Network-based Origin Confusion Attacks against HTTPS Virtual Hosting

Antoine Delignat-Lavaud
Inria Paris-Rocquencourt
antoine@delignat-lavaud.fr

Karthikeyan Bhargavan
Inria Paris-Rocquencourt
karthikeyan.bhargavan@inria.fr

ABSTRACT

We investigate current deployment practices for virtual hosting, a widely used method for serving multiple HTTP and HTTPS origins from the same server, in popular content delivery networks, cloud-hosting infrastructures, and web servers. Our study uncovers a new class of HTTPS origin confusion attacks: when two virtual hosts use the same TLS certificate, or share a TLS session cache or ticket encryption key, a network attacker may cause a page from one of them to be loaded under the other's origin in a client browser. These attacks appear when HTTPS servers are configured to allow virtual host fallback from a client-requested, secure origin to some other unexpected, less-secure origin. We present evidence that such vulnerable virtual host configurations are widespread, even on the most popular and security-scrutinized websites, thus allowing a network adversary to hijack pages, or steal secure cookies and single sign-on tokens. To prevent our virtual host confusion attacks and recover the isolation guarantees that are commonly assumed in shared hosting environments, we propose fixes to web server software and advocate conservative configuration guidelines for the composition of HTTP with TLS.

1. INTRODUCTION

Web applications are increasingly being moved to the cloud or deployed on *distributed content delivery networks* (CDNs), raising new concerns about their security. The cloud environment requires the sharing of servers and network addresses between many unrelated, mutually distrusting principals. On the client side, the problem of securely isolating websites from each other within the browser has been a core topic of security research in recent years, producing a rich literature centered around the notion of *security origin*. Yet, on the server side, the security implications of hosting large numbers of websites from the same web servers has gathered relatively little attention, even though cloud infrastructures constitute a prime target for attacks, both from criminals and from governmental adversaries [1].

WWW 2015, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3469-3/15/05.
http://dx.doi.org/10.1145/2736277.2741089.

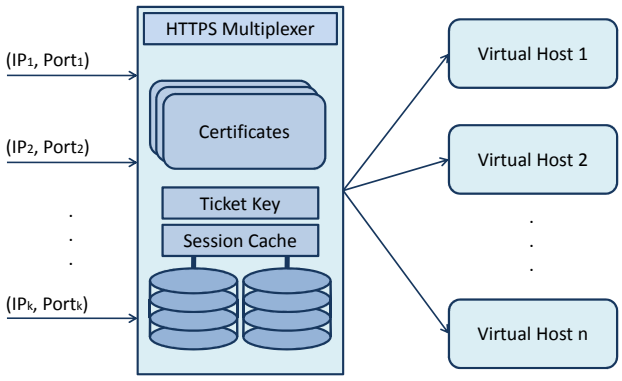

Figure 1: HTTPS server with multiple virtual hosts

The Transport Layer Security (TLS) protocol [2], as used within HTTP over TLS (HTTPS) [3], remains the only defense against network-layer attacks on the web. It provides authentication of the server (and optionally, of the client), as well as confidentiality and integrity of HTTP requests and responses, against attackers that control both the network and malicious websites visited by the client.

While the precise security guarantees of TLS have been extensively studied [4–6], these formal works all consider a simple deployment model, where each server only has one network interface and one certificate valid for a single domain that matches the server identity. This model does not reflect current practices, especially in the cloud, but also in many mainstream web servers.

Sharing TLS Server Credentials Many web servers employ *virtual hosting* to serve multiple HTTPS domains behind the same TLS server. To do this, the TLS server needs to decide which certificate to present to an incoming connection. This decision is either based on the incoming IP address, or increasingly often, on the server identity requested within the TLS *server name indication* (SNI) extension [7]. Even when different domains use different certificates, by using the same TLS server, they often implicitly share the TLS session cache that is used for fast session resumption.

Moreover, the same certificate may be used across multiple domains on different servers. Recent measurement studies of TLS certificate issuance [8, 9] show that a majority of newly issued certificates are valid for more than one domain name, with a significant number of them containing at least one wildcard. For example, all the front-end Google servers

share a certificate that covers `*.google.com` as well as 50 other DNS names, many with wildcards.

Finally, the same certificate may be used on multiple ports on the same domain. For example, web servers often listen for HTTP-based protocols such as WebSocket [10] on non-traditional ports, but reuse the same IP address, domain name, and TLS certificate as the main website.

When TLS credentials are shared between different HTTP server entities, how do the security guarantees provided by TLS relate to those desired by HTTPS? In this paper, we investigate this question with regard to the origin-based security policies commonly used in modern web applications.

Same Origin Policy Web browsers identify resources by the *origin* from which they were loaded, where an origin consists of the *protocol, domain name* and *port number*, e.g. `https://y.x.com:443`. The *same-origin policy* [11] allows arbitrary sharing between pages on the same origin, but strictly regulates cross-origin interactions. Hence, if any page on a given origin is compromised, either by a cross-site scripting (XSS) flaw [12], or because the server is under attacker control, the whole origin must be considered compromised as well. Consequently, prudent websites divide their content into different subdomains at different security levels, so that the compromise of one (e.g. `blog.x.com`) does not affect another (e.g. `login.x.com`).

In the presence of a network attacker, the same origin policy only protects HTTPS origins, for which the underlying TLS protocol can guarantee that the browser is connected to a server that owns a certificate for the desired origin. However, when TLS server credentials, such as certificates and cached sessions, are shared across servers, the isolation guarantees of the same origin policy crucially rely on the routing of HTTP requests to the correct origin server.

Routing Requests to Virtual Hosts The server-side counterpart of the notion of *origin* is the *virtual host*, whose role is to produce the response to an HTTP request given its *path* and *query parameters* (i.e. what appears in the URL after the origin). Virtual hosts used to correspond to directories on the server's filesystem, but with the widespread use of rewriting rules and dynamically generated pages, virtual hosts are now best treated as abstract request processors.

Figure 1 depicts the process that a web server uses to choose a virtual host for a given HTTPS request. The decision depend on parameters gathered at various levels: the IP address and port that the TCP connection was accepted on, the SNI extension received during the TLS handshake, and the `Host` header received in the HTTP request. On the client, all these parameters are derived from the request URL (a DNS request yields the IP address). On the server, each parameter is considered separately in a manually configured set of complex rules to determine the virtual host that will handle the request (see Section 3 for more detail).

In particular, most web servers will pick a *fallback* virtual host when the normal routing rules fail. In plain HTTP, routing fallback can be quite useful, for instance to access a website by its IP address or former domain name, or to use the same request handler for all subdomains. However, HTTPS routing fallback can be extremely dangerous, since it may allow a request for a client-requested secure origin to be processed by the virtual host for some unexpected, less-secure origin.

Virtual Confusion Attacks The main contribution of this paper is the identification of a new class of attacks on virtual hosts that share TLS credentials, either on the same or on different web servers. In these attacks, a network attacker can take an HTTPS connection meant for one of these virtual hosts and redirect it to the other. The TLS connection succeeds because of the shared TLS credentials; then, because of virtual host fallback, the request is processed by a virtual host that was never intended to serve contents for the domain in the `Host` header.

In particular, we show that a network attacker can always break the same-origin policy between different ports on the same domain, by redirecting connections from one port to another. Moreover, if two servers serving two independent domains share a common certificate (covering both domains), or a cached TLS session, the network attacker can cause pages from one server to be loaded under the other's origin. In all these cases, the attacker subverts the browser's intended origin of the request, often with exploitable results.

Concrete Website Exploits Origin confusion attacks between two HTTPS domains are particularly dangerous when one of them is less secure than the other, for example, if one has an XSS flaw or an insecure redirection. We detail five exemplary instances of origin confusion attacks that demonstrate different attack vectors and illustrate the applicability and impact of this class of attacks:

1. We show how HTTPS requests to many websites hosted by the Akamai CDN can be hijacked by a server controlled by an attacker (Section 2).
2. We show how single sign-on access tokens on Yahoo (and several other major websites) can be stolen by exploiting an unsafe redirector on Yahoo (Section 4.1).
3. We describe a combined network- and web-based XSS attack on Dropbox that exploits malicious hosted content and cookie forcing (Section 4.2).
4. We show how HTTPS requests to highly-trusted Mozilla websites such as `bugzilla.mozilla.org` can be redirected to user-controlled pages on `git.mozilla.org`, by exploiting shared TLS session caches (Section 4.3).
5. We show how TLS session reuse in the SPDY protocol [13] can be exploited to impersonate any HTTPS website in Chrome (Sections 6).

These attacks were responsibly disclosed, acknowledged, and fixed in the relevant websites, CDNs, browsers, and web servers. They have been awarded bug bounties by HackerOne, Chromium, and Mozilla. More worryingly, the attacks show the dangerous consequences of seemingly innocuous routing decisions within widely used web servers, TLS terminators, and reverse proxies. Section 7 discusses some countermeasures, and we hope this paper will trigger a more systematic study of server-side HTTPS multiplexing, and research on new robust designs that can prevent such attacks.

2. IMPERSONATING WEBSITES SERVED BY THE AKAMAI CDN

Akamai is the leading content delivery network (CDN) on the web, claiming to be responsible for up to 20% of the total Internet traffic [14]. Like other CDNs, Akamai has a large network of *points of presence* (PoP) distributed all around

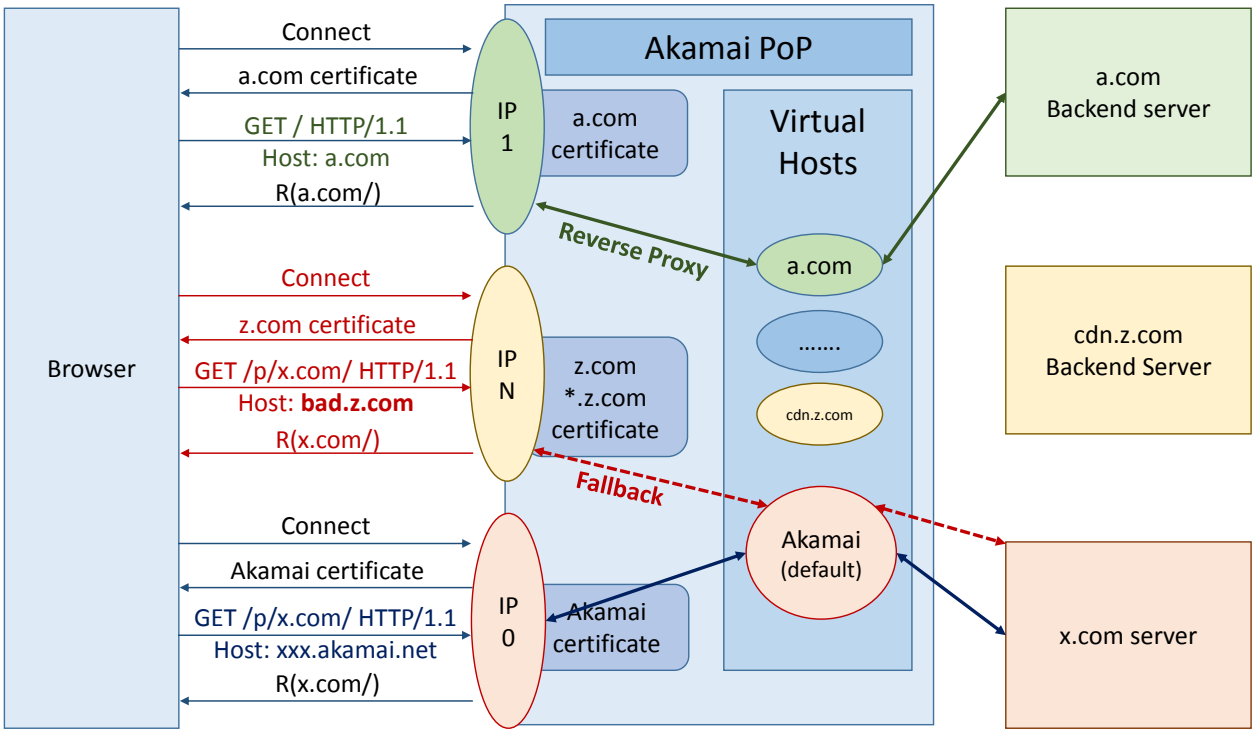

Figure 2: Akamai Point-of-Presence (P0P) server design

the world, whose job is to cache static contents from the websites of Akamai customers, to reduce latency and distribute load. Akamai serves varied customers, including popular social networks like `linkedin.com` and sensitive websites like `nsa.gov` that are often accessed over HTTPS. We will see how virtual host fallback on Akamai's PoPs leads to a serious origin confusion attack on such websites.

CDNs use one of two strategies to deploy HTTPS for customer websites; an extensive survey of real-world practices appears in [15]. Some CDNs (e.g. CloudFlare) use *shared certificates* that are fully managed by the CDN operator with no involvement from its customer. Shared certificates are valid for a large number of customer domains and may be deployed on all the PoPs of the CDN; their private keys remain under the CDN provider's control. Other CDNs (e.g. Akamai) require customers to obtain *custom certificates* for their HTTPS domains from certification authorities. The CDN must be given access to the private keys of these certificates, so that they can be installed on the PoPs allocated to the customer. On a PoP with custom certificates, the choice of server certificate on a TLS connection may depend on the incoming IP address or on the server name in the TLS SNI extension. CDNs increasingly prefer SNI, but it is not available on some legacy clients (e.g. Windows XP).

Virtual Host Fallback in Akamai PoPs The Akamai CDN uses a uniform virtual host configuration on its PoPs, all of which run a custom HTTP server implementation called "AkamaiGhost". Figure 2 depicts how HTTPS requests are processed by Akamai: each PoP has `N` custom certificates installed for `N` virtual hosts, and each certificate is served on a dedicated IP address. Therefore, if a client connects to IP 1, it will be given the certificate for `a.com`, whereas if it connects to IP 2, it will be given the certificate for `*.z.com`. After the TLS connection is complete, the PoP inspects the HTTP `Host` header and routes the request to the appropriate virtual host.

Each PoP also serves a special Akamai virtual host, which is also the fallback default. Hence, if the `Host` header of a request (received on any IP address) isn't one of the `N` configured customer domains, it is routed to this default host. Interestingly, the Akamai virtual host acts as a universal proxy: when a request for `/p/a.com/path` is received, for a certain well-known prefix `p`, the PoP forwards the request to `a.com/path`, along with all HTTP headers sent by the client (including cookies and HTTP authentication). Then, it caches and forward the response from `a.com` to the client. Providing an open proxy for HTTP connections to any website is a perfectly reasonable design decision and may even be considered a generous gesture[1]. Unfortunately, the impact of this proxy on HTTPS connections to customer domains is severe.

Server Impersonation Attack We now consider a concrete example. LinkedIn uses Akamai only for the domain `static.linkedin.com`, but the certificate it provides to Akamai is valid for `*.linkedin.com`. Suppose a user is logged in to LinkedIn from her browser. The attack (shown for `bad.z.com` in Figure 2) proceeds as follows:

1. A network attacker gets the browser to visit: `https:`

[1] `http://www.peacefire.org/bypass/Proxy/akamai.html`

Figure 3: Outcome of the attack against nsa.gov

`//www.linkedin.com/p/attacker.com/` by injecting JavaScript on some HTTP page loaded by the browser.

2. The attacker redirects the resulting TLS connection to the LinkedIn IP address on some Akamai PoP.
3. The TLS connection succeeds since the certificate returned from the PoP is valid for `*.linkedin.com`.
4. The PoP only has a virtual host configured for `static.linkedin.com`; hence, the request falls back to the Akamai virtual host, which triggers the open proxy to `attacker.com`.
5. The user's browser loads the attacker's website under the `https://www.linkedin.com` origin (no certificate warning). It also sends the user's `Secure, HttpOnly` LinkedIn cookies to the attacker.

This is an instance of an *origin confusion attack* that leads to full server impersonation. It defeats all existing HTTPS protections: it leaks all cookies, it allows the attacker to disable HSTS and content security policy. Worse, it does not leave any trace on the impersonated server (which is never involved during the attack). In the PoP's HTTP log, the request looks like a harmless caching query to the proxy.

Responsible Disclosure This critical flaw existed in Akamai servers for nearly 15 years without getting noticed. Based on domains in the Alexa list, we estimate that at least 12,000 websites have been vulnerable to this attack, including 7 out of the top 10 websites in the USA. For example, Figure 3 depicts the server impersonation attack on the `nsa.gov` domain. Following our report, Akamai changed its default virtual host to one that only returns an error page.

3. MULTIPLEXING HTTPS CONNECTIONS

In this section, we investigate how real-world HTTPS implementations decide which certificate and virtual host to use when processing an incoming request. This problem applies to all popular web servers such as Apache, Nginx or IIS, but also to *SSL terminators*, CDN frontend servers and other reverse proxy software.

Virtual Host Parameters There are three layers of identity involved in the processing of HTTPS request: the network layer identity corresponds to an IP address and port; the transport/session layer identity consists of a server certificate and TLS session database and/or ticket encryption key; lastly, the application layer identity is conveyed in the

```
ssl_session_ticket_key "/etc/ssl/ticket.key";
ssl_session_cache shared:SSL:1m;

server { #1
        listen 1.2.3.4:443 ssl;
        server_name www.a.com;
        ssl_certificate "/etc/ssl/a.pem";
        root "/srv/a";
}
server { #2
        listen 4.3.2.1:443 ssl;
        server_name ~^(?<sub>api|dev)\.a\.com$;
        ssl_certificate "/etc/ssl/a.pem";
        root "/srv/api";
}
server { #3
        listen 2.1.4.3:443 ssl;
        server_name www.learn-a.com;
        ssl_certificate "/etc/ssl/learn-a.pem";
        root "/srv/learn";
}
```

Figure 4: Sample virtual host configuration

`Host` header of HTTP requests (however, there is no equivalent header in responses, which are origin-unaware).

Concretely, each web server implements some *multiplexing* logic based on a configuration file that defines how to route an incoming HTTPS connection to the right virtual host. While each server software has its own configuration syntax, there is a common set of parameters that are used to define new TLS-enabled virtual hosts:

1. A *listen* directive that specifies at least one pair of IP address and port number on which the virtual host accepts connections. It is possible to use a wildcard in the IP address to accept connections to any address, whereas a port must be specified.
2. A *server name* directive that may contain one or more fully qualified domain names or regular expressions defining a class of domain names. Without loss of generality, we assume that the server name is always given as a single regular expression.
3. A *certificate* directive which points to the certificate and private key to use for this virtual host.
4. A *session cache* directive, that optionally describes how to store the data structures for session identifier based resumption, either in memory, or on a hard drive or external device. This directive may also specify the encryption key for ticket-based resumption.

If any of the last three items is not defined in the configuration of the virtual host, its value is typically inherited from the server-wide configuration settings, if available. Figure 4 shows an example virtual host configuration for Nginx.

Request Routing The process of selecting the virtual host to use for a given incoming connection can be broken up as follows (see [16, 17] for implementation-specific references):

1. First, the server initializes the list of candidates with every virtual host defined in the configuration.
2. Then, the server inspects the IP address and port on which the client connected. Virtual hosts defined on a different IP address (save for wildcards) or port are removed from the list of candidates.

3. The server next inspects the TLS handshake message sent by the client.
   (a) if the client hello message does not include the SNI extension, the server will return the certificate configured in the virtual host that has been marked as default for the given IP address and port, or if no default is defined, in the first one;
   (b) if an SNI value is specified, the server returns the certificate from the first virtual host whose server name matches the given SNI. If no server name matches, once again, the certificate from the default host is used.
4. Next, the web server finishes the handshake and waits for the client to send its request to inspect the `Host` HTTP header. If it includes a port number, it is immediately discarded. Then, the server picks either the first virtual host from the candidate list whose server name matches the HTTP `Host`. If none matches, it picks either the default virtual host, if one is defined, or the first host from the candidate list otherwise.

There are multiple problems with this virtual host selection process: for instance, it may allow the server to pick TLS settings (including certificate and session cache) from one virtual host, but route the incoming request to a different one (this behavior may be justified by the SPDY optimization described in Section 6).

Port Routing Even though the requested port is included in the `Host` header, and thus reflects the actual port that the browser will use to enforce the same-origin policy, is ignored by all the implementations we tested in favor of the port the connection was received on, which is unauthenticated. This means that it is *always* possible for an attacker to redirect requests from one port to another, and confuse the two origins. Because of this observation, *we strongly recommend to remove the port number from the same-origin policy*, considering that cross-port origin isolation simply does not work in practice (it is already known not to work with cookies).

Fallback Most dangerously, fallback mechanisms open a wide range of unexpected behaviors, and they often depend on the order in which the virtual hosts have been written in the configuration file. The configuration in Figure 4 includes one of the most widespread vulnerable patterns. A certificate valid for two subdomains of `a.com` is used in virtual hosts on different IP addresses (possibly on different physical machines). If an attacker intercepts a connection to `www.a.com` and redirects it to `4.3.2.1:443`, a page from `api.a.com` will be loaded under the `www.a.com` origin, because the host selected during routing must match the IP address and port of the connection.

TLS Session Cache Similarly, the TLS session caching behavior appears to have serious pitfalls in several popular web servers (unlike the request processing algorithm, session caching mechanisms can significantly differ between implementations). For instance, in Nginx:

- By default, only ticket-based session caching is enabled. If no ticket key has been configured, a fresh one is generated for each IP address and port (but not for each virtual host). On the other hand, if a ticket key is specified in the global configuration of the server, all tickets created by any virtual host can be resumed on any other. If a ticket key is given in the configuration of a given virtual host, it will also replace the key on all previously defined hosts on the same IP address.
- Session identifier-based resumption must be explicitly enabled by configuring a session cache database on the server. In-memory shared caches (shared in the sense of threads), which carry an identifier, are commonly used. Sessions from all virtual hosts that use the same identifier in their shared cache can be resumed on each other, regardless of IP address, SNI or certificate.

Once again, it is easy to mis-configure a server to allow sessions to be resumed across virtual hosts. For instance, the configuration in Figure 4 has a global ticket key: if the user has a TLS session created with `www.a.com`, a resumption attempt can be redirected by the attacker to `4.3.2.1:443`: the TLS session ticket will be accepted but because of fallback, a page from `www.learn-a.com` gets loaded under the wrong origin, even thought they don't use the same certificate. Such an attack is enabled by the lack of authentication during the abbreviated TLS handshake: indeed, resumption is purely based on the session identifier or session ticket, regardless of the original SNI or server certificate.

In the next section, we demonstrate various classes of exploit that rely on virtual host confusion, illustrated by concrete attacks against popular websites.

4. ORIGIN CONFUSION EXPLOITS

In itself, virtual host confusion does not sound like a big problem: fundamentally, it only allows a network attacker to load a page under an unexpected, but related (in certificate or session cache), origin. The interesting question is, what can an attacker do with this capability? We found a variety of possible exploits that always follow the same pattern: the loaded page contains bad HTTP characteristics that can break the security of the confused origin.

In the case of Akamai, the loaded page was under complete attacker control. We found similar cases where the loaded page is controlled by the adversary. However, weaker forms of control are a lot more common, but can be still exploited. For instance, if the page sets the `X-Frame-Options` header to `allow`, the confused origin can be loaded in an `iframe`, even though the confused origin might have relied on that header to block clickjacking. Similarly, the origin may have relied on the `Content-Security-Policy` [18] header to block the execution of injected inline scripts, but this can be broken if the loaded page contains a more relaxed CSP. In the rest of this section, we present three more exploits that rely on more creative uses of a network attacker's capabilities.

4.1 Cross-Protocol Redirections: OAuth

The first class of exploits relies on the observation that many websites only use HTTPS on the security-critical parts of their website (for instance, the login form). If, on a low-security virtual host, there exists a page that redirects either to plain HTTP, or to an arbitrary page on another origin (open redirector), then, by confusing a request on a high trust virtual host to such a page, an attacker may learn some secret parameters from the query string or URL fragment by intercepting the redirection.

The prime candidate for this type of exploit is single sign-on access tokens, used by Facebook, Twitter, Google or Yahoo on a large proportion of websites as a replacement for

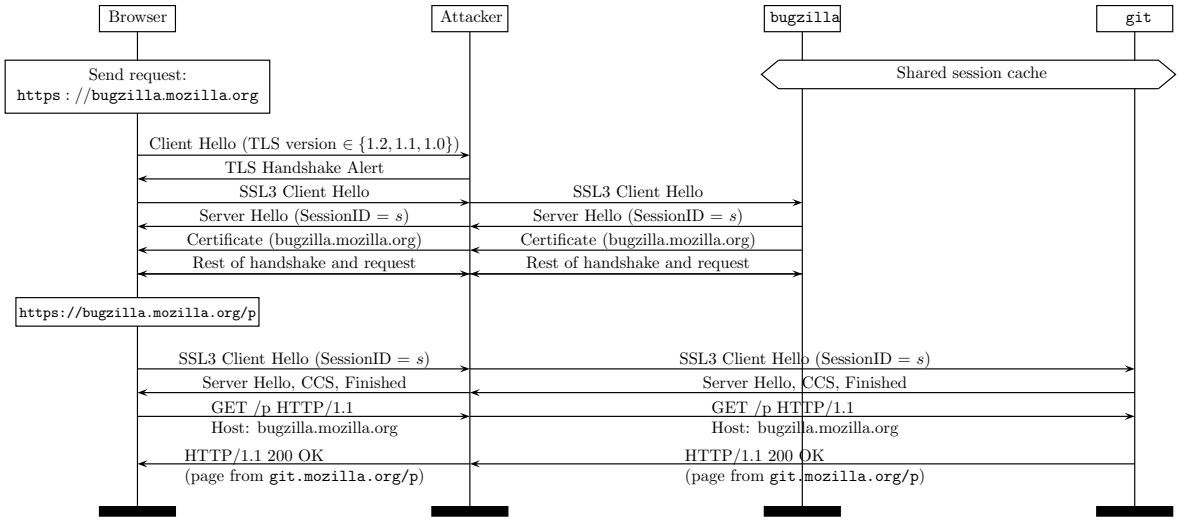

Figure 5: Session cache sharing attack against two Mozilla servers

login forms. For instance, in the OAuth 2.0 protocol [19], a client website registers its origin with the identity provider (e.g. Google), and can obtain an access token to access the user credentials by sending the user to the authorization page on the identity provider's website. This request includes a redirection URL on the registered high-trust origin of the client website. The access token is included in the redirection response in the URL fragment.

Assume `X=https://oauth.a.com` is the registered OAuth origin, served by a virtual host that can be confused with the one for `https://www.a.com` (e.g. because they share a wildcard certificate for `*.a.com`). If the attacker finds a page on `www.a.com` that redirects to HTTP or to his own website, say on the path `/p`, then it can send the user to the URL: `https://idp.com/token?redirect_url=X/p`, which in turn redirects to: `https://oauth.a.com/p#token`. The attacker redirects the request to `oauth.a.com` to point to the server that handles `www.a.com`. The request is thus redirected to, say, `http://attacker.com/#token` which leaks the access token to the attacker. We found that many of the top Alexa websites that use single sign-on systems are vulnerable to these origin confusion exploits based on cross-protocol redirections in practice (including Pinterest and Yahoo).

Responsible Disclosure We discussed this attack with leading identity providers such as Facebook. We agree it is inherently caused by the weakness of OAuth to redirection attacks, a problem that is well known and can only be avoided by properly following recommendations regarding redirections on OAuth-enabled websites.

4.2 Hosted Contents: Dropbox

Dropbox allows users to share their public files on the low-trust origin `dropboxusercontent.com`, whereas it deploys state of the art HTTP security protections on its high-trust origin `www.dropbox.com`, including HSTS to prevent any network attack. However, non-public files cannot be served from this low-trust origin when the user wants to download data from her account, because they require access to the session cookie to prove that the user is authorized to view the file. Thus, the `dl-web.dropbox.com` origin is used for the purpose of displaying files from the user's own Dropbox account while he is logged in. This origin uses the same wildcard certificate as `www.dropbox.com`.

Using virtual host confusion, an attacker is able to load a page from the `dl-web` subdomain under the `www` origin. To turn this into an exploit, the attacker can take advantage of the complete lack of integrity guarantee for cookies [20, 21]. The attacker then performs the following steps:

1. store a malicious HTML page on his own Dropbox account (on `https://dl-web.dropbox.com/m`);
2. trigger request to `http://attacker.dropbox.com` (not protected by HSTS) and inject a `Set-Cookie` header in the response with `domain=.dropbox.com` and a very low `max-age`, that contains his own session identifier;
3. trigger request to `https://www.dropbox.com/m`, but forward the connection to the `dl-web` server. The `Cookie` header of the request contains (depending on browser) the user's session identifier, followed by the attacker's; the Dropbox server authenticates the latter and returns the malicious page.
4. wait for the delay specified in `max-age` until the forced cookie expires;
5. perform arbitrary requests on the user's Dropbox account (same impact as a XSS flaw on `www.dropbox.com`).

Responsible Disclosure We reported this attack to the Dropbox security team, who immediately confirmed the attack and fixed their virtual host configuration.

4.3 Shared TLS Cache: Mozilla

When two different servers or virtual hosts share a TLS session cache or session ticket encryption keys, an HTTPS connection to one host may be redirected to the other (using session resumption). If one of these hosts has a lower trust level than the other, this amounts to a cross-site scripting attack. We found multiple interesting examples of servers on the web that share TLS session caches, most of which can be found in cloud infrastructures, such as Amazon Web Services, Yahoo or Google. Google is an interesting case: every single Google front-end server uses the same session

cache and ticket key. However, because they also have the exact same virtual host configuration, we found no exploit against Google servers.

We found shared session caches to be a lot more common than shared ticket keys within the sample of cloud servers we tested, which we assume to be caused by improperly configured, global-scoped caches. We observed that these global caches are often too small to store the large amounts of sessions created on these cloud services for more than a few seconds, a sufficiently long time window for attacks. However, most of these servers also implement ticket-based resumption, even though ticket keys are often not synchronized across servers (e.g. on Yahoo). Exploiting shared caches when tickets are enabled requires another tool in the network attacker arsenal.

Browsers attempt to maximize their compatibility with buggy TLS implementations by retrying failed handshakes with downgraded TLS versions, all the way from TLS 1.2 to SSL3. There have been concerns about downgrading; in fact, browsers are moving away from the practice because of another TLS attack (see Section 7 for details). By intercepting connections and injecting TLS alerts on strong protocol versions, an attacker is able to ensure that the browser will connect to its target website with SSL 3.0. Hence, features that rely on TLS extensions, such as SNI and ticket-based resumption, become unavailable.

We put this attack into practice to exploit origin confusion on Mozilla servers hosted on the Amazon cloud. We first noticed that a number of Mozilla domains serve dangerous content. For instance, `git.mozilla.org` or `hg.mozilla.org` contain many third party files, as well as a number of test HTML pages for the Firefox browser, some of which deliberately include XSS flaws. Even though these domains use dedicated certificates, their server share a server-side session cache with several other Mozilla domains, including high-security ones such as the one used for bug reports `bugzilla.mozilla.org`.

Figure 5 depicts the virtual host confusion exploit, which translate to the following steps for the attacker:

1. find a vulnerable page `/p` on low-trust origin `git`;
2. trigger a request to `https://bugzilla.mozilla.org/` (which has a single-domain, extended-validation certificate), while downgrading the connection to SSL 3.0, ensuring the lack of a TLS session ticket;
3. trigger request to `https://bugzilla.mozilla.org/p`, forwarding the connection to `git` server. The browser resumes the previous TLS sessions, even though the `git` server uses a completely different certificate. Despite the wrong `Host` header, the request is processed by the `git` virtual host;
4. compromise `bugzilla` origin with the XSS flaw on `/p`.

As usual, the whole attack can occur in the background without any user involvement (besides visiting any HTTP website on the attacker network).

Responsible Disclosure We reported this attack to Mozilla in bug 1004848. It was traced to a session cache isolation vulnerability in the Stingray Traffic Manager, which was fixed in version 9.7. We learned that a similar attack presented at Black Hat 2010 [22] had described how to transfer an XSS flaw from one Mozilla domain to another, also using virtual host confusion. Surprisingly, the hackers who described the attack consider it too targeted to be serious.

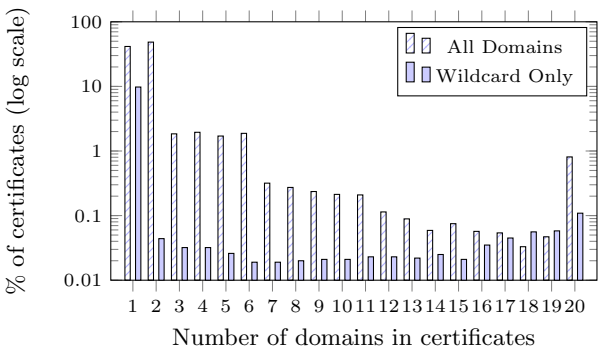

Figure 6: Issuance of multi-domain certificates

5. IMPACT MEASUREMENT

We have described four different exploits of virtual host confusion against major websites. However, these particular exploits do not give a clear picture of the general proportion of all websites vulnerable to similar attacks.

Virtual Host Fallback The main ingredient of our origin confusion attacks is virtual host fallback. We tested the top three most popular HTTPS implementations according to the September 2014 Netcraft Web Survey[2] with a configuration similar to the one in Figure 4 (without any deliberate effort to defend against virtual host confusion) and found that fallback was indeed possible on IIS (36% of servers), Apache (35%), and Nginx (14%).

Multi-Domain Certificates The issuance of new certificates by certification authorities is monitored fairly closely, including by the academic community [23]. We can easily build statistics about the number of domains found in publicly trusted certificates issued between July 2012 and July 2013 based on data collected in [9]. The results, depicted in Figure 6, show that about 40% of issued certificates are valid for a single domain; however, about 10% of them contain a wildcard. Many certificates are valid for two domains, but among them, over 95% list the same top-level domain with and without the `www` prefix (which can already lead to confusion attacks, but in most cases, both are be served by the same virtual host).

Shared TLS Cache Evaluating TLS session cache sharing is very difficult: any two servers on the web can potentially share their session ticket key or session database, regardless of their IP addresses and certificates. We were able to find several examples of shared session caches by manually testing servers within the same IP ranges known to be used by cloud services. Still, the actual number of vulnerable servers remains mostly unknown.

Cross-Protocol Redirections We have shown in Section 4.1 that a network attacker can impersonate users on websites that use single sign-on protocols based on token redirection to a secure registered origin, if this origin can be confused for another which contains redirections to any plain HTTP URL. To evaluate this scenario, we considered the HTTPS-enabled subset of the Alexa Top 10,000 websites [24], and simply sent a request for the path `/404`. In

[2] `http://news.netcraft.com/archives/2014/09/24/september-2014-web-server-survey.html`

about 1 out of 6 cases, this request was redirected to HTTP. Next, we decided to manually inspect the top 50 Alexa websites in the US that implement a single sign-on system. We found that 15 of them had in fact registered an HTTP origin with their identity provider (allowing a network to get an access token to impersonate the user without any effort). In 21 other cases, we found a page that redirects to HTTP within the secure registered origin (in such cases, the attacker can obtain access tokens without virtual host confusion). Finally, we found 11 instances where virtual host confusion could be used to recover the access tokens.

Overall, the results of our study on the 50 most popular websites in the US show that access tokens are for the most part not adequately protected against network attacks, which is consistent with previous results [25–28]. In particular, the dangers of cross-protocol redirections appears to be widely underestimated, especially on websites that implement single sign-on protocols.

6. CONNECTION SHARING IN SPDY AND HTTP/2

We have demonstrated in the previous sections that there exists a significant gap between the models used to analyze the security of TLS and the actual deployment of HTTPS in practice. However, web technologies are evolving so quickly that even the HTTPS multiplexer model presented in this paper fails to capture all current uses of TLS on the web.

In this section, we investigate the next-generation web protocols: SPDY [13] (which is already implemented major browsers such as Chrome, Firefox and Internet Explorer), and its derived IETF proposal for HTTP 2.0 [29]. An important design goal of these new protocols is to improve request latency over HTTPS. To this end, SPDY attempts to reduce the number of non-resumption TLS handshakes necessary to load a page by allowing browsers to reuse previously established sessions that were negotiated with a different domain, under certain conditions. In current HTTP2 drafts, this practice is called *connection reuse* [29, Section 9.1.1], but we also use the expression *connection sharing*.

Connection Sharing Recall that in normal TLS resumption, the browser caches TLS sessions indexed by domain and port number. On the client-side, there is no confusion between the different notions of identity: the origin of the request matches the SNI sent by the client, its HTTP Host header, the index of the session in the cache, and the origin used by the same-origin policy (assuming the client is not buggy). Thus, when accessing a website `https://w.com`, the browser may resume its session to download a picture at `https://w.com/y`, but it needs to create a fresh session if the picture is loaded from `https://i.w.com`, even if the domains `w.com` and `i.w.com` are served by the same server, on the same IP, and using the same certificate.

Connection reuse in SPDY and HTTP2 is a new policy that allows the browser to send the request to `i.w.com` on the session that was established with `w.com`, because it satisfies the two following conditions:

1. the certificate that was validated during the handshake of the session being reused also covers the domain name of the new request;
2. the original and new domain names both point to the same IP address.

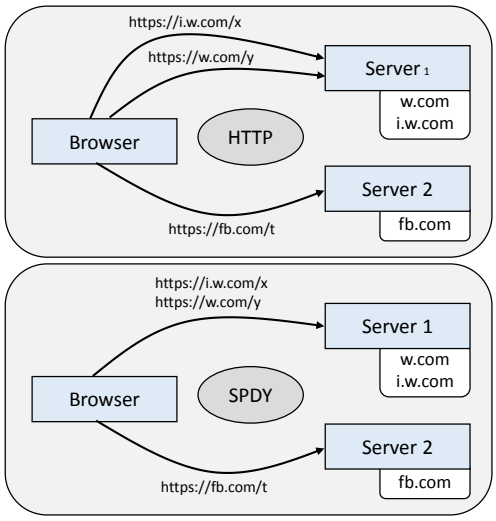

Figure 7: Connection Reuse in SPDY

Figure 7 illustrates connection reuse in SPDY: each arrow represents the TLS session used for the request(s) in its label. Because `w.com` and `i.w.com` point to the same IP or Server 1, which uses a certificate that covers both names, the same TLS session can be reused for requests to both domains.

Security Impact Connection sharing negates important assumptions used in several TLS and HTTPS mechanisms, such as TLS client authentication [30], Channel ID and Channel Bindings [31, 32] or certificate pinning [33, 34]. Concretely, every feature derived from the TLS handshake may no longer be considered to apply to the domain for which the session was created, but instead, to potentially any name present in the certificate used during the handshake. It is tempting to argue that the fact these domains appear in the same certificate is a clue that their sharing of some TLS session-specific attributes could be acceptable, but we stress that it is in fact not the case. For instance, recall from Section 2 that CloudFlare uses shared certificates that cover dozens of customers' domains [9, 35]. In fact, it is common on today's web to *connect to a website whose certificate is shared with a malicious, attacker-controlled domain.* With connection reuse, requests for the honest and malicious domain not only use the same TLS session, but possibly *the same connection* as well.

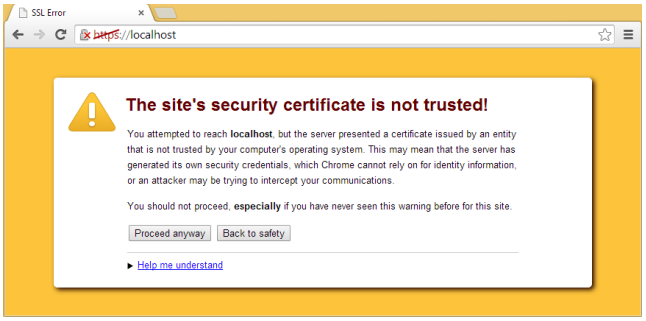

Figure 8: Interstitial Certificate Warning in Chrome

Exploit against Chrome With connection reuse, when a certificate is accepted by the browser during a TLS handshake, the established session can potentially be used for requests to all the domains listed in the certificate. The condition about the IP address of all these domains being the same doesn't matter to a network attacker who is anyway in control of the DNS.

In Chrome up to version 36, if a network attacker can get a user to click through a certificate warning for any unimportant domain (users may be used to ignore such warnings when connecting to captive portals, commonly found in hotels and other public network), he may be able to impersonate an arbitrary set of other domains, by listing them in the subject alternative name extension of the certificate (which has no displayed feedback in the interstitial warning, as shown in Figure 8). If the user attempts to connect to any of these added domains (say, `facebook.com`), the attacker can tamper with the DNS request for `facebook.com` to return his own IP address, which tells the browser it can reuse the SPDY connection established with the attacker when the self-signed malicious certificate was accepted. Although Chrome will keep the red crossed padlock icon in the address bar because of the invalid certificate of the original session, the attacker can still collect the session cookies for any number of websites in the background.

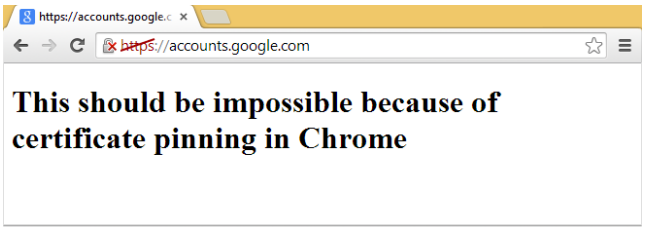

Figure 9: Compromise of Pinned, HSTS Origin

Interestingly, since the only trust decision made by the browser occurs when the bad certificate is accepted, this attack is able to bypass all security protections in Chrome against TLS MITM attacks. For instance, when a domain enables HSTS, certificate warning on this domain can not longer be clicked through by the user. Similarly, the Chrome browser includes a pinning list of certificates used by top websites, which successfully detected at least two man-in-the-middle attacks that were using improperly issued trusted certificates recently. Since these checks are only performed when a certificate is validated, they fail to trigger on reused connections, as shown in Figure 9. The user isn't shown any further certificate warning after the one caused by the attacker on an innocuous domain.

Responsible Disclosure This bug (CVE-2014-3166) was fixed by a security update for Chrome 36.

7. COUNTERMEASURES AND MITIGATION

Thorough this paper, we have pointed out multiple flaws both at the transport and application levels that prevent proper virtual host isolation on the server, and break the same origin policy on the client as a result. In this section, we summarize the possible countermeasures and mitigations that can prevent this class of attacks at each network layer.

Preventing Virtual Host Fallback Our evaluation shows

```
server {
        listen 1.2.3.4:443 ssl default_host;
        server_name "";
        # Used if no SNI is present in client hello
        ssl_certificate "/etc/ssl/a.com.pem";
        return 400;
}
```

Figure 10: Preventing virtual host fallback

that the fallback mechanism of the virtual host selection algorithm in current HTTPS servers is by far the leading factor in exploiting confusion vulnerabilities. For instance, even though all the services hosted by Google suffer from TLS session confusion, it cannot be directly exploited because all the front-end servers serve the same set of virtual hosts without fallback. We propose that upon receiving a request with a `Host` header that doesn't match any of the configured virtual host names, the server should immediately return an error. In particular, a request without a `Host` header would always be rejected (thus breaking compatibility with HTTP/1.0 clients). While this change only needs to apply to requests received over TLS, it does break backwards compatibility and may cause improperly configured websites to stop working. Therefore, none of the vendors we contacted are willing to implement such a change.

Authenticating Port in Host Header Currently, web servers ignore the port indicated by the client in the `Host` header, thus making it useless for the purpose of origin isolation. We propose that for requests received over TLS, a web server should compare the port included in the `Host` header with the one the request was sent to. We argue that unless this change is implemented in all HTTPS server software, browsers should stop using the port for origin isolation purposes, given that this isolation is mostly illusionary. This is the approach currently adopted by Internet Explorer.

Preventing Cross-Virtual Host Resumption In HTTP 1.1, there is no circumstance under which a session negotiated on a given virtual host would ever be resumed on another host with a different name or certificate. However, this invariant was broken with the introduction of SPDY and HTTP2 connection sharing. Thus, our initial suggestion to cryptographically bind TLS sessions with the virtual host they were created for was rejected. However, since connection sharing is only supposed to happen on the same IP address, it still makes sense to strictly block resumption across hosts on different TCP sockets. We convinced Nginx to implement such an isolation both for their server-side cache and session ticket implementation, starting from version 1.7.5 (CVE-2014-3616).

Preventing SSL Downgrading The attack we present in Section 4.3 was first to demonstrate that SSL downgrading can be taken advantage of by a network attacker to exploit virtual host confusion attacks. The recent POODLE attack (CVE-2014-3566) also exploits downgrading to mount a padding oracle attack; as a result, Chrome and NSS have removed downgrading to SSL3. A draft has also been submitted to the TLS working group of the IETF to introduce a new extension that prevents the attacker from downgrading the TLS version [36].

Configuration Guidelines for Current Web Servers Even without modifying web server software or the TLS library, there are some safe usage guidelines that website administrators can use to mitigate the attacks described in this paper. As a general rule, we recommend that only domains with the same trust levels should be allowed to share a certificate. It is best to avoid wildcard certificates, as they are valid even for non-existing subdomains, which increases the likelihood of a virtual host fallback. Anytime a certificate is used on a virtual host, it is necessary to ensure that all the domain names it contains have a matching virtual host configured on the same IP address and port; or at least a default one that returns an error. The same check applies to every other pair of IP address and port where this certificate is used. For domains with wildcards, the associated virtual host must use a regular expression that reflects all possible names. In cases where only some of the domains in the certificate are served on this IP, it is necessary to configure an explicit default host similar to the one given in Figure 10.

Session caches should be configured on a per-virtual host basis. Furthermore, all the ticket keys and shared cache names must be different in every virtual host where they are defined, unless SPDY connection sharing is used.

Cross-protocol redirections should be avoided in all TLS-enabled virtual hosts. When plaintext and encrypted versions of the same virtual host need to coexist, protocol-relative URLs (such as `//x.a.com/p`) should be used.

Finally, whenever possible, it is best to avoid cookies altogether, in particular to implement sessions: the origin-bound `localStorage` provides a safer alternative. If cookie-based sessions cannot be avoided (e.g. because a session cookie must be available to multiple subdomains), the page that sets the cookie should be served from the top-level domain using the `includeSubdomains` option of HSTS.

8. RELATED WORK

Origin confusion attacks may target the same-origin policy at various levels in the browser. The policy for cookies (which are always attached to requests regardless of their source origin) is often abused to mount cross-site request forgery attacks [20]. Implementations of single sign-on protocols [37] have been found to suffer from many flavors of origin confusion, sometimes on the messaging between frames by `postMessage` [38], sometimes because of JavaScript bugs [39], and often because of dangerous redirections [26, 40].

Among the documented network attack on HTTPS [34], the easiest is to trick clients into using HTTP instead; a method called SSL stripping [41]. To prevent such attacks, browsers and servers now implement Strict Transport Security (HSTS) [42], which can itself be sometimes attacked [32, 43]. Virtual host confusion attacks apply even to websites that use HSTS, since they rely on TLS credential sharing. However, some of the concrete exploits in this paper rely on some domains not requiring HSTS, for instance the exploit against Dropbox from Section 4.2.

Typical man-in-the-middle attacks on HTTPS rely on DNS rebinding [22, 44] or cache poisoning [45, 46] and on fooling the client into accepting a bogus, mis-issued, or compelled certificate [47, 48]. The goal is for a network attacker to impersonate a trusted HTTPS server [49]. Our attacks rely on shared server credentials to obtain similar impact, but do not require buggy clients [50], or on users clicking through certificate warnings [51] on the attacked origin. Our threat model, which mixes web and network attacks, is similar to those of recent cryptographic attacks on HTTPS, notably BEAST [52], CRIME [53] and FREAK [54], but the attacks in this paper do not rely on cryptographic weaknesses.

There have been many proposals to improve the PKI such as pinning [33], certificate transparency [55], or ARPKI [56], but they fail to prevent our attacks, which rely on bad certificate practices (in particular, the use of wildcard and shared certificates) by honest websites.

9. CONCLUSION

In this paper, we have have shown that the isolation between HTTPS origins in various kinds of shared environments (shared or overlapping certificate, content delivery networks, shared session cache, different ports on the same domain) can be broken by weaknesses in the handling of HTTP requests and the isolation of TLS session caches, resulting in high impact exploits. Preventing all virtual host confusion attacks requires vendors of HTTP servers to stop virtual host fallback when processing requests over TLS. However, from the feedback we received when we disclosed these attacks, such a change is unlikely to occur. In fact, virtual host confusion may become more common when HTTP2 gets deployed, and features such as connection sharing introduce a new "same-certificate policy" approach that can interfere badly with the same-origin policy enforced by browsers. We leave to future work the question of how to redesign HTTPS deployment to improve security isolation in the presence of shared credentials and connections.

10. REFERENCES

[1] S. Landau, "Highlights from making sense of Snowden, part II: What's significant in the NSA revelations," *IEEE Security & Privacy*, vol. 12, pp. 62–64, 2014.

[2] T. Dierks and E. Rescorla, "The Transport Layer Security Protocol Version 1.2," RFC 5246, 2008.

[3] E. Rescorla, "HTTP over TLS," RFC 2818, 2000.

[4] K. G. Paterson, T. Ristenpart, and T. Shrimpton, "Tag size does matter: attacks and proofs for the TLS record protocol," in *ASIACRYPT*, 2011.

[5] H. Krawczyk, K. G. Paterson, and H. Wee, "On the security of the TLS protocol: a systematic analysis," in *CRYPTO*, 2013.

[6] K. Bhargavan, C. Fournet, M. Kohlweiss, A. Pironti, and P. Strub, "Implementing TLS with verified cryptographic security," in *IEEE S&P*, 2013.

[7] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright, "Transport Layer Security (TLS) Extensions," IETF RFC 3546, 2003.

[8] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the HTTPS certificate ecosystem," in *IMC*, Oct. 2013.

[9] A. Delignat-Lavaud, M. Abadí, M. Birrell, I. Mironov, T. Wobber, and Y. Xie, "Web PKI: closing the gap between guidelines and practices," in *NDSS*, Feb 2014.

[10] I. Fette and A. Melnikov, "The WebSocket protocol," RFC 6455, 2011.

[11] M. Zalewski, "Browser Security Handbook," Web: http://code.google.com/p/browsersec/, undated.

[12] J. Grossman, *XSS Attacks: Cross-site scripting exploits and defense*. Syngress, 2007.

[13] M. Belshe and R. Peon, "The SPDY protocol," IETF draft-mbelshe-httpbis-spdy-00, 2012.
[14] Akamai Technologies, "Visualizing akamai," akamai.com/html/technology/dataviz3.html, 2014.
[15] J. Liang, J. Jiang, H. Duan, K. Li, T. Wan, and J. Wu, "When HTTPS meets CDN: A case of authentication in delegated service," in *IEEE S&P*, 2014.
[16] I. Sysoev and B. Mercer, "How nginx processes requests," nginx.org/docs/http/request_processing.html, 2012.
[17] Apache Foundation, "Virtual host documentation," http://httpd.apache.org/docs/current/vhosts/, 2014.
[18] S. Stamm, B. Sterne, and G. Markham, "Reining in the web with content security policy," in *WWW*, 2010.
[19] E. Hammer-Lahav, D. Recordon, and D. Hardt, "The OAuth 2.0 Authorization Protocol," IETF Draft, 2011.
[20] A. Barth, C. Jackson, and J. C. Mitchell, "Robust defenses for cross-site request forgery," in *CCS*, 2008.
[21] A. Bortz, A. Barth, and A. Czeskis, "Origin cookies: session integrity for web applications," in *W2SP*, 2011.
[22] R. Hansen and J. Sokol, "MitM DNS rebinding SSL wildcards and XSS," http://goo.gl/23Yt9l, 2010.
[23] M. Schloesser, B. Gamble, J. Nickel, C. Guarnieri, and H. D. Moore, "Project sonar: IPv4 SSL certificates," https://scans.io/study/sonar.ssl, 2013.
[24] Alexa Internet Inc., "Top 1,000,000 sites (updated daily)," http://goo.gl/OZdT6p, 2014.
[25] S. Pai, Y. Sharma, S. Kumar, R. M. Pai, and S. Singh, "Formal verification of oauth 2.0 using alloy framework," in *CSNT*. IEEE, 2011.
[26] S.-T. Sun and K. Beznosov, "The devil is in the (implementation) details: an empirical analysis of OAuth SSO systems," in *CCS*. ACM, 2012.
[27] C. Bansal, K. Bhargavan, and S. Maffeis, "Discovering concrete attacks on website authorization by formal analysis," in *CSF*. IEEE, 2012.
[28] D. Akhawe, A. Barth, P. Lam, J. Mitchell, and D. Song, "Towards a formal foundation of web security," in *CSF*, 2010, pp. 290–304.
[29] M. Belshe, R. Peon, and M. Thomson, "Hypertext transfer protocol version 2," 2012. [Online]. Available: http://tools.ietf.org/html/draft-ietf-httpbis-http2-14
[30] A. Parsovs, "Practical issues with TLS client certificate authentication," in *NDSS*, 2014.
[31] M. Dietz, A. Czeskis, D. Balfanz, and D. S. Wallach, "Origin-bound certificates: a fresh approach to strong client authentication," in *Usenix Security*, 2012.
[32] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, , A. Pironti, and P.-Y. Strub, "Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS," in *IEEE S&P*. IEEE, 2014.
[33] C. Evans and C. Palmer, "Certificate pinning extension for HSTS," 2011. [Online]. Available: http://tools.ietf.org/html/draft-evans-palmer-hsts-pinning-00
[34] C. Meyer and J. Schwenk, "SoK: Lessons learned from SSL/TLS attacks," in *Information Security Applications*, ser. LNCS. Springer, 2014, pp. 189–209.
[35] J. Liang, J. Jiang, H. Duan, K. Li, T. Wan, and J. Wu, "When HTTPS meets CDN: A case of authentication in delegated service," in *IEEE Symposium on Security & Privacy 2014 (Oakland'14)*. IEEE, 2014.
[36] B. Moeller and A. Langley, "TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks," Internet Draft (v.01), 2014.
[37] R. Wang, S. Chen, and X. Wang, "Signing me onto your accounts through Facebook and Google: A traffic-guided security study of commercially deployed single-sign-on web services," in *IEEE S&P*, 2012.
[38] D. Fett, R. Kusters, and G. Schmitz, "An expressive model for the web infrastructure: definition and application to the BrowserID SSO system," in *IEEE S&P*, 2014.
[39] K. Bhargavan, A. Delignat-Lavaud, and S. Maffeis, "Language-based defenses against untrusted browser origins," in *Usenix Security*, 2013.
[40] C. Bansal, K. Bhargavan, and S. Maffeis, "Discovering concrete attacks on website authorization by formal analysis," in *CSF*, 2012.
[41] M. Marlinspike, "More tricks for defeating SSL in practice," *Black Hat USA*, 2009.
[42] J. Hodges, C. Jackson, and A. Barth, "HTTP Strict Transport Security (HSTS)," IETF RFC 6797, 2012.
[43] J. Selvi, "Bypassing http strict transport security."
[44] C. Jackson, A. Barth, A. Bortz, W. Shao, and D. Boneh, "Protecting browsers from DNS rebinding attacks," *TWEB*, vol. 3, no. 1, 2009.
[45] S. Son and V. Shmatikov, "The hitchhiker's guide to DNS cache poisoning," in *SecureComm*, 2010.
[46] D. Dagon, M. Antonakakis, P. Vixie, T. Jinmei, and W. Lee, "Increased DNS forgery resistance by 0x20-bit encoding: security via leet queries," in *CCS*, 2008.
[47] N. Karapanos and S. Capkun, "On the effective prevention of TLS man-in-the-middle attacks in web applications," in *Usenix Security*, 2014.
[48] C. Soghoian and S. Stamm, "Certified lies: setecting and defeating government interception attacks against SSL," in *FC*, 2012.
[49] C. Karlof, U. Shankar, J. D. Tygar, and D. Wagner, "Dynamic pharming attacks and locked same-origin policies for web browsers," in *CCS*, 2007.
[50] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov, "The most dangerous code in the world: validating SSL certificates in non-browser software," in *ACM CCS*, 2012.
[51] D. Akhawe, B. Amann, M. Vallentin, and R. Sommer, "Here's my cert, so trust me, maybe? understanding TLS errors on the web," in *WWW*, 2013.
[52] T. Duong and J. Rizzo, "Here come the XOR ninjas," *White paper, Netifera*, 2011.
[53] J. Rizzo and T. Duong, "The CRIME attack," in *EKOparty Security Conference*, vol. 2012, 2012.
[54] B. Beurdouche, K. Bhargavan, A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, A. Pironti, P.-Y. Strub, and J. K. Zinzindohoue, "A messy state of the union: taming the composite state machines of TLS," in *IEEE S&P*, 2015.
[55] B. Laurie, "Certificate transparency," *Commun. ACM*, vol. 57, no. 10, 2014.
[56] D. Basin, C. Cremers, T. H.-J. Kim, A. Perrig, R. Sasse, and P. Szalachowski, "ARPKI: Attack resilient public-key infrastructure," in *CCS*, 2014.