontinued services is summarized in Algorithm 1. RDScan traverses each resolving chain to identify whether it points to one of the services supported by our framework. This step is implemented according to the documentation provided by individual services, and typically relies on checking for the presence of (i) an `A` record resolving to a specific IP address, (ii) the canonical name of a `CNAME` record matching a given host, or (iii) the existence of a `NS` record pointing to the DNS server of a service. (Sub)domains mapped to services are then checked to verify if the bindings between user accounts and (sub)domains are in place. For the majority of the services considered in this study, a simple HTTP request suffices to expose the lack of a correct association of a (sub)domain. Other services require active probing to determine whether a domain can be associated to a fresh test account that we created. This has been done using the automated browser testing library puppeteer with Chromium [1]. RDScan also performs the detection of DNS wildcards that might be abused as described in §3.2. A DNS wildcard for a domain such as `test.example.com` can be easily detected by attempting to resolve a `CNAME` or `A` DNS record for `<nonce>.test.example.com`, where `nonce` refers to a random string that is unlikely to match an entry in the DNS zone of the target domain.

Deprovisioned Cloud Instances. The detection of potentially deprovisioned cloud instances has been performed similarly to the probabilistic approach adopted by [8, 33]. We did not create any virtual machine or registered any service at cloud providers in this process. Instead, we collected the set of IP ranges of 6 major providers: Amazon AWS, Google Cloud Platform, Microsoft Azure, Hetzner Cloud, Linode, and OVHcloud. We tested each (sub)domain in our dataset to check whether the pointed IP was included in any of the cloud IP ranges. In case the IP falls within the address range of a cloud provider, we make sure that it does not point to a reserved resource such as a proxy or a load balancer. As the last step, we perform a liveness probe to determine if the IP is in use. This is done by executing a ping to the IP: if no answer is received, we use a publicly available dataset [43] comprising a scan of the full IPv4 range on 148 ports (128 TCP, 20 UDP). If no open ports for the given IP are found, we deem the resource as potentially deprovisioned.

Algorithm 1 Detection of Discontinued Services

Input: Set of DNS resolving chains RC, set of supported services S
Output: Set of vulnerable subdomains V_s

```
procedure DISCONTINUED_SERVICES(RC, S)
    V_s ← ∅
    for each chain ∈ RC do
        for each service ∈ S do
            ▷ Check if a record in the chain points to the service
            if chain points to service then
                d ← target_domain(chain)
                if d is unclaimed at service then
                    V_s ← V_s ∪ {d}
                ▷ Detect wildcard if the service allows a subdomain of a
                ▷ claimed domain to be mapped to a different account
                else if service vulnerable to wildcard issue then
                    r ← generate_nonce()
                    rd_chains ← compute_resolving_chains(r.d)
                    for each rd_chain ∈ rd_chains do
                        if rd_chain points to service then
                            V_s ← V_s ∪ {r.d}
```

4.3 Web Analyzer

Our web security analysis aims at quantifying the number of domains hosting web applications that can be exploited by taking over the vulnerable domains discovered by RDScan. In particular, for every apex domain with at least one vulnerable subdomain, we selected from the CommonCrawl dataset [19] the list of 200 most popular related domains according to the Pagerank score [10]. From the homepage of these domains, we extracted the same-origin links that appear in the HTML code. For each related domain, we considered the homepage and up to 5 of these URLs as the target of our web analysis, and we accessed these links using the Chromium browser automated by puppeteer. In the following, we present the data collection process and the security analyses we have conducted to identify the threats discussed in §3.3. We postpone the summary of the results to §5.

4.3.1 Analysis of Cookies

We used the puppeteer API to collect cookies set via HTTP headers and JavaScript. Our goal is to identify cookies affected by confidentiality or integrity issues. In particular, we flag a cookie as affected by confidentiality issues if, among the

related domains vulnerable to takeover, there exists a domain d such that:

- d is a subdomain of the `Domain` attribute of the cookie;
- by taking over d, the attacker has acquired the capabilities required to leak the cookie.

We mark a cookie as affected by integrity issues if:

- the name of the cookie does not start with `__Host-`;
- we identified a vulnerable domain that grants the capabilities required to set the cookie.

We also rely on a heuristic proposed by Bugliesi et al. [12] to statically identify potential (pre-)session cookies, i.e., cookies that may be relevant for the management of user sessions.

The capabilities required to perform these attacks depend on the security flags assigned to the cookie and the usage of cookie prefixes (see §3.3.2). For instance, to compromise integrity either the capability `js` or `headers` is required and, if the prefix `__Secure-` is used, `https` is also necessary.

4.3.2 Analysis of CSP policies

For this analysis, we implemented a CSP evaluator according to the draft of the latest CSP version [55], which is currently supported by all major browsers. This is not a straightforward task, due to the rich expressiveness of the policy and various aspects that have been introduced into the specification for compatibility purposes across different CSP versions, e.g., for scripts and styles, the `'unsafe-inline'` keyword, which whitelists arbitrary inline contents in a page, is discarded when hashes or nonces are also specified.

In our analysis, we focus on the protection offered against click-jacking and the inclusion of active contents [37], i.e., resources that have access to (part of) the DOM of the embedding page. This class of contents includes scripts, stylesheets, objects, and frames.

For each threat considered in our analysis, we first check if the policy is unsafe with respect to any web attacker. This is the case for policies that allow the inclusion of contents from any host (or framing by any host, when focusing on click-jacking protection). For scripts and styles, the policy is also deemed unsafe if arbitrary inline contents are whitelisted. If the policy is considered safe, we classify it as exploitable by a related domain if one of the vulnerable domains detected by RDScan is whitelisted and the attacker acquires the relevant capabilities to perform the attack, which vary depending on the threat under analysis (see §3.3.3). For instance, script injection requires the `file` capability, given that attackers need to host the malicious script on a subdomain they control. Moreover, if the page to attack is served over HTTPS, the `https` capability is required due to the restrictions imposed by browsers on mixed content [37].

4.3.3 Analysis of CORS

To evaluate the security of the CORS policy implemented by a website, we perform multiple requests with different `Origin` values and inspect the HTTP headers in the response to understand whether CORS has been enabled by the server.

Inspired by the classes of CORS misconfigurations identified in [18], we test 3 different random origins with the following characteristics: (i) the domain is a related domain of the target URL; (ii) the domain starts with the registrable domain of the target URL; (iii) the domain ends with the registrable domain of the target URL. While the first test verifies whether CORS is enabled for a related domain, the other two detect common server-side validation mistakes. Such errors include the search of the registrable domain as a substring or a suffix of the `Origin` header value, which results in having, e.g., `www.example.com` whitelisting not only `a.example.com` but also `atkexample.com`. For each test, we check if the `Access-Control-Allow-Origin` header is present in the response and if its value is either `*` or that of the `Origin` header contained in the request. We also control if the `Access-Control-Allow-Credentials` header is present and set to `true` (when `Access-Control-Allow-Origin` differs from `*`) to identify the cases in which requests with credentials are allowed.

We report a CORS deployment as vulnerable to web attackers if either the second or the third test succeeds. Instead, a page is exploitable exclusively by a related-domain attacker if only the first test succeeds and, among the vulnerable related domains discovered by RDScan, one grants the `js` capability to the attacker. Since in our tests we use the same protocol of the page under analysis in the `Origin` header, we conservatively require the `https` capability when HTTPS is used.

4.3.4 Analysis of postMessage Handlers

PMForce [51] is an automated in-browser framework for the analysis of postMessage event handlers. It combines selective force execution and taint tracking to extract the constraints on the message contents (e.g., presence of a certain string in the message) that lead to execution traces in which the message enters a dangerous sink that allows for code execution (e.g., `eval`) or the alteration of the browser state (e.g., `document.cookie`). A message satisfying the extracted constraints is generated using the Z3 solver and the handler under analysis is invoked with the message as a parameter to ensure that the exploit is successfully executed.

We integrated PMForce in our pipeline and modified it to generate, for each handler, multiple exploit messages with the same contents but a different `origin` property, e.g., a related-domain origin and a randomly-generated cross-site origin. We consider a page vulnerable to any web attacker if any of its handlers is exploitable from a cross-site position. Instead, we consider a page exploitable by a related-domain attacker if its handlers can be exploited only from a related-domain

position and one of the vulnerable domains discovered by RDScan grants the `js` capability to the attacker, which is required to open a tab and send messages to it. If the handlers whitelist only HTTPS origins, then the capability `https` is also required to mount the attack.

4.3.5 Analysis of Domain Relaxation

As a first step, the analyzer detects whether the property `document.domain` is set after the page is loaded. This task is straightforward except for the case in which the page sets the property to its original value (see §3.3.6) since this cannot be detected just by reading the value of `document.domain`. To identify this particular case, we leverage puppeteer APIs to:

- inject a frame from a (randomly generated) subdomain of the page under analysis;
- intercept the outgoing network request and provide as response a page with a script that performs domain relaxation and tries to access the parent frame, which succeeds only if the parent has set `document.domain`.

The relaxation mechanism is exploitable by a related-domain attacker if RDScan discovered a vulnerable subdomain (which is a subdomain of the value of `document.domain`) that grants the `js` capability to the attacker. If the webpage is hosted over HTTPS, the `https` capability is also required.

4.4 Heuristics and False Positives

Our methodology is based on testing sufficient preconditions to execute the reported attacks, thus minimizing false positives. Nevertheless, the scanning pipeline makes use of two heuristics in the RDScan and web analyzer modules to, respectively, detect potentially deprovisioned cloud instances and label security-sensitive cookies; moreover, we identify a potential TOCTOU issue between the two modules of the analysis pipeline. We discuss below why this has only a marginal effect on the overall results of the analysis.

RDScan. We developed automated procedures to test sufficient preconditions for a takeover. Expired domains are trivially verified by checking if the target domain can be purchased. For discontinued services, we created personal testing accounts on each service considered in the analysis and used these accounts to probe the mapping between the target subdomain and the service. If we detect all necessary conditions to associate the subdomain to our account, we deem it as vulnerable. We manually vetted these conditions against our own domain. Due to ethical concerns, we did not mount attacks against real websites, but we reviewed all the occurrences of subdomain takeover vulnerabilities before disclosing them to the affected sites and found no false positives in the results (see Appendix A). The detection of subdomains pointing to deprovisioned cloud instances relies instead on a heuristic which might introduce false positives, as discussed in §4.2. We performed this investigation to capture the magnitude of the problem, but we excluded the results on deprovisioned cloud instances from the pipeline to avoid false positives in the web analyzer. To avoid misunderstandings in the paper, we refer to domains matching our heuristic as *potentially vulnerable*.

Web Analyzer. The web vulnerabilities discovered by this module have been identified via dynamic testing and analysis of the data collected by the crawler. We manually verified samples of each detected vulnerability to ensure the correctness of the results and confirmed the absence of false positives. The usage of heuristics is limited to the labeling of cookies which likely contain session identifiers and are thus particularly interesting from a security standpoint; this approach has been proved reasonably accurate in prior work [12].

Interplay between the modules. The modules of the pipeline described in Figure 2 have been executed in sequence at different points in time. The DNS enumeration phase terminated in June 2020, while RDScan ran during the first half of July 2020. The severity of the discovered issues motivated us to immediately report them to the affected parties. Therefore, we launched a large-scale vulnerability disclosure campaign in the second half of the month. We executed the web scanner right after that. Having the DNS data collection running first, RDScan might have missed new subdomains that were issued after the completion of the DNS enumeration. This leads to a possible *underestimation* of the threats in the wild concerning unresolvable domains and expired services. On the other hand, subdomain takeover vulnerabilities might have been fixed prior to the web security analysis. We performed a second run of RDScan 6 months later to verify the fix rate of notified parties. Surprisingly, we discovered that, as of January 2021, 85% of the subdomains that we tested are still affected by leftover subdomain takeover vulnerabilities, confirming that the early remediation of the reported vulnerabilities had a marginal effect on the web analysis. We provide more details on our large-scale disclosure campaign in Appendix A.

5 Security Evaluation

We report on the results of our security evaluation on the top 50k domains from the Tranco list. We quantify the vulnerabilities that allow an attacker to be in a related-domain position, and we provide a characterization of the affected websites. Then, we delve into the security of 31 service providers by discussing common pitfalls and the capabilities that could be abused by an attacker. Finally, we present the outcome of our web analysis, and we identify practical vulnerabilities by intersecting the capabilities on vulnerable domains with the threats found on web applications hosted on their related domains. Table 3 provides a breakdown of the results by combining attack vectors and web threats: the values reported in the cells represent the number of vulnerable domains/sites

compared to those deploying the corresponding web mechanism. We discuss these results in the following. Due to space constraints, we move representative examples of confirmed attacks to Appendix B.

5.1 Attack Vectors and Capabilities

RDScan identified 1,520 subdomains exposed to a takeover vulnerability, distributed among 887 domains from the top 50k of the Tranco list. Most of the vulnerabilities are caused by discontinued third-party services (83%), with expired domains being responsible for the remaining 17%. The analysis of deprovisioned cloud instances discovered 13,532 potentially vulnerable domains, confirming the prevalence of this threat as reported in previous work [33].

5.1.1 Characterization of Vulnerable Domains

As expected, the likelihood of a domain to be vulnerable is directly related to the breadth of its attack surface, i.e., the number of subdomains we found. Figure 3a pictures well this correlation, showing that around 15% of the domains with more than 50,000 subdomains are vulnerable. Figure 3b outlines the distribution of vulnerable domains depending on the rank of each site in the Tranco list. Sites in top positions are more likely to have a vulnerable subdomain than those with a lower rank.

The analyzed websites have been further partitioned into categories in Figure 3c. Special care has to be taken when considering dynamic DNS: the 49 domains listed in this category are those used by dynamic DNS services, such as `ddns.net`, `noip.com`, `afraid.org`. RDScan identified vulnerable subdomains belonging to 8 domains, but 4 of them were listed in the PSL. We excluded these domains from our analysis, given that taking control of one of their subdomains would not put the attacker in a related-domain position with respect to the parent domain. The same principle has been adopted when evaluating service and hosting providers offering subdomains to their users. We refer to §5.1.2 for a detailed analysis of Dynamic DNS services and hosting providers.

The second most affected category concerns education websites. We found that academic institutions generally have complex and heterogeneous public-facing IT infrastructures that translate into a high number of subdomains. By restricting the analysis to the `.edu` TLD, we observed 1,229 domains having on average 6,033 subdomains each. The percentage of domains with at least one vulnerable subdomain is 7.32%, which is substantially higher than any other TLD considered. For comparison, the percentage in `.com` is 1.81%.

Overall, we identified vulnerabilities affecting top domains across all categories. To exemplify, we found subdomain takeover vulnerabilities on news websites like `cnn.com` and `time.com`, university portals like `harvard.edu` and `mit.edu`, governmental websites like `europa.eu` and `nih.gov`, and IT companies like `lenovo.com` and `cisco.com`. Although most of the discovered issues could be easily fixed by routinely checking the validity of DNS records, our large-scale vulnerability assessment raises concerns due to the number and pervasiveness of the identified threats.

5.1.2 Analysis of Third-Party Services

We examined 26 service and hosting providers and 5 dynamic DNS services for a total of 31 third-party services. Our selection comprises services mentioned in previous work [33] and community efforts [22], excluding those that required payment to carry out our analysis.

The results are summarized in Table 4. We combined manual testing and review of the documentation to assess the capabilities available to a registered user of each service. We also evaluated the considered services against the security pitfalls described in §3.2.1: (i) *wildcard*, the domain ownership verification allows attackers to claim subdomains of an already mapped domain, e.g., due to the presence of a wildcard DNS entry; (ii) *redirect*, if the `www` subdomain of a mapped domain automatically redirects to the parent domain, e.g., `www.shop.example.com` redirects to `shop.example.com`, whether the former can be claimed by a different account; (iii) *PSL*, if the service allows users to create a website under a specific subdomain, whether the parent domain of the assigned website is included in the PSL.

Table 3 shows the distribution of the vulnerable subdomains across service providers. The majority of the vulnerable subdomains (93%) are hosted on the first four most used services: WordPress, Shopify, Tumblr, and GitHub Pages. These prominent services give users the ability to host a website with a valid TLS certificate for the associated domain. Users are allowed to customize the markup and JavaScript code of the pages, and for Tumblr and GitHub Pages, users are allowed to upload arbitrary files to their websites. In general, the capabilities obtained by an attacker controlling a service vary depending on the specific platform, ranging from `content` only (UptimeRobot) to full host control (ngrok). We found that 19 out of 26 services grant the `js` and `https` capabilities, while 21 provide the `js` capability alone. The `file` capability is the most uncommon, being available for 4 services only.

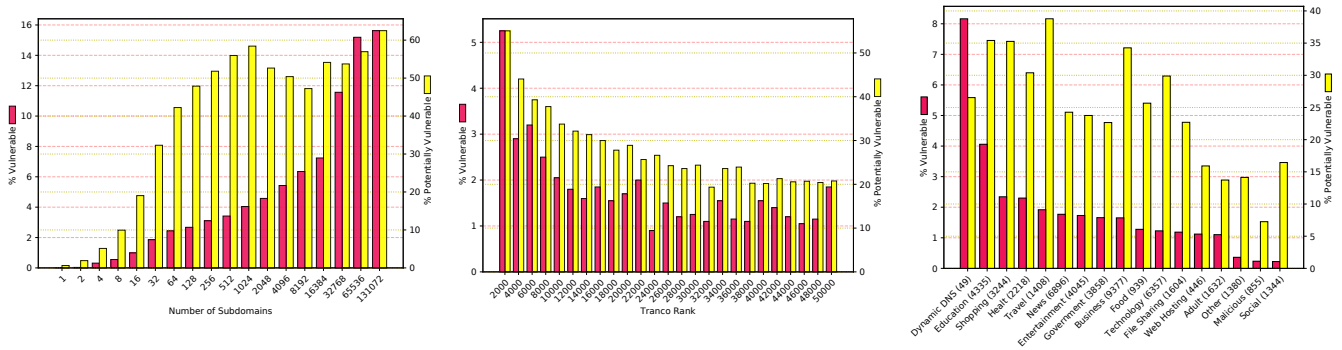
Surprisingly, we discovered that in 20 out of the 31 analyzed services, any registered user controls a website that is in a related-domain position to all the other websites hosted on the platform. Tumblr and WordPress, along with 11 additional services, even share their primary domain with user-controlled websites, e.g., `attacker.tumblr.com` is related to `tumblr.com`. Only GitHub and ngrok prevent this threat by including the apex domains assigned to their users in the PSL.

Lastly, we found that 17 services have issues with the ownership verification mechanism. Among the four most used services, only WordPress prevents attackers from claiming

Table 3: Breakdown of the results in terms of affected domains/sites.

Attack Vector	Takeover		Web mechanisms exploitable *exclusively* by related-domain attackers							
			Cookies		CSP		CORS		Relaxation	
	Domains	Sites	Domains	Sites	Domains	Sites	Domains	Sites	Domains	Sites
Expired Domains	260	201	5,394/5,394	195/195	35/141	13/28	35/317	16/107	9/11	6/8
Discontinued Services	1,260	699	18,798/19,020	662/674	104/294	32/75	196/1,980	37/392	49/88	24/55
↳ WordPress	466	320	13,803/13,803	312/312	43/168	23/52	164/1,221	21/186	30/49	14/28
↳ Shopify	326	254	2,638/2,638	244/244	32/66	5/12	26/459	11/153	7/19	5/15
↳ Tumblr	310	24	404/404	23/23	1/2	1/2	5/29	2/12	1/2	1/2
↳ GitHub	42	25	899/899	24/24	22/49	1/5	2/116	2/18	2/3	2/3
↳ Webflow	24	20	601/601	18/18	0/0	0/0	2/122	2/14	1/3	1/3
↳ Ngrok	22	13	250/250	13/13	7/9	2/2	0/17	0/5	8/11	1/3
↳ Helpscout	18	17	425/425	16/16	1/4	1/3	0/28	0/6	0/2	0/2
↳ Others	52	37	464/724	22/35	1/7	1/3	0/25	0/8	9/10	2/3
Total	**1,520**	**887**	**23,178/23,400**	**845/857**	**139/428**	**45/100**	**224/2,254**	**51/488**	**57/97**	**29/61**

Note: Deployment of CSP only considers policies that are not trivially exploitable by a web attacker (§4.3.2) and whitelist one or more related domains. CORS policies are only exposed to requests coming from whitelisted origins [18]: for the deployment we report the count of domains/sites vulnerable either to web attackers or related-domain attackers that were discovered during dynamic testing. postMessage is omitted since related-domain attackers have no gain compared to web attackers.



(a) # Subdomains (b) Tranco Rank (c) Categories

Figure 3: Characterization of vulnerable domains.

subdomains of an already mapped domain. Moreover, 8 service providers perform an automatic redirection from the `www` subdomain to the parent domain. Therefore, users of these services might erroneously assume that the `www` subdomain is implicitly bound to their account and cannot be claimed by others. Only Shopify and Launchrock do not prevent this subdomain from being mapped to different accounts. We reported to GitHub and Shopify, two of the major service providers, the vulnerabilities discovered on the domain ownership verification process. GitHub acknowledged the problem and told us that they "[...] are exploring various changes to the custom domain flow that will improve this situation by requiring formal domain ownership verification". Shopify awarded us $1,000 for the report and shipped a fix on April 12, 2021.

Dynamic DNS Services. The adoption of the PSL across different dynamic DNS providers is shown in Table 5, together with the number of domains that a user can choose from. We observed that only 2 providers listed all their domains in the PSL. Noip and DynDNS left out a small number of the domains they offer, but it is not clear to us whether this is due to negligence or if this is a deliberate choice. Instead, FreeDNS, with more than 50k domains, did not include any of them in the list, leaving their massive user base at risk. We reported this major flaw to the FreeDNS maintainer, who acknowledged it but took no action, as it would be impossible to maintain an updated list of thousands of domains in the PSL, given the lack of an API to manage PSL entries.

5.2 Web Threats

We now turn the attention to the web application security implications of our analysis, as summarized in Tables 3 and further detailed in Table 6.

We start by discussing confidentiality and integrity of session cookies. Overall, our crawler collected 85,169 cookies, out of which 24,924 have been labeled as session cookies by our heuristic. Among these, we identify 3,390 (14%) cookies from 5,051 (33%) domains on 687 sites (81%) whose confidentiality can be violated by a related-domain attacker. This shows that related-domain attackers can often get access to session cookies, which may enable attacks like session hijacking. Our analysis also shows that the state of cookie integrity is even worse: in particular, we identify 24,689 (99%) session cookies from 14,964 (99%) domains on 834 (99%) sites

Table 4: Attackers' capabilities on vulnerable services.

Service	Wildcard	Redirect (www)	PSL	Capabilities
agilecrm	!	–	!	js https
anima	!	–	–	js https
campaignmonitor	!	–	!	content
cargo	!	✓	!	js
feedpress	!	–	–	html
gemfury	!	–	–	file https
github	!	–	✓	js file https
helpscout	!	–	!	js file https
jetbrains	✓	–	!	content
launchrock	!	!	!	js https
ngrok	?	?	✓	js file headers https
persona	!	✓	!	js https
pingdom	!	–	–	js
readme.io	!	–	!	js https
shopify	!	!	!	js https
smartjobboard	!	✓	!	js https
statuspage	✓	–	!	js https
strikingly	?	?	!	js https
surgesh	✓	✓	!	js https
tumblr	!	–	!	js file https
uberflip	?	?	–	js https
uptimerobot	!	–	–	content
uservoice	?	?	!	js https
webflow	?	?	!	js https
wordpress	✓	✓	!	js https
worksites	!	✓	!	js https

Note: We use the following notation: service not affected (✓); service is vulnerable (!); the conditions of *redirect* and *PSL* do not apply (–); could not evaluate, e.g., due to payment required, no public registration form, etc. (?). Helpscout allows to host only arbitrary *active content* files (js, css); Gemfury allows to host only arbitrary *passive content* files (images, media, ...); Launchrock implicitly associates every subdomain to the mapped domain, not only the www subdomain.

Table 5: PSL on dynamic DNS services.

Service	# Domains	PSL
afraid (FreeDNS)	52,443	! 0/52,443
duckdns	1	✓ 1/1
dyndns	293	! 287/293
noip	91	! 85/91
securepoint	10	✓ 10/10

which do not have integrity against a related-domain attacker, hence may enable attacks like session fixation and cookie forcing. This increase comes from the fact that related-domain attackers can compromise the confidentiality of domain cookies alone, while they can break the integrity of any cookie by exploiting *cookie shadowing* [62]. The fraction of domains not affected by integrity issues is only due to the lack of capabilities available for the subdomain we could possibly take over. The only robust way to improve cookie integrity in this setting is the adoption of the __Host- prefix, which is unfortunately negligible in the wild: we only identified one cookie using it in our dataset.

Concerning CSP, the first observation we make is that, as reported by previous studies [15,46,57], the majority of CSPs in the wild suffer from incorrect configurations, voiding their security guarantees even against web attackers. Remarkably, however, related-domain attackers are more powerful than traditional web attackers for real-world CSPs, being able to bypass the protection mechanism on 139 additional domains. This is apparent for object injection, frame injection, and framing control. For example, we quantified the following

Table 6: Web security abuses by related-domain attackers (RDA).

	Mechanism	Deployed Domains	Deployed Sites	Exploitable by RDA Domains	Exploitable by RDA Sites
Cookies	all	23,400	857	*C* 15,025 *I* 23,178	826 845
	session	15,179	846	*C* 5,051 *I* 14,964	687 834
CSP	script inclusion	1,144	260	901 (0)	212 (0)
	style inclusion	961	232	930 (0)	225 (0)
	object inclusion	1,027	250	598 (+12)	123 (+5)
	frame inclusion	967	229	664 (+45)	152 (+12)
	framing control	1,676	360	344 (+97)	59 (+21)
CORS	all	-	-	2,254 (+224)	488 (+51)
	with credentials	-	-	179 (+63)	71 (+27)
postMessage		14,045	823	14 (0)	11 (0)
Domain Relaxation		97	61	57	29

Note: *C* and *I* denote cookie confidentiality and integrity. Numbers within parenthesis represent the improvement compared to a web attacker; when missing, the web attacker cannot perform the attack.

increase in the attack surface for frame injection: 45 (+7%) domains are exploitable exclusively by controlling one of the vulnerable subdomains identified in our dataset.

As to the other mechanisms, CORS deployments are significantly more at risk against related-domain attackers rather than against traditional web attackers. In particular, we identify 224 (+11%) new exploitable cases, including 63 (+54%) cases with credentials. Note that the use of CORS with credentials is particularly delicate from a security perspective, hence the strong percentage increase in the number of vulnerable cases is concerning. Domain relaxation, instead, can be abused by related-domain attackers in 57 out of 97 domains (59%) making use of this mechanism. Exploiting domain relaxation puts a related-domain attacker in the same origin of the target web application, hence bypassing all web security boundaries: this is a critical vulnerability, which deserves attention. Domain relaxation is a bad security practice, which should better be avoided in the modern Web. Finally, our analysis of postMessage shows that all sites suffering from unsafe programming practices are already vulnerable against web attackers, i.e., for this specific attack vector related-domain attackers are no more powerful than traditional web attackers, at least based on the collected data. In other words, sites either do not enforce any security check or restrict communication to selected individual origins: this might be a consequence of the postMessage API granting access to origin information, rather than site information directly.

6 Related Work

Related-Domain Attackers. The notion of related-domain attacker was first introduced by Bortz, Barth, and Czeskis [9]. Their work identified the security risks posed by related domains against (session) cookies and proposed a possible solution called *origin cookies*. A similar defense mechanism, i.e.,

the __Host- prefix, was eventually integrated into major web browsers. Other than that, related-domain attackers received only marginal attention from the security community, with a few notable exceptions. Zheng et al. discussed the security implications of the lack of cookie integrity in many top sites, considering both network and related-domain attackers [62]. Calzavara et al. presented black-box testing strategies for web session integrity, including related-domain attackers in their threat model [16]. Related-domain attackers have also been considered in formal web security models, again in the context of web sessions [13]. Our paper significantly advances the understanding of related-domain attackers by discussing new security threats, which go beyond web sessions and have been quantified in the wild through a large-scale measurement.

Attacking Subdomains. Subdomain takeover is an infamous attack, which has been covered by a body of work. Liu et al. [33] studied the threat posed by dangling DNS records, e.g., records that contain aliases to expired domains or pointing to IP addresses hosted on cloud services. The authors performed a large-scale analysis that uncovered the existence of hundreds of dangling records among the subdomains of the top 10k sites of Alexa and under the .edu and .gov zones. With respect to [33], we improved the subdomain enumeration part by a factor of 13 and increased the number of analyzed services from 9 to 31. Also, the paper does not extensively analyze the web security implications of subdomain takeover. Borgolte et al. [8] improved on the results of [33] concerning deprovisioned cloud instances and proposed an extension of the ACME protocol used by some CAs for domain validation (e.g., *Let's Encrypt*). Schwittman et al. [48] studied these domain validation techniques and discovered several vulnerabilities that could be exploited by attackers to obtain valid certificates for domains they do not own.

Liu et al. [34] proposed a technique to detect shadowed domains used in malware distribution campaigns, i.e., legitimate domains that are compromised to spawn an arbitrary number of subdomains after taking control of the DNS configuration panel at the registrar. Alowaisheq et al. [5] recently demonstrated a domain hijacking attack that relies on the exploitation of stale NS records. Zhang et al. [61] showed how a domain with HTTPS misconfigurations can be abused by a network attacker to force the communication over HTTP with its related domains. However, the authors consider two domains as related if they share the same TLS certificate, which differs from the definition considered in this work. A large body of works studied the problem of domain impersonation (e.g., [4, 28, 45]) where attackers trick users to interact with their malicious websites by using domain names that mimic those of honest sites. An example is provided by doppelganger domains [60] which are spelled similarly to legitimate subdomain names except for the dots that separate the components of the domain name. We consider all these threats out of the scope of our analysis, as they have different security implications than the vulnerabilities we discuss.

Web Measurements. Meiser et al. [35] studied the cross-origin data exchange practices of 5k websites to assess to which extent their security could be affected by the presence of an XSS vulnerability on one of their communication partners. In our work, we study a similar problem, but we restrict our focus to related domains, and we consider other mechanisms that are out of scope for [35], e.g., CSP. Chen et al. [18] performed a large-scale measurement of CORS misconfigurations. Among the 480k domains that they analyzed, they discovered that 27.5% of them are affected by some vulnerability and, in particular, 84k trust all their subdomains and can thus be exploited by a related-domain attacker. Son and Shmatikov [50] analyzed the usage of the Messaging API on the top 10k Alexa websites. The authors found that 1.5k hosts do not perform any origin checking on the receiving message, while 261 implement an incorrect check: (almost) all these checks can be bypassed from a related-domain position, although half of them can also be bypassed from domains with a specially-crafted name. More recently, Steffens and Stock [51] proposed an automated framework for the analysis of postMessage handlers and used it to perform a comprehensive analysis of the first top 100k websites of the Tranco list. The authors discovered 111 vulnerable handlers, out of which 80 do not perform any origin check. Regarding the remaining handlers, the authors identified only 8 incorrect origin validations, showing an opposite trend with respect to [50]. Finally, insecure configurations of CSP have been analyzed in a number of research papers [15, 46, 57, 58]. However, none of these works considered the problem of related-domain attacks.

7 Conclusion

In this paper, we presented the first analysis tailored at quantifying the threats posed by related-domain attackers to the security of web applications. We first introduced a novel framework that captures the capabilities acquired by such attackers, according to the position in which they operate, and we discuss which web attacks can be launched from that privileged position, highlighting the advantages with respect to traditional web attackers. We also studied the security implications of 31 third-party service providers and dynamic DNS to identify the capabilities that a related-domain attacker acquires when taking over a domain hosted by them, and presented a novel subdomain hijacking technique that resulted in a bug bounty of $1,000. Then, we described the design of our automated toolchain used to assess the pervasiveness of these threats in the wild. The toolchain consists of an analysis module for subdomain takeover that identifies which subdomains can be hijacked by an attacker. Next, the web security module quantifies how many related domains can be attacked from the domains discovered in the previous step. We performed a large-scale analysis on the 50k most popular domains, and we identified vulnerabilities in 887 of them, including major websites like cnn.com and cisco.com. Then, we correlated

for the first time the impact of these vulnerabilities on the security of web applications, showing that related-domain attackers have an additional gain compared to web attackers that goes beyond the traditional cookie issues.

Acknowledgments

We thank the anonymous reviewers for their helpful suggestions. We also thank Google for sponsoring our research with $5,000 in credits for Google Cloud Platform and Cisco Talos for granting us access to a dataset that was used during a preliminary investigation for this project. This work has been partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 research (grant agreement 771527-BROWSEC); by the Austrian Science Fund (FWF) through the project PROFET (grant agreement P31621); by the Austrian Research Promotion Agency (FFG) through the Bridge-1 project PR4DLT (grant agreement 13808694) and the COMET K1 SBA.

References

[1] Puppeteer. https://pptr.dev/, 2020.

[2] J. Aas, R. Barnes, B. Case, Z. Durumeric, P. Eckersley, A. Flores-López, J. A. Halderman, J. Hoffman-Andrews, J. Kasten, E. Rescorla, S. Schoen, and B. Warren. Let's Encrypt: An Automated Certificate Authority to Encrypt the Entire Web. In *CCS*, 2019.

[3] Abusix. Abuse Contact Database. https://www.abusix.com/contactdb, 2020.

[4] P. Agten, W. Joosen, F. Piessens, and N. Nikiforakis. Seven Months' Worth of Mistakes: A Longitudinal Study of Typosquatting Abuse. In *NDSS*, 2015.

[5] E. Alowaisheq, S. Tang, Z. Wang, F. Alharbi, X. Liao, and X. Wang. Zombie Awakening: Stealthy Hijacking of Active Domains Through DNS Hosting Referral. In *CCS*, 2020.

[6] A. Barth, C. Jackson, and J. C. Mitchell. Robust Defenses for Cross-Site Request Forgery. In *CCS*, 2008.

[7] N. Biasini. Threat Spotlight: Angler Lurking in the Domain Shadows. http://blogs.cisco.com/security/talos/angler-domain-shadowing, 2015.

[8] K. Borgolte, T. Fiebig, S. Hao, C. Kruegel, and G. Vigna. Cloud Strife: Mitigating the Security Risks of Domain-Validated Certificates. In *NDSS*, 2018.

[9] A. Bortz, A. Barth, and A. Czeskis. Origin Cookies: Session Integrity for Web Applications. In *W2SP*, 2011.

[10] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Comput. Networks*, 1998.

[11] Bugcrowd. Public Bug Bounty List. https://www.bugcrowd.com/bug-bounty-list/, 2020.

[12] M. Bugliesi, S. Calzavara, R. Focardi, and W. Khan. CookiExt: Patching the Browser Against Session Hijacking Attacks. *Journal of Computer Security*, 23(4):509–537, 2015.

[13] S. Calzavara, R. Focardi, N. Grimm, M. Maffei, and M. Tempesta. Language-Based Web Session Integrity. In *CSF*, 2020.

[14] S. Calzavara, R. Focardi, M. Squarcina, and M. Tempesta. Surviving the Web: A Journey into Web Session Security. *ACM Computing Surveys (CSUR)*, 50(1):13:1–13:34, 2017.

[15] S. Calzavara, A. Rabitti, and M. Bugliesi. Semantics-Based Analysis of Content Security Policy Deployment. *ACM Transactions on the Web*, 2018.

[16] S. Calzavara, A. Rabitti, A. Ragazzo, and M. Bugliesi. Testing for Integrity Flaws in Web Sessions. In *ESORICS*, 2019.

[17] Censys. https://censys.io/, 2020.

[18] J. Chen, J. Jiang, H. Duan, T. Wan, S. Chen, V. Paxson, and M. Yang. We Still Don't Have Secure Cross-Domain Requests: an Empirical Study of CORS. In *USENIX Security*, 2018.

[19] Common Crawl. Host- and Domain-Level Web Graphs Feb/Mar/May 2020. https://commoncrawl.org/2020/06/host-and-domain-level-web-graphs-febmarmay-2020/, 2020.

[20] M. Conca. Changes to SameSite Cookie Behavior – A Call to Action for Web Developers. https://hacks.mozilla.org/2020/08/changes-to-samesite-cookie-behavior/, 2020.

[21] L. Daigle. RFC3912: WHOIS Protocol Specification, 2004.

[22] EdOverflow. can-i-take-over-xyz. https://github.com/EdOverflow/can-i-take-over-xyz.

[23] E. Foudil and Y. Shafranovich. A File Format to Aid in Security Vulnerability Disclosure, 2020.

[24] FreeDNS. Free DNS Hosting, Dynamic DNS Hosting, Static DNS Hosting, subdomain and domain hosting. https://freedns.afraid.org/, 2020.

[25] P. Hallam-Baker, R. Stradling, and J. Hoffman-Andrews. RFC8659: DNS Certification Authority Authorization (CAA) Resource Record, 2019.

[26] S. Helme. Cross-Site Request Forgery is dead! https://scotthelme.co.uk/csrf-is-dead/, 2017.

[27] IANA. Root Zone Database. https://www.iana.org/domains/root/db.

[28] P. Kintis, N. Miramirkhani, C. Lever, Y. Chen, R. R. Gómez, N. Pitropakis, N. Nikiforakis, and M. Antonakakis. Hiding in Plain Sight: A Longitudinal Study of Combosquatting Abuse. In *CCS*, 2017.

[29] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre Attacks: Exploiting Speculative Execution. In *S&P*, 2019.

[30] C. Lever, R. J. Walls, Y. Nadji, D. Dagon, P. D. McDaniel, and M. Antonakakis. Domain-Z: 28 Registrations Later Measuring the Exploitation of Residual Trust in Domains. In *S&P*, 2016.

[31] E. P. Lewis. RFC4592: The Role of Wildcards in the Domain Name System, 2006.

[32] F. Li, Z. Durumeric, J. Czyz, M. Karami, M. Bailey, D. McCoy, S. Savage, and V. Paxson. You've Got Vulnerability: Exploring Effective Vulnerability Notifications. In *USENIX Security*, 2016.

[33] D. Liu, S. Hao, and H. Wang. All Your DNS Records Point to Us: Understanding the Security Threats of Dangling DNS Records. In *CCS*, 2016.

[34] D. Liu, Z. Li, K. Du, H. Wang, B. Liu, and H. Duan. Don't Let One Rotten Apple Spoil the Whole Barrel: Towards Automated Detection of Shadowed Domains. In *CCS*, 2017.

[35] G. Meiser, P. Laperdix, and B. Stock. Careful Who You Trust: Studying the Pitfalls of Cross-Origin Communication. In *ASIA CCS*, 2021.

[36] P. Mockapetris. RFC1035: Domain Names - Implementation and Specification, 1987.

[37] Mozilla. Mixed content. https://developer.mozilla.org/en-US/docs/Web/Security/Mixed_content.

[38] Mozilla. Public Suffix List. https://publicsuffix.org/.

[39] M. Nottingham. RFC8615: Well-Known Uniform Resource Identifiers (URIs), 2019.

[40] C. Osborne. Uber Patches Security Flaw Leading to Subdomain Takeover. ZDNet, https://www.zdnet.com/article/uber-patches-security-flaw-leading-to-subdomain-takeover/, 2017.

[41] OWASP. Amass. https://owasp.org/www-project-amass/, 2020.

[42] V. L. Pochat, T. V. Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *NDSS*, 2019.

[43] Rapid7 Labs. Open Data, TCP and UDP scans. https://opendata.rapid7.com/, 2020.

[44] C. Reis, A. Moshchuk, and N. Oskov. Site Isolation: Process Separation for Web Sites within the Browser. In *USENIX Security*, 2019.

[45] R. Roberts, Y. Goldschlag, R. Walter, T. Chung, A. Mislove, and D. Levin. You Are Who You Appear to Be: A Longitudinal Study of Domain Impersonation in TLS Certificates. In *CCS*, 2019.

[46] S. Roth, T. Barron, S. Calzavara, N. Nikiforakis, and B. Stock. Complex Security Policy? A Longitudinal Analysis of Deployed Content Security Policies. In *NDSS*, 2020.

[47] S. Röttger and A. Janc. A Spectre proof-of-concept for a Spectre-proof web. https://security.googleblog.com/2021/03/a-spectre-proof-of-concept-for-spectre.html, 2021.

[48] L. Schwittmann, M. Wander, and T. Weis. Domain Impersonation is Feasible: A Study of CA Domain Validation Vulnerabilities. In *EuroS&P*, 2019.

[49] Sectigo. Crt.sh: Certificate search. https://crt.sh/, 2020.

[50] S. Son and V. Shmatikov. The Postman Always Rings Twice: Attacking and Defending postMessage in HTML5 Websites. In *NDSS*, 2013.

[51] M. Steffens and B. Stock. PMForce: Systematically Analyzing postMessage Handlers at Scale. In *CCS*, 2020.

[52] B. Stock, G. Pellegrino, F. Li, M. Backes, and C. Rossow. Didn't You Hear Me? - Towards More Successful Web Vulnerability Notifications. In *NDSS*, 2018.

[53] B. Stock, G. Pellegrino, C. Rossow, M. Johns, and M. Backes. Hey, You Have a Problem: On the Feasibility of Large-Scale Web Vulnerability Notification. In *USENIX Security*, 2016.

[54] The Chromium Projects. SameSite Updates. https://www.chromium.org/updates/same-site, 2020.

[55] W3C. Content Security Policy Level 3. https://www.w3.org/TR/CSP3/, 2018.

[56] J. Walker. Subdomain Autofill Feature Raises Questions over LastPass Security. https://portswigger.net/daily-swig/subdomain-autofill-feature-raises-questions-over-lastpass-security.

[57] L. Weichselbaum, M. Spagnuolo, S. Lekies, and A. Janc. CSP Is Dead, Long Live CSP! On the Insecurity of Whitelists and the Future of Content Security Policy. In *CCS*, 2016.

[58] M. Weissbacher, T. Lauinger, and W. K. Robertson. Why Is CSP Failing? Trends and Challenges in CSP Adoption. In *RAID*, 2014.

[59] M. West and J. Wilander. RFC6265: Cookies: HTTP State Management Mechanism, 2020.

[60] Wired. Researchers' Typosquatting Stole 20 GB of E-Mail From Fortune 500. https://www.wired.com/2011/09/doppelganger-domains/, 2011.

[61] M. Zhang, X. Zheng, K. Shen, Z. Kong, C. Lu, Y. Wang, H. Duan, S. Hao, B. Liu, and M. Yang. Talking with Familiar Strangers: An Empirical Study on HTTPS Context Confusion Attacks. In *CCS*, 2020.

[62] X. Zheng, J. Jiang, J. Liang, H. Duan, S. Chen, T. Wan, and N. Weaver. Cookies Lack Integrity: Real-World Implications. In *USENIX Security*, 2015.

A Disclosure and Ethical Considerations

RDScan identified 1,520 vulnerable subdomains on 887 distinct domains, of which 260 are subdomains pointing to an expired domain and 1,260 are those mapped to a discontinued service (see §4.2). Besides disclosing the vulnerabilities found on service providers (§5.1.2), we also attempted to notify all the websites affected by the issues we discovered. Prior work [32, 52, 53] showed that the identification and selection of correct security contact points is the main issue behind an overall unsatisfactory remediation rate. To maximize the chances of a successful notification campaign, we examined the following sources until a valid point of contact was found: (i) the list of bug bounty and security disclosure programs maintained by Bugcrowd [11]; (ii) the `security.txt` file [23] in the root directory of the vulnerable domains and under the `/.well-known/` folder [39]; (iii) the Abusix [3] database, queried with the ip addresses of the vulnerable domains to collect the associated email contacts; (iv) a WHOIS lookups [21]. We validated the obtained email addresses to avoid reporting vulnerabilities to unrelated parties, e.g., by checking whether the domain part of the email address matches any of the input domains. Unfortunately, using this procedure we could not find any security contact for the majority of the considered domains (62%). To inform them about their security vulnerabilities, we contacted our national CERT that willingly agreed to disclose the issues to the affected parties on our behalf. Among the few contacted websites with a bug bounty program, F-Secure awarded us with €250 for the reported subdomain takeover vulnerability.

Aside from vulnerability disclosure programs, our notification campaign is fully automatic: we sent an email to all the identified contacts containing a high-level description of the vulnerabilities and a link to the security advisory on our web application which contains a detailed description of the problems found for a given domain, the required actions to fix the reported vulnerabilities, and instructions to opt out from future scans.

A.1 Outcome of the Notification Campaign

We performed a second run of RDScan on January 2021, 6 months after the first analysis, to picture the state of vulnerable instances left in the wild after our disclosure. We repeated the test for the whole set of expired domains instances. Concerning discontinued services, we focused on the 3 largest providers (WordPress, Shopify and Tumblr), representing 87% of the vulnerable subdomains found in the first round. Overall, we covered 1362 out of the original 1520 vulnerable subdomains (90%), which translates to 781 out of 887 sites (88%). To account for possible changes in services occurred in the meanwhile, we verified the takeover preconditions included in RDScan. After the conclusion of the analysis, we manually assessed a random sample of 10% of the results to ensure the correctness of the procedure without finding any discrepancy.

We discovered that only 200 out of 1362 subdomains (15%) have been fixed during this time frame, for a total of 125 sites over 781 (16%). We noticed that the sites which we contacted directly exhibit a noticeably higher fix rate (31% subdomain, 22% sites) than those alerted by our national CERT (10% subdomains, 14% sites). Unfortunately, we also observed that a considerable amount of sites fixed only a subset of their vulnerable subdomains, resulting still affected by threats posed by related-domain attackers.

The overall remediation rate of our notification campaign is in line with previous studies [53]. Nonetheless, we report that our procedure to identify appropriate contact points turned out to be successful considering that 34% of the contacted parties accessed the full vulnerability report on our web application.

A.2 Ethical Considerations

We consciously designed our vulnerability scanning framework to avoid raising network alerts or causing harm to the

analyzed targets. Specifically, the subdomain takeover assessment phase has been carried out mostly by DNS queries and simple HTTP requests. Active websites have never been affected by our tests since we restricted the analysis to abandoned DNS records. We did not perform any large-scale portscan, but we opted, instead, for using a public dataset consisting of a scan of the full IPv4 range on 148 ports.

We also avoided checking the availability of IP addresses on cloud providers by iterating over the creation of multiple virtual machines, since this practice could interfere with the normal operations of the cloud platforms. Similarly, the web analysis module did not execute attacks against the targets, but limited its operations to the passive collection of data (cookies and security policies), simple HTTP requests, and client-side testing. Overall, our approach proved to be lightweight and unobtrusive: we did not receive requests from the analyzed websites to opt out from future scans, and no complaints concerning our activity were sent to the abuse contact of the IPs used to perform the analysis.

B Case Studies

We report on manually vetted case studies of confirmed attacks. All vulnerable parties have been promptly informed of the discovered issues, see Appendix A for details.

B.1 Site Impersonation

We provide a concrete example of how the Shopify vulnerability described in §5.1.2 could have been abused to impersonate a major website. As of September 2020, the e-shop of fox.com was hosted on Shopify and made available at `shop.fox.com` using a custom domain mapping. Our scan verified the two preconditions to connect `www.shop.fox.com` to a Shopify store under our control, i.e., the existence of a DNS `A` record pointing the domain `www.shop.fox.com` to 23.227.38.65 (the IP address owned by Shopify to map custom domains) and that `www.shop.fox.com` was not associated with any registered store on Shopify.

We manually investigated the e-shop of fox.com and found that the redirection performed by Shopify from `www.shop.fox.com` to `shop.fox.com` caused the www-prefixed subdomain to be referenced in the store as a legitimate URL.[3] By taking over `www.shop.fox.com`, criminals could have abused this implicit trust to mount severe attacks against the legitimate store, such as phishing, reputation damage, and credential stealing. We notified the vulnerability to Shopify on August 27, 2020 and received a bounty for our disclosure. Around one month after the report, we noticed that FOX moved its e-shop to a different domain (`maskedsingershop.com`). We have no evidence to assert whether this change is connected to our disclosure.

[3] See https://web.archive.org/web/20200113052608/https://shop.fox.com/pages/faq for a page mentioning `www.shop.fox.com`.

B.2 Session Hijacking

We describe an example of a subdomain takeover vulnerability that could have been exploited to hijack authenticated user sessions at the FedEx website. RDScan discovered a dangling DNS affecting the `cn.grantcontest.fedex.com` subdomain due to a `CNAME` record pointing to the purchasable domain `cngrantcontest.com`.

After taking control of the subdomain, attackers could escalate their privileges by exploiting the insecure configuration of session cookies on the main website. We manually verified that authenticated sessions with `www.fedex.com` were built upon domain cookies, which are sent by default to all subdomains (see §3.3.2). Thus, authenticated users would disclose their session cookies to the attackers just by visiting the compromised subdomain. After acquiring the victim's cookies, an attacker could automatically break into the victim's session and access confidential data stored on the web portal. We notified FedEx about the takeover vulnerability in August 2020. The company acknowledged our findings and, as of January 2021, we confirmed that the vulnerability was fixed.

B.3 Leakage of PII data

Now we show how a related-domain attacker can abuse misconfigurations in the CORS policy to access personally identifiable information (PII) of a user on the F-Secure website. Our vulnerability scanning pipeline detected a `CNAME` record `uk.safeandsavvy.f-secure.com` pointing to the deleted WordPress blog at `safeandsavvyuk.wordpress.com`. Notice that subdomains of deleted blogs still resolve to a WordPress IP thanks to a `CNAME` wildcard for `*.wordpress.com`. To take over the F-Secure subdomain, an attacker could simply create an account on wordpress.com and set `uk.safeandsavvy.f-secure.com` as a custom domain.

We observed that WordPress allows paid accounts to install plugins which enable the inclusion of arbitrary scripts as part of the blog's theme. The ability to execute JavaScript from a subdomain of `f-secure.com` would allow attackers to exploit a CORS vulnerability identified by our web analyzer on `api.my.f-secure.com`. Such domain was configured to relax the SOP on requests originating from any subdomain of `f-secure.com`, even when cookies are attached. An attacker could trick a victim into visiting a page on the compromised subdomain which performs a `fetch` request to, e.g., the `https://api.my.f-secure.com/get_userinfo` endpoint to read private information such as past billing details, tokens, etc. We notified F-Secure through their bug bounty program in August 2020 and received €250 for the report.