ForceHTTPS: Protecting High-Security Web Sites from Network Attacks

Collin Jackson
Stanford University
collinj@cs.stanford.edu

Adam Barth
Stanford University
abarth@cs.stanford.edu

ABSTRACT

As wireless networks proliferate, web browsers operate in an increasingly hostile network environment. The HTTPS protocol has the potential to protect web users from network attackers, but real-world deployments must cope with misconfigured servers, causing imperfect web sites and users to compromise browsing sessions inadvertently. ForceHTTPS is a simple browser security mechanism that web sites or users can use to opt in to stricter error processing, improving the security of HTTPS by preventing network attacks that leverage the browser's lax error processing. By augmenting the browser with a database of custom URL rewrite rules, ForceHTTPS allows sophisticated users to transparently retrofit security onto some insecure sites that support HTTPS. We provide a prototype implementation of ForceHTTPS as a Firefox browser extension.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Unauthorized Access*; K.4.4 [**Computers and Society**]: Electronic Commerce—*Security*

General Terms

Design, Security, Human Factors

Keywords

HTTPS, eavesdropping, pharming, same-origin policy

1. INTRODUCTION

HTTPS is designed to be secure against both eavesdroppers and active network attackers. In practice, however, all modern web browsers are willing to compromise the security of sites that use HTTPS in order to be compatible with sites that deploy HTTPS incorrectly. For example, if an active attacker presents a self-signed certificate, web browsers permit the user to click through a warning message and access the site despite the error. This behavior compromises the confidentiality of the site's `Secure` cookies, which often store a second factor of authentication, and allows the attacker to hijack a legitimate user's session, potentially letting the attacker to transfer money out of the user's bank account or perform other misdeeds. Browsers accept broken certificates and allow embedding of insecure scripts for two reasons:

- **Compatibility.** Many web sites have incorrectly configured certificates and embed insecure scripts. A browser that enforces strict error processing is incompatible with these sites and will lose users to a more permissive browser.
- **Unknown Intent.** Some site owners intentionally use self-signed certificates and host portions of their site over HTTP because these mechanisms provide protection from passive attackers and they believe the risk of an active attack is outweighed by the cost of implementing HTTPS fully.

Although a security-conscious site owner, such as a bank, might aim to implement a high-security site, he or she currently has no mechanism for communicating this intent to the browser. Other site owners that are less security-conscious, desiring protection only from passive network attackers, implement low-security sites by deploying certificates that are self-signed or have incorrect common names. The browser has no mechanism for differentiating these two kinds of sites and cannot distinguish between a legitimate misconfiguration in a low-security site and an attack on a high-security site. Without guidance, a browser does not have the context to make an useful risk-management decision about whether to trade off security for compatibility on a particular site.

1.1 Our Proposal

We propose ForceHTTPS, a simple mechanism that security-conscious sites can use to opt in to stricter error processing by the browser, essentially giving the browser guidance to be more secure. By setting a ForceHTTPS cookie, a site owner asks the browser to treat HTTPS errors as attacks, not as simple configuration mistakes. Specifically, enabling ForceHTTPS causes the brower to modify its behavior as follows:

1. Non-HTTPS connections to the site are redirected to HTTPS, preventing contact to the site without TLS.
2. All TLS errors, including self-signed certificates and common-name mismatches, terminate the TLS session.
3. Attempts to embed insecure (non-HTTPS) content into the site fail with network errors.

This stricter error handling has several benefits, including protecting the URL parameters, fragments, and `Secure` cookies from network attackers and users who click through security warnings. ForceHTTPS blocks participating sites from

WWW 2008, April 21–25, 2008, Beijing, China.
ACM 978-1-60558-085-2/08/04.

embedding insecure content, such as scripts, cascading style sheets, and SWF movies, in order to secure the user's session with buggy sites that would otherwise allow an active network attacker to steal the user's password and second factor of authentication by silently replacing SWF movie embedded in the login page. By enabling ForceHTTPS, a site protects itself from careless mistakes by its own web developers. ForceHTTPS also offers a "developer mode" that explains these errors so that the site's web developer can find and fix vulnerabilities.

Used in concert with a phishing defense, such as Bank of America's SiteKey [1], ForceHTTPS lets a site protect itself from pharming. Previously proposed anti-pharming defenses [6, 20, 15] are difficult to implement and face major challenges to deployment. By contrast, ForceHTTPS is easy to implement because browsers already detect the errors sites wish to block and easy to deploy because sites need only set a single cookie. To demonstrate the feasibility of our approach, we provide a prototype of ForceHTTPS as a Firefox browser extension [12].

1.2 Power Users

ForceHTTPS also enables "power users" to upgrade the security of sites that implement HTTPS insecurely by setting a ForceHTTPS cookie on the site's behalf. This approach follows a recent trend in which sophisticated users have taken web security into their own hands. The NoScript [18] browser extension enables users to fix cross-site scripting vulnerabilities in sites they visit by disabling or limiting the capabilities of scripts on that site, albeit at the cost of functionality. Other client side tools for mitigating web site vulnerabilities include Noxes [16] and NoMoXSS [28]. The GMailSecure user script (which has had over 25,000 downloads) enables users to force secure connections to Gmail, mitigating eavesdropping attacks without any reduction in functionality.

In fact, this paper arose largely out of a desire by the authors to secure their Gmail sessions while using the wireless networks at security conferences after witnessing an alarmingly effective attack demonstration at Black Hat 2007 [10]. Securing Gmail without Google's cooperation is challenging because Gmail's session identifier is stored in an insecure cookie that is transmitted whenever a user visits any other Google property. By setting the ForceHTTPS cookie, a Gmail user upgrades the session cookie to a `Secure` cookie that is protected from both eavesdropping and active attackers.

1.3 Organization

The rest of this paper is organized as follows. In Section 2 we describe the threats that ForceHTTPS is designed to protect against. In Section 3 we survey existing techniques that attempt to defend against these threats. In Section 4 we provide a specification of our proposal. In Section 5 we discuss design decisions and implementation details. We conclude in Section 6.

2. THREAT MODEL

2.1 Threats Addressed

ForceHTTPS is concerned with three threats: passive network attackers, active network attackers, and imperfect web developers.

- **Passive Network Attackers.** When a user browses the web on a wireless network, a nearby attacker can eavesdrop on unencrypted connections, such as HTTP requests. Such a passive network attacker can steal session identifiers and hijack the user's session. These eavesdropping attacks can be performed easily using wireless sniffing toolkits [29, 10]. Some sites, such as Gmail, permit access over HTTPS, leading a user to believe that accessing such a service over HTTPS protects them from an passive network attacker. Unfortunately, this is often not the case as session identifiers are typically stored in insecure cookies to permit interoperability with HTTP versions of the service. For example, the session identifier for Gmail is usually stored in a non-`Secure` cookie, permitting an attacker to hijack the user's Gmail session if the user makes a single HTTP request to Gmail. Additionally, the subjects and snippets of the one hundred most recent email messages can be retrieved using the user's `.google.com` session cookie, which is sent in the clear during every Google search request.

- **Active Network Attackers.** A more determined attacker can mount an active attack, either by impersonating a user's DNS server or, in a wireless network, by spoofing network frames or offering a similarly-named "evil twin" access point. If the user is behind a wireless home router, the attacker can attempt to reconfigure the router using default passwords and other vulnerabilities [26, 27, 25]. Some sites, such as banks, rely on HTTPS to protect them from these active attackers. Unfortunately, browsers allow their users to opt-out of these protections in order to be compatible with sites that incorrectly deploy HTTPS. These sites wish to be protected from active network attackers even if users do not understand the security warnings provided by their browsers.

- **Honest but Imperfect Web Developers.** Large web sites are constructed by numerous developers, who occasionally make mistakes and are not all security experts. One simple mistake, such as embedding a cascading style sheet or a SWF movie over HTTP, can allow an active attacker to compromise the security of an HTTPS site completely.[1] Even if the site's developers carefully scrutinize their login page for mixed content, a single insecure embedding anywhere on the site compromises the security of their login page because the attacker can script (control) the login page by injecting script into the page with mixed content. Both the site's owner and the site's users could wish the site to be secure despite its developers making mistakes.

2.2 Threats Not Addressed

- **Phishing.** Phishing attacks [7] occur when an attacker solicits authentication credentials from the user by hosting a fake site located on a different domain than the real site, perhaps driving traffic to the fake

[1] Both cascading style sheets and SWF movies can script the embedding page, to the surprise of many web developers. Most browsers do not issue mixed content warnings when insecure SWF files are embedded.

site by sending a link in an email. Phishing attacks can be very effective because users find it difficult to distinguish the real site from a fake site [5]. ForceHTTPS is not a defense against phishing, but it complements many existing phishing defenses, such as SiteKey [1], the Yahoo! Sign-in Seal [30], and Chase's Activation Code [4], by instructing the browser to protect session integrity and long-lived authentication tokens.

- **Malware and Browser Vulnerabilities.** Because ForceHTTPS is implemented as a browser security mechanism, it relies on the trustworthiness of the user's system to protect the session. Malicious code executing on the user's system can compromise a browser session, regardless of whether ForceHTTPS is used.

3. RELATED WORK

Previously known defenses to the threats described in Section 2 are shown in Table 1 and summarized in this section.

3.1 User-Controlled Defenses

- **User-enforced HTTPS.** Many web sites serve the same content over both HTTP and HTTPS, taking care to use HTTPS on the login or credit card entry page and HTTP elsewhere. This protects the user's long-lived authentication credentials and financial details from being stolen by eavesdroppers while retaining the performance benefits of unencrypted HTTP traffic. Unfortunately, many such sites set an non-`Secure` cookie containing the user's session identifier. This cookie is sent in the clear over HTTP and can be used by an eavesdropper to hijack the user's session.

  Security-conscious users can mitigate this vulnerability by attempting to visit the site using HTTPS, to the exclusion of HTTP. For example, the user can diligently type HTTPS URLs into the address bar and check the status bar before clicking on links. Unfortunately, even a single insecure HTTP request by the web site can lead to a compromise of the session cookies. If the insecure request is the result of a redirect or button click, the user could be unaware of the request until their credentials have already been compromised.

  For example, Gmail serves its content to authenticated users both over HTTPS and HTTP. The login form, however, is served exclusively over HTTPS. Users that want to check sensitive mail using Gmail can access the Gmail site over HTTPS instead of HTTP. In fact, many users install GMailSecure [21] to automatically redirect them to HTTPS pages when using Gmail. Unfortunately, GMailSecure does not actually protect the session cookie on `mail.google.com` because it performs the redirect after the browser has already sent the HTTP request (which contains the cookie) in the clear.

- **Certificate Errors.** Incorrectly configured web servers can cause a number of HTTPS certificate errors:

  - **Common-Name Mismatch.** HTTPS requires that a server present a certificate whose common name matches the server's host name. Many web servers erroneously present certificates with incorrect common names.

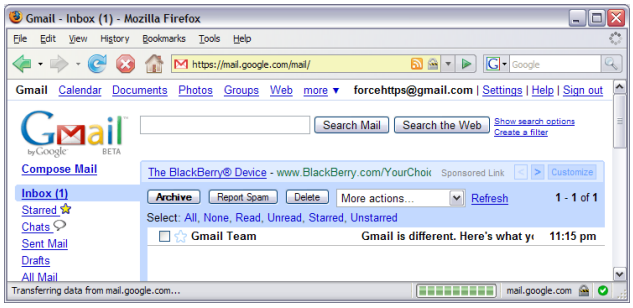

Figure 1: This account has only ever been accessed over HTTPS, but the confidentiality of this user's email has already been compromised because Firefox leaked the user's cookie in an automatic request for anti-phishing data from Google.

  - **Self-Signed.** Many site owners wish to use HTTPS but are unable or unwilling to purchase certificates from certificate authorities. Instead, these owners deploy self-signed certificates that provide security against passive attackers.
  - **Expired.** Certificates are valid only for a limited time period. Many web servers present certificates that have either not yet become valid or whose validity period has expired.

  When it encounters a certificate error, the browser presents the user with a security warning dialog, giving the user the option to continue despite the error. Browsers permit users to override these security errors in order to be compatible with misconfigured servers. Unfortunately, the warnings have become commonplace, with approximately 63% of certificates causing errors [24]. Although the user is in control, many users do not understand these warnings and are trained to ignore them by the multitude of misconfigured sites [23]. ForceHTTPS lets sites force these certificate errors to be treated as fatal.

- **Extended Validation.** Many certificate authorities issue "extended validation" (EV) certificates that require more extensive investigation by the certificate authority before being issued [9]. Like certificate warnings, EV certificates are used to present information about the connection security to the user. For example, Internet Explorer 7 and Firefox 3 highlight the site's identity in green if the site supplies a valid EV certificate. Extended validation certificates have no effect on the browser's defenses against network attackers. A site that uses EV can still be contacted via HTTP and mix insecure content into secure pages. Moreover, the user is still able to accept a broken certificate for the host, putting primary control over enforcement in the hands of the user. ForceHTTPS allows the site to make a security commitment to the browser, rather than to the user.

- **Firefox 3.** Firefox 3 contains a new user interface for dealing with certificate errors. Early versions of this interface required ten clicks to accept certificate errors and asked the user to type the domain name

	Threat Model		
	Passive Attacker	**Active Attacker**	**Imperfect Developer**
User-controlled	GMailSecure	Certificate warnings	Mixed content warnings
Site-controlled	`Secure` cookies	Locked same-origin policy, HTTPSSR	Content restrictions

Table 1: Current attempts to defend against the threats that ForceHTTPS addresses.

manually in the hopes that this process would discourage users from giving up their security. This proposal was controversial [11] and was eventually scaled back to require only four clicks [3] as a compromise for site owners that use HTTPS with self-signed certificates. ForceHTTPS avoids compromising security for usability by affecting only those sites that are security-conscious.

- **Mixed Content Warnings.** Many sites serve the same content over both HTTP and HTTPS. If the developer expected some of the content to be served over HTTP only, the developer is likely to embed scripts using absolute paths containing the `http` scheme:

```
<script src="http://a.com/foo.js"></script>
```

Unfortunately, this compromises the security of HTTPS on the entire site because an active attacker can navigate the user's browser to the broken page over HTTPS, replace the insecure script with his own, and invade the security context of the secure site. These mistakes can easily be corrected by using scheme-relative paths [8]:

```
<script src="//a.com/foo.js"></script>
```

These paths cause the browser to load the script over HTTP when the page is viewed over HTTP and over HTTPS when the page is viewed over HTTPS. Using this technique, a site can benefit from caching and increased performance when the page is viewed over HTTP but retain security when the page is viewed over HTTPS. Unfortunately, many web developers are unaware of scheme-relative paths and often accidentally embed insecure scripts into secure pages. Browsers warn the user about these insecure embeddings in different ways:

  - **Internet Explorer** displays a "mixed content" dialog that asks the user's permission before continuing. Insecure SWF movies and Java applets are loaded automatically without any warnings.
  - **Firefox** automatically accepts the mixed content, but draws a red slash over the browser's lock icon. Insecure images, SWF movies, and Java applets do not trigger the slash.
  - **Opera** automatically accepts the mixed content, but replaces the lock icon with a question mark.
  - **Safari** does not attempt to detect mixed content.

As with certificate warnings, many users do not understand mixed content warnings, and some browsers do not even give users the option of remaining secure. Users have been trained to ignore these warnings because many HTTPS pages, such as the Gmail login page shown in Figure 2, embed mixed content. ForceHTTPS lets security-conscious sites block unwanted mixed content inadvertently introduced by their imperfect developers.

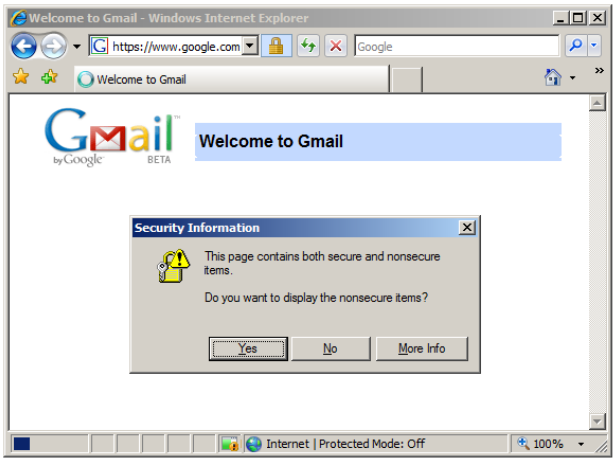

Figure 2: Users have been trained to click through mixed content warnings at sites such as Gmail.

3.2 Site-Controlled Defenses

- **Secure Cookies.** A security-conscious site can mark a cookie as `Secure`, instructing the browser to refrain from transmitting the cookie over an insecure connection. To use these cookies, the site must ensure that all authenticated web traffic occurs over HTTPS. Many sites, including those that have deployed anti-phishing defenses such as SiteKey, also use a long-lived `Secure` cookie to store a second factor of authentication.
  - **Passive Attackers.** `Secure` cookies defend well against passive eavesdroppers. We recommend that sites use `Secure` cookies as they prevent a passive attacker from learning the confidential information they store.
  - **Active Attackers.** Unfortunately, active attackers can use invalid certificates to steal `Secure` cookies if users click through certificate warning dialog boxes.

  ForceHTTPS expands the usefulness of `Secure` cookies to defend against active attackers by recording the web site's intent to use a correct HTTPS certificate. When the attacker presents an invalid certificate for the site, the browser terminates the connection and does not reveal the site's `Secure` cookies.
- **Locked Same-Origin.** Web Server Key Enabled Cookies [20] proposes restricting access to cookies based on

the public key of the server. The goal of this policy is to prevent a pharming attacker from accessing HTTPS cookies set by the victim server. Karlof et. al. [15] extend this work to defend against dynamic pharming through the use of two *locked same-origin policies* for browsers. These policies augment the browser's security policy to isolate web pages based on the security of the connection from which they were loaded. Unfortunately, both locked same-origin policies face major deployment challenges.

  - **Weak.** The weak locked same-origin policy isolates pages loaded over broken HTTPS connections from those loaded over unbroken connections. To be secure against an active attacker, a site must not embed any scripts, cascading style sheets, applets, or SWF movies (instead, the site must inline all scripts and style sheets) [15], but this requires virtually all web sites to implement major changes in order to meet this condition.
  - **Strong.** The strong locked same-origin policy segregates two pages if they where loaded over HTTPS connections with different public keys. To enable the strong policy, a site must deploy a `pk.txt` file that specifies the public keys with which it intends to interact. This file is difficult to deploy correctly and must be maintained as servers refresh their keys, likely resulting in a similar misconfiguration rate to that of deploying certificates for HTTPS.

  ForceHTTPS also isolates broken and unbroken pages by allowing security-conscious sites to forbid the browser from loading broken sites, but ForceHTTP is easier for sites to deploy: the site can opt in to ForceHTTPS by simply setting a cookie.

- **Content Restrictions.** Using content restrictions, web servers can transmit metadata to browsers instructing them to impose certain restrictions on the web site's content, such as which scripts are allowed to run. Content restrictions can limit the damage caused by a cross-site scripting attack in which the developer incorrectly sanitizes malicious input. Content restrictions can be communicated in HTTP headers or `<meta>` tags [19]. Other proposals include whitelists written in JavaScript, or using a special `noexecute` property of DOM nodes [13]. ForceHTTPS is another set of content restrictions, but instead of defending against a web developer who inadvertently exposes the session to cross-site scripting attacks, it defends against a web developer who inadvertently exposes the site to network attacks via mixed content.

4. SPECIFICATION

ForceHTTPS can be enabled in two ways:

- **Site.** A security-conscious site can enable ForceHTTPS by setting a cookie with the name `ForceHTTPS` using a `Set-Cookie` header in an error-free HTTPS response. The browser will enable ForceHTTPS for that site as long as the cookie has not expired. The `domain` and `path` attributes of the cookie are ignored.
- **User.** A security-conscious user can enable ForceHTTPS for a host through the browser user interface. The browser gives them the option of configuring custom HTTP-to-HTTPS redirection rules and non-`Secure`-to-`Secure` cookie upgrades for that domain.

ForceHTTPS can be disabled only by an error-free HTTPS response or by the browser's user interface.

When ForceHTTPS is enabled for a host, the browser modifies its behavior as follows:

- Attempts to connect over a non-HTTPS protocol are redirected to HTTPS.
- TLS errors during connections are treated as fatal.
- Attempts to embed insecure content in pages fail.

These rules prevent an active attacker from injecting script into the host's security origin.

5. DISCUSSION

This section contains a discussion of design decisions, error handling scenarios, limitations, and alternate policy advertisement mechanisms.

5.1 Design Decisions

Although the ForceHTTPS mechanism is simple, a number of subtle decisions were made during its design.

- **Redirecting URLs.** When ForceHTTPS is enabled for a host, the browser redirects HTTP requests to that host to HTTPS. For example, if the user types `www.paypal.com` in the location bar, the browser connects to `https://www.paypal.com/` instead, preventing a network attacker from intercepting the HTTP request and redirecting the user to a phishing web site. Additionally, this browser-side redirection transparently corrects a common mixed content scenario in which a site embeds active content from itself over HTTP. To retrofit security onto sites like Google that do not serve all of their content over HTTPS, ForceHTTPS lets power users configure custom rewrite rules.
- **State Exhaustion.** Because the browser has limited state, the browser's cookie eviction policy is critical to the security of ForceHTTPS. An attacker who is able to force the browser to evict the ForceHTTPS cookie is effectively able to "unforce" HTTPS. Moreover, if the browser evicts the ForceHTTPS cookie before other cookies for the same host, the attacker can potentially use the non-evicted cookies (which might store session tokens or second factors of authentication) as part of an attack. To prevent these state exhaustion attacks, the browser should reserve space for ForceHTTPS cookies and limit the rate at which it accepts new ForceHTTPS cookies. If the browser uses an rate-limiting scheme with exponential back-off, the browser can typically prevent an attacker from flooding its ForceHTTPS cookie store in a single session. A concerted attacker, however, can eventually overflow the state limit over many successive sessions. To prevent the other cookies from being stolen, the browser should evict all other cookies for a domain if it evicts the ForceHTTPS cookie.

- **Denial of Service.** The largest risk in deploying ForceHTTPS is that of denial of service. An attacker who can set a ForceHTTPS cookie for a victim host can prevent users from using that site if the site requires broken HTTPS to function properly. There are two restrictions on when a site can set a ForceHTTPS cookie to mitigate this issue:

  - The server must set the ForceHTTPS cookie during a non-broken HTTPS session. By establishing a non-broken HTTPS session, the host has demonstrated the ability to conduct secure HTTPS. If the browser permitted ForceHTTPS cookies to be set over HTTP, an active attacker could conduct denial of service beyond his ability to control the user's network.
  - The server must set the ForceHTTPS cookie using the `Set-Cookie` header, rather than using script to set the `document.cookie` property. If script were permitted to set ForceHTTPS cookie, a transient cross-site scripting vulnerability could result in a long-lasting denial of service.

  Even with these restrictions, a shared domain ForceHTTPS cookie could still be used for denial of service: A student hosting content on `https://www.stanford.edu/` could set a ForceHTTPS cookie for `.stanford.edu`, denying service to many Stanford web sites. To prevent this scenario, a ForceHTTPS cookie enables ForceHTTPS only for the host that sent the cookie.

- **Policy Expressiveness.** When a site enables ForceHTTPS, the browser makes several modifications to its behavior at once. Instead, the browser could respect finer-grained policies capable of expressing more specific behavior changes, for example allowing a site to require HTTPS without disavowing mixed content or certificate errors. However, exposing a more expressive policy interface increases the burden on site developers to select the appropriate policy and on browser developers to correctly implement each policy permutation. We reserve the value of the ForceHTTPS cookie for future enhancements to the mechanism.

5.2 Error Handling

Although it provides stricter error handling, ForceHTTPS must be prepared to handle misconfigured clients and servers. If ForceHTTPS simply were to provide a click-through error dialog box, the benefits of the mechanism would be lost. Many users consider clicking through security dialog boxes to be a routine task.

- **Wireless HotSpot.** The most common client error occurs when a user first connects their computer to a wireless hotspot. Before allowing access to the Internet, the hotspot typically redirects all network requests to its registration page. If the user attempts to navigate to an HTTPS site, the hotspot will be unable to present a valid certificate and the connection will generate a certificate error. In this situation, the two options offered by current browsers are both poor. The user can either abandon the request (and not join the network) or can accept the broken certificate, sending their secure cookies to the hotspot registration page.

  To better recover from this error condition, the browser could attempt to connect to a known HTTP page on the browser vendor's web site and compare its contents to a known value. If a redirect is encountered or the contents of the page do not match the expected value, the browser could ask the user if they would like to connect to the wireless network registration page (which consists of the redirected content). This technique permits the registration page to successfully redirect the user without compromising the user's cookies and without revealing any sensitive query parameters (as used by PHP sites that set `session.use_trans_sid` to true and `session.use_cookies` to false).

- **Embedded Content.** When ForceHTTPS is enabled for a host, the browser prevents pages on that host from embedding non-HTTPS content. The security of the site can still be compromised, however, if the site embeds content from an HTTPS connection that encountered a certificate error. For this reason, certificate errors are treated as fatal network errors during any dependent load on a ForceHTTPS page. For content that would appear in a frame, the broken content is replaced with a message indicating that the content could not be loaded securely.

- **Opting Out.** If a ForceHTTPS site persists in being misconfigured, the user can remove the ForceHTTPS cookie through the same user interface used to enable ForceHTTPS. This process requires several steps, i.e. not a single mouse click, and both clears the user's cookies and restarts the browser to prevent any existing browser state from being compromised. We expect that the rate of ForceHTTPS hosts misconfiguration will be significantly lower than the general HTTPS misconfiguration rate because the owners of the ForceHTTPS hosts have indicated (by enabling ForceHTTPS) that they take seriously the security of their sites and do not wish to allow users to connect over broken HTTPS connections. In contrast, users will need to become familiar with the browser's mechanism to bypass standard certificate errors in order to access many misconfigured sites.

5.3 Limitations

Although ForceHTTPS has numerous security benefits, it cannot prevent all attacks. In this section, we describe some vulnerabilities that ForceHTTPS does not address.

- **Attacks on Initialization.** If a user is unable to establish a secure connection to a server, then that server cannot set a ForceHTTPS cookie. An attacker who controls the user's network on *every* visit to a target site can prevent the ForceHTTPS cookie at that site from ever being set. Although the user will be exposed to a large number of warnings, ForceHTTPS will not yet be enabled and thus cannot force the user to make the correct security decision. However, if the user does ever connect to the site securely, the browser enforce security until the ForceHTTPS cookie expires.

- **Privacy.** Like any cookie, ForceHTTPS leaves a trace on the user's system for each ForceHTTPS site visited. Users who are concerned about privacy from

web sites or from other users who use the same system often reject or frequently clear their cookies. By clearing cookies, these users can remove all evidence of the ForceHTTPS cookie. Although they lose ForceHTTPS protection their next visit, the user's decision to purge all browser state associated with the site will make it unlikely that the browser will have second factor authentication tokens for a future attacker to steal. (Note that the preconfigured ForceHTTPS cookies and rewrite rules are the same for each user and do not reveal the user's browsing behavior other than to identify them as a ForceHTTPS user.)

- **Developer Errors Other Than Mixed Content.** By enabling ForceHTTPS, the web developer opts in to more stringent error processing, but the developer still compromise the security of his or her site by making mistakes. We list a few common mistakes of this sort to remind the reader that ForceHTTPS (and more generally encryption) is not a panacea.
  - **Cross-Site Scripting (XSS).** ForceHTTPS provides no protection if the site contains a cross-site scripting vulnerability. Such a site is completely vulnerable to a web attacker.
  - **Cross-Site Request Forgery (CSRF).** Similarly, ForceHTTPS does not protect a site that contains a cross-site request forgery vulnerability [14]. CSRF vulnerabilities often give attackers the ability to issue commands from the user's browser.
  - **HTTP Response Splitting.** If the server does not properly sanitize carriage returns and other whitespace in input included in HTTP response headers, an attacker can inject headers (and potentially scripts) into HTTP responses. An HTTP response splitting vulnerability can often be used to manipulate ForceHTTPS cookies.
  - **document.domain.** A site that sets its domain to a value must trust all the hosts with that value as a suffix. These hosts can enter the site's security sandbox and script its pages.
- **Plug-ins.** Analysis of browser security features must take plug-ins into account because plug-ins such as Flash Player and Java are widely deployed and can often provide attackers an alternate route to circumventing a security mechanism. ForceHTTPS must ensure that browser network requests on behalf of plug-ins, which carry the user's cookies, enforce the ForceHTTPS restrictions. Furthermore, all cookie management by plug-ins must respect the ForceHTTPS policy. If the plug-in allows the site to make direct network requests using raw sockets, it cannot be forced to use HTTPS without breaking backwards compatibility. We consider it the web site's responsibility to provide appropriate encryption of the raw socket traffic if necessary; ForceHTTPS does not provide protection from the imperfect developer in this case.
- **Complexity of Rewrite Rules.** As we describe in Section 5.5, the rewrite rules required to enable ForceHTTPS at a legacy web site can range from very simple to impossible. A site could become vulnerable if rewrite rules are introduced that redirect sensitive information to an attacker. Rewrite rules can also break functionality at the web site, rendering certain pages inaccessible or issuing unauthorized transactions. If the web site changes significantly, or the site decides to change its support for HTTPS, the rewrite rules might need to be updated. We consider the installation and editing of rewrite rules to be a decision with serious security consequences, similar to installing a browser plug-in. The addition of new rewrite rules is a feature primarily for advanced users.

5.4 Other Policy Advertisement Mechanisms

Other mechanisms that could be used for advertising a ForceHTTPS policy include DNS records and XML files.

- **DNS.** In the HTTP Service Security Requirements (HTTPSSR) proposal [22], a site can indicate its desire for HTTPS by including an HTTPSSR record in DNS. The proposal relies on DNSSEC to prevent a network attacker from manipulating this record. Although the HTTPSSR proposal does not address mixed content, certificate error user interfaces, or cookie security, it could be extended to do so. The DNS policy advertisement mechanism has a number of advantages:

  1. The secure initialization step is not required. The browser can obtain the ForceHTTPS policy on the first visit to the site, even if the network is compromised.
  2. The browser is not required to maintain any persistent state associated for each host, preventing state exhaustion attacks.
  3. HTTP response splitting attacks do not allow an attacker to manipulate ForceHTTPS policies.

  Unfortunately, DNSSEC is not widely deployed. Without DNSSEC, sites can store their ForceHTTPS policies in DNS using the stateful, secure-initialization approach of ForceHTTPS cookies. To support this approach, HTTPSSR records would need to include an "expires" field. The Time-To-Live (TTL) supplied by DNS is not suitable for storing policy expiry because it provides a maximum, rather than a minimum, duration for the validity of the record.

- **XML.** Using the XML paradigm, a site can advertise its ForceHTTPS policy in an XML document hosted over HTTPS at a well-known location. This technique is used by Adobe Flash Player to determine if a server is willing to receive cross-domain URL requests. Adobe's `crossdomain.xml` policy file could be extended to advertise a `ForceHTTPS` policy:

```
<?xml version="1.0" ?>
<cross-domain-policy
    xmlns:f="http://www.forcehttps.com/">
  <allow-access-from
      domain="*.stanford.edu" />
  <f:forcehttps
      expires="Mon, 11 Feb 2009 23:39:27 GMT"/>
</cross-domain-policy>
```

The browser will enable ForceHTTPS for that site for the duration specified by the `expires` attribute of this element. This element can be included in existing `crossdomain.xml` files using a unique XML namespace for the element. This approach has the advantage that a site must already control the contents of its `crossdomain.xml` file in order to be secure against attacks using the Flash plug-in. Additionally, using XML to store policy information makes it possible to extend this policy advertisement mechanism to include future security policies.

5.5 Example Rewrite Rules

In creating our prototype implementation of ForceHTTPS, we developed rewrite rules for seven popular sites to understand the subtleties in deploying ForceHTTPS. To develop the rewrite rules, we installed the ForceHTTPS extension and enabled ForceHTTPS for each site we wanted to support. We then turned on client-side error logging and tried to log in and log out on each site. Using the error messages we identified HTTP content that could be served over HTTPS and used rewrite rules to transform those HTTP requests into HTTPS. The results are summarized below.

- **PayPal.** We did not need specialized rewrite rules for `paypal.com`, which serves all content on its main site over HTTPS. We also enabled ForceHTTPS for `paypalobjects.com`, where PayPal's static scripts and stylesheets are hosted. This precaution is necessary for Firefox 2, which prompts users to override certificate errors for embedded content, but is no longer necessary in Firefox 3, which blocks such content automatically.

- **American Express.** American Express uses SWF movies to load HTTP files to display advertisements, but the insecure files are served from a different domain (`doubleclick.net`) and cannot script the main American Express page.

- **Fidelity.** Fidelity uses SWF movies that load HTTP files to display stock quotes, but these requests do not require cookies, so no rewrite rules are necessary. Fidelity hosts a `crossdomain.xml` file that allows access from `*.fidelity.com` and `*.fmr.com`. Thus, to be protected from network attackers, Fidelity needs a ForceHTTPS cookie for both `.fidelity.com` and `.fmr.com`.

- **Bank of America.** Bank of America uses both HTTP and HTTPS on its main home page, and certain pages require cookies to be sent over HTTP. However, the login page and online banking are handled on subdomains, such as `sitekey.bankofamerica.com`. These subdomains use HTTPS exclusively, so we set ForceHTTPS cookies for the online banking subdomains.

- **Gmail.** Google's Gmail web site, `mail.google.com`, presents a challenge because the site sets a domain-wide `.google.com` cookie. We enabled ForceHTTPS for the entire Google site and wrote rewrite rules to redirect all Google pages to HTTPS except the search page (which cannot be accessed over HTTPS). Additionally, we rewrote a query parameter for the login page to indicate that we wished Google to mark its session cookies `Secure`. It is important to redirect all pages (except search) to HTTPS because Google's login page sometimes transmits sensitive authentication information in URL parameters. With ForceHTTPS enabled, search traffic at Google is not protected from eavesdropping, but no cookies are sent with this traffic, keeping the user's session identifier secure.

- **Chase.** Chase refuses to serve its home page over HTTPS. We chose to redirect `http://www.chase.com/` to `https://chaseonline.chase.com`, allowing the user to log in securely, but preventing access to any news or special offers that appear only on the Chase home page. ForceHTTPS also automatically repairs mixed content on Chase's login page by redirecting an insecure SWF movie to HTTPS.

- **Yahoo! Mail.** We were unable to develop rewrite rules for the Yahoo! Mail site because Yahoo! Mail does not support HTTPS. We enabled ForceHTTPS for the Yahoo! login page, with the goal of protecting the user's password (rather than the session) from active attacks. Because the Yahoo! Sign-in Seal [30] is revealed by an insecure cookie, an active attacker could display the sign-in seal on an HTTP page without requiring the user to click through a security warning dialog. With ForceHTTPS installed, the attacker cannot display the Sign-in Seal, upgrading Yahoo!'s phishing defense to a pharming defense as well.

6. CONCLUSIONS AND FUTURE WORK

ForceHTTPS lets users and web sites to opt in to stricter error processing by the browser. For users, ForceHTTPS can fix vulnerabilities in web sites and enable sites that were not designed to be used over hostile networks to be browsed securely over such networks. For web sites, ForceHTTPS protects `Secure` cookies from active network attackers and remediates accidental embedding of insecure content.

Previous anti-pharming proposals required either overhauling DNS or the deployment of complex, digitally signed policy files encoding the frequently-changing trust relationships between domains. By contrast, ForceHTTPS merely requires setting a cookie, a procedure that many sites already handle with every new session.

ForceHTTPS is a useful mitigation for mixed content, but sites should strive to fix these bugs by removing insecure embeddings. Developers have trouble detecting mixed content because all the major browsers have significant bugs in their mixed content detection mechanisms. In future work, we plan to collaborate with web application vulnerability scanner vendors to build a mixed content scanner that spiders a web site and reports its mixed content vulnerabilities.

ForceHTTPS has already proven itself useful to its authors, who now check their email at security conferences without fear of eavesdropping and other network attacks. We look forward to extending this protection to other users.

Acknowledgements

We thank Michael Barrett, Dan Boneh, John C. Mitchell, Umesh Shankar, and Andy Steingruebl for their helpful suggestions and feedback. This work is supported by grants from the National Science Foundation and the US Department of Homeland Security.

7. REFERENCES

[1] Bank of America SiteKey. http://www.bankofamerica.com/privacy/sitekey/.

[2] A. Barth, C. Jackson, and J. C. Mitchell. Session swapping: Login cross-site request forgery, March 2008. Manuscript.

[3] M. Beltzner et al. Create preference which restores per-page ssl error override option for it professionals. https://bugzilla.mozilla.org/show_bug.cgi?id=399275.

[4] Chase. Increased security. http://www.chase.com/ccpmapp/shared/assets/page/occ_alert.

[5] R. Dhamija, J. D. Tygar, and M. Hearst. Why phishing works. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI)*, 2006.

[6] DNS Security Extensions. http://www.dnssec.net/.

[7] E. W. Felten, D. Balfanz, D. Dean, and D. S. Wallach. Web Spoofing: An Internet Con Game. In *20th National Information Systems Security Conference*, October 1997.

[8] R. Fielding. Relative Uniform Resource Locators. IETF RFC 1808, June 1995.

[9] C. A. B. Forum. Extended validation certificate guidelines. http://cabforum.org/EV_Certificate_Guidelines.pdf.

[10] R. Graham. Sidejacking with Hamster, August 2007. http://erratasec.blogspot.com/2007/08/sidejacking-with-hamster_05.html.

[11] F. Hecker et al. Improve error reporting for invalid-certificate errors. https://bugzilla.mozilla.org/show_bug.cgi?id=327181.

[12] C. Jackson and A. Barth. ForceHTTPS Firefox extension, 2008. https://crypto.stanford.edu/forcehttps.

[13] T. Jim, N. Swamy, and M. Hicks. BEEP: Browser-enforced embedded policies. In *Proceedings of the 14th International World Wide Web Conference (WWW)*, 2007.

[14] N. Jovanovic, E. Kirda, and C. Kruegel. Preventing cross site request forgery attacks. In *Proceedings of the IEEE International Conference on Security and Privacy for Emerging Areas in Communication Networks (Securecomm)*, 2006.

[15] C. Karlof, U. Shankar, J. D. Tygar, and D. Wagner. Dynamic pharming attacks and locked same-origin policies for web browsers. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS 2007)*, November 2007.

[16] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic. Noxes: A client-side solution for mitigating cross site scripting attacks. In *Proceedings of the 21st ACM Symposium on Applied Computing (SAC)*, 2006.

[17] D. Kristol and L. Montulli. HTTP State Management Mechanism. IETF RFC 2109, February 1997.

[18] G. Maone. NoScript. http://noscript.net/.

[19] G. Markham. Content restrictions. http://www.gerv.net/security/content-restrictions/.

[20] C. Masone, K.-H. Baek, and S. Smith. Wske: Web server key enabled cookies. In *Proceedings of Usable Security 2007 (USEC '07)*.

[21] M. Pilgrim. GMailSecure, 2005. http://userscripts.org/scripts/review/1404.

[22] S. E. Schechter. Storing HTTP security requirements in the domain name system, April 2007. http://lists.w3.org/Archives/Public/public-wsc-wg/2007Apr/att-0332/http-ssr.txt.

[23] S. E. Schechter, R. Dhamija, A. Ozment, and I. Fischer. The emperor's new security indicators. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*.

[24] Security Space and E-Soft. Secure server survey, May 2007. http://www.securityspace.com/s_survey/sdata/200704/certca.html.

[25] S. Stamm, Z. Ramzan, and M. Jakobsson. Drive-by pharming. Technical Report 641, Indiana University Computer Science, Decenber 2006.

[26] A. Tsow. Phishing with consumer electronics – malicious home routers. In *Models of Trust for the Web Workshop at the 15th International World Wide Web Conference (WWW)*, 2006.

[27] A. Tsow, M. Jakobsson, L. Yang, and S. Wetzel. Warkitting: the drive-by subversion of wireless home routers. *Journal of Digital Forensic Practice*, 1(2), November 2006.

[28] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. Cross site scripting prevention with dynamic data tainting and static analysis. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2007.

[29] Wireshark: What's on your network? http://www.wireshark.org/.

[30] Yahoo! Inc. What is a sign-in seal? http://security.yahoo.com/article.html?aid=2006102507.