HTTPS Can Byte Me

Robert "RSnake" Hansen and Josh Sokol

05/17/2010

Overview

HTTPS was created to protect confidentiality and prove integrity of content passed over the internet. It has become the de-facto standard for e-commerce transport security. Over the years a number of exploits have attacked the principle, underlying PKI infrastructure and overall design of HTTPS. This paper will drive another nail in the HTTPS coffin through a number of new exploitation techniques leveraging man-in-the-middle attacks; the goal of which is to break confidentiality and integrity of HTTPS traffic. The impact of these flaws suggests a need for changes in the ways we protect the transmission of data online.

History

Taher Elgamal, the principle architect behind the SSL protocol once said, "I think all of these problems have to do with browser design rather than security or protocol. It's interesting because SSL gets blamed for all the stuff, but [they are] actually not even related to SSL." This statement sums up much of the past thinking on HTTPS security. While there have been many flaws in the browser and PKI that have lead to exploitation, it is a common perception that SSL/TLS itself has had no major flaws within it. However, if you look at the versions alone, this is not true.

SSL was not even adopted by the browser community until 2.0. SSL 3.0 fixed a number of significant flaws within the browser. Additionally, SSL 3.0 has been replaced by TLS, which itself is already up to revision 1.2. These failings alone cannot be ignored; proving that there is a long history of flaws in SSL/TLS from its earliest inception and throughout its history. Additionally, there is a current flaw that allows an attacker to break the integrity of the SSL/TLS stream if the server allows renegotiation.

To Taher Elgamal's credit, there have been a number of flaws in the browser and surrounding PKI infrastructure~~,~~ and regarding the surrounding trust. For instance, there is an overly permissive number of root certificate authorities built into the majority of modern browsers; creating a default configuration of implicit trust in hundreds or thousands of people. The potential abuse of this trust is high. In fact, companies like Packet Forensics have recently been outed by the FTC as being able to man-in-the-middle an SSL connection, presumably using an existing trusted root certificate. The reseller community is vast, and less regulated as well. This greatly expands the threat landscape. In fact, there have been disclosed vulnerabilities in the applications that these resellers use for SSL procurement and user verification.

There have been attacks against the algorithms used to generate SSL/TLS keys as well. For instance, using a large cluster of computing resources, a group of researchers (lead by Alexander Sotirov) was able

to generate their own root signing authority that computed to the same hash as another root certificate in the browser. Using this attack, it would have been trivial for this group to take over any SSL/TLS stream that allowed signing by that root authority, breaking the explicit trust that the CA was created to provide.

There have been a number of tools released to attack poor implementations of SSL/TLS on the web, including SSLStrip (by Moxie Marlinspike) and Cookiemonster (by Mike Perry). SSLStrip piggybacks both on user ignorance and shoddy website design to downgrade the user from SSL into an unencrypted HTTP stream. Cookiemonster uses the fact that HTTP can read cookies that are created in HTTPS space, as long as they aren't marked as "secure". Both of these tools are very effective at bypassing confidentiality, one of the two tenants that HTTPS was originally designed for.

UI issues are also a large point of contention, as it is unclear if users even understand what SSL/TLS is in any meaningful way; if they can tell when they are actually in it, and if they understand what the associated warnings could be. As such, it may be entirely unnecessary for an attacker to even bother circumventing HTTPS at all, but rather, generate a snake-oil certificate as the user will most likely ignore the warnings.

There are many more attacks as well, like attacking the unencrypted nature of the user authentication step for SSL/TLS issuance, attacks against webmail providers where ssladmin@ can be registered by an attacker, and so on. The major take-away from this is that SSL/TLS cannot be decoupled from the rest of the browser and the surrounding environment in the HTTPS context. They are one in the same.

Further Exploits

When a browser creates a SSL/TLS connection to a website, it is typically not a straightforward event. The user quite frequently does not manually type in the full URL themselves or follow a bookmark. That said, there are a number of places an attacker can get in the middle of the data stream and provide erroneous information. DNS itself is plaintext, as is HTTP. In this way it is quite easy to deny service to the HTTPS-enabled site and force the user to use the HTTP site. In this scenario the user might presume that the HTTPS site is down or the link has been changed.

There is existing research (by Shuo Chen, Rui Wang, XiaoFeng Wang, Kehuan Zhang) that shows that it is fairly easy, given certain environments, to parse apart the HTTPS traffic and understand what the original traffic contained. There is a relational property between the HTML and the (sometimes 3rd party) embedded content. This can be analyzed further to understand relational aspects of the payloads, and understand both in which direction traffic is flowing and what is being requested.

Adding onto this, determining the difference between cached and un-cached embedded objects can help reduce the noise of the total amount of HTTPS content. By pre-loading this information or verifying that it has already been cached, the attacker can determine where the user has been within the HTTPS enabled website.

One of the greatest flaws within the browser is that it is quite possible to map out where users have been on the website~~,~~; even despite the same origin policy. If the attacker can map out the website

ahead of time, it is possible to know where pages are located, what their sizes are, page flow, and more. Using this information it is possible to piggyback on existing browser flaws, like the CSS history hack, timing attacks (E.g. the issues found by Chris Evans), using the browser's JavaScript history variable and so on.

It is possible to abuse the nature of cookies as well. Cookies have a more permissive policy than JavaScript or other variables in that it can be set and read in other protocols, other sub-domains, and so on, depending on certain variables. This fact has been exploited to read non-secure HTTPS cookies, but there are other attacks here as well. For instance, in the HTTP stream it is possible to set a cookie that will be read by the HTTPS site, and even overriding existing cookies. This can lead to a great number of different attack variants.

Some variants of this attack include automatic logout, cross site scripting, redirects, and session fixation. Depending on how the site is built, how the cookie is formed, and other attributes, it is possible to use the nature of the cookie's variant of the same origin policy to wreak havoc; breaking integrity within the browser session. Other attacks against the HTTPS stream using cookies can lead to server side denial of service of particular pages, directories, and so on. Because HTTPS can read cookies set within HTTP space, it is rather easy to set cookies in such a way that it has detrimental effects on a large percentage of default HTTPS websites.

Autocomplete represents another information leakage vulnerability. During the process of typing in the URL path, which could include things like basic authentication or typos, that information is quite often leaked out to third parties, like Google, over unencrypted connections, and can include incredibly sensitive information like internal corporate domain names that do not resolve externally and so on.

Prefetching is yet another information leak that an attacker can leverage. Browsers like Chome make DNS requests off of domain links that they find within HTTPS sites, which can be viewed on the wire, regardless of if the user ever intended to click the links in question.

Lastly, tabbed browsers represent a unique potential for abuse in that individual tabs can interact with other tabs, injecting content into them. While information cannot be read cross domain, that does not necessarily matter to an attacker. Delayed popups can wreak havoc on a user's perception of the site and cause them to perform actions that they shouldn't; breaking browser confidentiality or worse.

Conclusions

HTTPS cannot guarantee confidentiality and integrity within the browser context. Because of how complex modern browsers are and how many different conduits an attacker has, it is incredibly difficult to protect the data stream from tampering and disclosure. Using proper tab isolation, better cookie management in the browser and better "white-noise" generation in the SSL stream could help prevent the majority of these attacks presented. However, given the sheer number of vulnerabilities within SSL/TLS in the browser, HTTPS should not be trusted for any high value transactions.