6.858 Final Project

Changping Chen, Xinyue Deng, Xiaomin Wang
ccp0101, dxy0420, xiaominw

Introduction

In recent years we have seen an increasing number of man in the middle (MITM) attacks from sophisticated attackers who have access to browser-trusted certificate authorities. An example of this is state actors that can target few activists to monitor their traffic. Such targeted attacks are likely go unnoticed. Our goal is to uncover such attacks by comparing server certificate with the one users have received on their first visit, assuming their first visit is over a secure, trusted network.

As a broad overview, we store server certificate while users access websites. When MITM attack occurs, the certificate is bound to change and we can alert users if that happens. Our implementation has to split our program into two parts, a chrome App and a chrome extension. This is to bypass some security restrictions imposed on third party developers. When a user visits a website, chrome notifies our extension about a pending request. Our extension forwards the URL to our app, which then initiates a TLS handshake to retrieve server certificate. Then it checks for changes and notifies our extension if the certificate has changed. Then our extension can present this information to users through extension UI.

Assumptions

1. Users are aware of basic security concepts
2. Users want to avoid MITM without any noticeable penalty on browsing experience
3. MITM attacks against a website could only be in effect for a short period of time in the entire course of user's interaction with a website
4. Attackers are sophisticated: they can compromise a certificate authority, either legally (state actors) or illegally (online accounts / credit card stealers)
5. No modification required from server administrators, a.k.a. 100% compatibility
6. Our applications (browser, extensions, and more) are secure

Our ideal users are security-sensitive users, e.g. activist, journalists and political dissidents. They fall into the categories of users that generally have some security knowledge and are very active in seeking ways to protect their internet privacy.

The Open Web Application Security Project (OWASP) has excellent resource on certificate pinning[1]. It has a discussion on what to pin, public key or certificate. We believe in ideal case, change in public key should carry more weight on determining if MITM has likely happened. However, in our prototype with limited time, we have decided to pin on certificate issuers from

[1] https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning

user's first visit, since we believe that the attackers are unlikely to compromise the exact intermediate CA that issued the legitimate certificate.

Implementation

The broad idea of our implementation is to retrieve certificate with TLS handshake in Javascript. For security reasons, Chrome does not expose an API that allows us to tap into its TLS-layer informations. To overcome the restrictions, we only partially implemented TLS handshake protocol, specifically the construction of initial "Client Hello" and parsing of "Server Certificate". By overlaying our TLS implementation on built-in API "`chrome.sockets.tcp`", we can obtain server certificate with minimal latency.

Our existing implementation has a few design flaws. Certainly it is unable to interrupt a connection until MITM. Moreover, the default browser behavior is to reuse a negotiated TLS session in many TCP connections to reduce overhead. However, the extra TCP connections initiated by our program would negotiate a second TLS session immediately after, which reveals our program in place. An adversary can exploit this information to selectively man-in-the-middle the first TLS session, and directly forward the second session to the legitimate server, thereby returning a different certificate. Performance-wise, additional TCP connections might negatively impact the browser performance because it usually maintains a connection pool of limited size.

In addition, we have implemented a few optimizations to delay and optionally block user request if server certificate has changed. First, we block requests to previously invalidated hosts, using "blocking" option in "`chrome.webRequest`". We also implemented a light-weight HTTP server within our Chrome app, and redirect non-embedded URLs to our server, e.g. "`http://localhost:port/validate?url=http://www.facebook.com`". Our server waits until website certificate is validated and redirect user back to the original website using HTTP status code `302 Moved Temporarily`. If the website certificate has changed, we respond with an error.

These optimizations allow us to have some control over a user's network flow. However, in practise, it's difficult to implement these features correctly, without disrupting user's experience, such as Same-Origin policies that would be violated when an embedded secure resource is redirected over an insecure channel. We have disabled our optimizations for now.

To install, enable Developer Mode in Chrome Extensions page. Drag the `app` and `ext` folders from our project, onto the page separately. Then a user can click on the extension icon to view verification results of all secure websites accessed in the current tab.

Alternative Implementation

We have considered implementing a Socks5 proxy in a Chrome Packaged App and parse TLS on all network flow. It's similar to inspecting network flows with Wireshark, but with the option to modify and drop connections. This does not send extra network traffic, so it mitigates

the issue with existing implementation that adversaries can fool our program. With "`chrome.proxy`" API, we could force to proxy all secure traffic via our program. However one caveat here is that we need to implement a simple to use interface by which users can specify upstream proxies, such as Tor, and a bad implementation in upstream proxy interaction can reveal user identity.

We believe that our applications if deployed by many could reveal broader issues with traditional PKI. If our users represent a substantial sample size of entire Internet users, in theory we could attempt to include more risk heuristics beyond the "I trust you therefore I trust him" approach of PKI. However, we do not have enough sample size to actually carry out our ideas.

Future Work

We have built a database to store certificates uploaded by users. Part of the database can be found at http://ccp0101.scripts.mit.edu/6858/. What we would like to explore more is Web of Trust with majority rule. Given the certificate records gathered from all over the world, we may be able to answer questions such as "is this certificate and public key that I received from BOA.com the same from what others received". We can also do things like identify public WiFi with paywall redirection (airport, hotel, cafe, etc.) or the network entity behind MITM attacks (an ISP, company, individual, etc.). We can even gauge the impact of compromising a particular CA, or detect and provide evidence of rogue CAs.

Conclusion

We have implemented a prototype application for uncovering sophisticated MITM attacks and detecting rogue CAs for Chrome browser. It has the advantage of being lightweight, informative, and simple to install via Chrome Web Store. The shortcomings are unable to prevent attack and may be noticed by attackers.