Work in progress

This is an early draft of a chapter of a book on the foundations of cryptographic authentication being coauthored by Francisco Corella, Sukhi Chuhan and Veronica Wojnas. Please send comments to the authors.

4. Phishing resistant authentication with cryptographic credentials

Chapter summary

At the beginning of the web, SSL client certificates were ready to be used for cryptographic authentication of users to websites, but they failed to be deployed. They were replaced with authentication by username and password, which is easier to implement for the relying party because verification is performed by application layer code rather than transport layer code.

As attackers became adept at exploiting the vulnerabilities of passwords to phishing, reuse, and backend breaches, websites deployed, and users were asked to use, two-factor authentication. But in 2017, the Evilginx attack demonstrated that the protection provided by 2FA is easily defeated by using as a phishing site a reverse HTTP proxy that relays traffic between the browser and the website and captures the session cookie after successful login.

The variety of cryptographic authentication initiatives surveyed in this book are a recognition that the time has come to try again cryptographic authentication. But this time it could be deployed the application layer. The same public certificates usable at the transport layer as TLS client certificates can also be presented at the application layer using a simple challenge-response protocol, with protection against man-in-the-middle phishing attacks by inclusion of an "audience", or "verifier designation" along with the challenge in the signed data.

4.1 A false start

The internet was designed in the 1960s to withstand attacks from nation-states, but not the kind of attacks that are now called cyberattacks. It was designed to withstand nuclear

attacks. Instead of being a monolithic network as specified by the OSI model, the internet is an internetwork, a network of networks, parts of which can continue to work after other parts have been physically destroyed.

Cyberattacks were not a concern because internet users were expected to be trusted government employees or academics. The threat model changed when Tim Berners Lee invented the World-Wide Web became a universal, all-purpose communication medium. Fraud became easier on the internet than it was before the internet, and nation-states hostile to the US were using the internet instead of threatening to attack it with ICBMs.

The internet community was quick to perceive the change and address the new threat mode. Using the recently discovered concept and techniques of asymmetric cryptography, the community designed and the IETF standardized an impressive array of security protocols, including IpSec, SSH, and SSL/TLS. And those protocols were successfully deployed worldwide.

But something could not be deployed: cryptographic authentication of individual human users. There was no technical reason for this failure. SSL had client and server certificates and browsers were ready to be used as wallets for client certificates. S/MIME had all the ingredients to provide encrypted and authenticated communication between authenticated parties. Cryptographic authentication was ready to go but it failed to start. This book is about the efforts that are now being made to start again.

4.2 Passwords and 2FA

Since authentication was essential for ecommerce and client certificates were not being deployed, a solution was found. SSL/TLS (henceforth we shall just say TLS) was used as a channel that provided encryption and data authentication between the client and the server, but only the server presented a certificate.

Encryption ensured that an eavesdropper could not obtain a password or a credit card number sent by the client. Data authentication implied to the client that it was receiving data that had been sent by a party that knew the private key associated with the certificate that the server had presented when performing the TLS exchange. To the server, it implied only that it was receiving data that had been sent by the party with which the server had performed the TLS handshake. But when the client later sent a password over the channel, the server knew that the password had been sent by that same party.

Actually, the channel that ended up being used was not just TLS with unilateral server authentication, it was HTTP over TLS with unilateral server authentication. Passwords and credit card data were sent in the body of HTTP Post requests addressed to https URLs. This is a very convenient channel because it is an application layer channel rather than a transport layer channel. We shall come back to it in Section 4.4 of this Chapter, and then in Section 12.4 of chapter 12 when we see how browsers can be used as application layer credential wallets.

This solution enabled ecommerce and websites proliferated. Users were burdened by having to remember large numbers of passwords, but this problem was solved, first by password managers, then by browsers playing the role of password managers.

However, authentication by username and password has two serious drawbacks.

First, it only provides returning-user authentication. It cannot be used to demonstrate to relying party that the user has attributes certified by a third party. One solution to this problem is Federated Identity Management, discussed in Chapter 5, where the relying party redirects a login request to a trusted identity provider, to user authenticates to the IdP as a returning user, and the IdP redirects the user's browser back to the relying party, conveying user attributes with user consent. This has a privacy drawback, as the IdP observes the user's logins to relying parties. And the user has to login to the relying party with an identifier controlled by the IdP, which is objectionable on grounds that have motivated the concept of self-sovereign identity.

Second, passwords have severe vulnerabilities that attackers became adept at exploiting:

1. If a user reuses a password at a malicious website set up by an attacker, the attacker can use that password to impersonate the user.
2. A breach of a backend database containing salted passwords can be exploited by a dictionary attack that will reveal many weak passwords.
3. Passwords can be easily captured by phishing attacks.

A second authentication factor is a countermeasure against all three of these vulnerabilities, so 2FA was implemented by many relying parties and, for several years, cybersecurity advocates tried hard to convince users to use it. But then, it was discovered that the security provided by 2FA was easily defeated.

4.3 The Evilginx attack

In 2017, Kuba Gretzky published [1] and demonstrated on GitHub [2] an attack against a website where login required authentication by username and password followed by two-factor authentication.

The attack used as a phishing site a commercial reverse HTTP proxy. In normal use, a reverse HTTP proxy is used at the edge of a corporate intranet, where it relays TLS-protected HTTP requests and responses between clients in the external network and servers in the internal network. More precisely, the proxy receives an HTTP request over an external TLS connection, determines which internal server is the intended recipient of the request using ad-hoc rules, sets the value of Host header in the HTTP request to the DNS name of the recipient, and forwards the request to the recipient over an internal TLS connection. The recipient sends the HTTP response over the internal connection, and the proxy relays the response to the client over the external connection.

In a phishing attack against a user who logs in to a website using a browser, a phishing proxy similarly relays HTTP requests from the browser to the website and HTTP responses from the website to the browser. Before relaying each request, the proxy sets the value of the Host header to the DNS name of the recipient, as it would in normal use when forwarding a request to an internal server. In a difference from normal use, before relaying an HTTP response from the website to the browser, the proxy replaces the DNS name of the website with the DNS name of the proxy in all the URLs that it finds in the body of the response. Requests sent to those URLs will thus go through the proxy instead of going directly to the website. We shall refer to these modifications of HTTP requests and responses as "fixing the Host header" and "fixing the URLs" respectively.

Details of the attack

The details of the Evilginx attack, extrapolated from a brief description in [1], could be as follows:

1. The user clicks on an https link to the phishing proxy, thinking the link targets the website. (All URLs are assumed to use the https scheme.)
2. The user's browser sends an HTTP Get request to the proxy over a browser-to-proxy TLS connection. The proxy fixes the Host header and forwards the modified request to the website over a proxy-to-site TLS connection.
3. The website sends the HTTP response to the request of step 2 over the proxy-to-site TLS connection.
4. The proxy receives the HTTP response of step 3 from the proxy-to-site TLS connection of step 2. The body of the response is the HTML source code of the home

page of the website, which has an https link to the login page. The proxy fixes the URLs in the HTTP response, causing the link to point to the phishing proxy.

5. The proxy forwards the modified HTTP response to the browser over the browser-to-proxy TLS connection of step 2. The browser renders the home page, and the user clicks on the login link.
6. The browser sends an HTTP Get request to the proxy over a browser-to-proxy TLS connection. The proxy fixes the Host header and forwards the modified request to the website over a proxy-to-site TLS connection.
7. The website sends the HTTP response to the request of step 6 over the proxy-to-site TLS connection.
8. The proxy receives the HTTP response of step 7 from the proxy-to-site TLS connection of step 6. The body of the response is the HTML source code of the login page of the website, which has a login form with fields for the username and the password. The proxy fixes the URLs in the HTTP response, causing the URL of the action attribute of the form to target the proxy.
9. The proxy forwards the modified HTTP response to the browser over the browser-to-proxy connection of step 6. The browser renders the login page, and the user submits the login form after filling in the username and the password.
10. The browser sends an HTTP Post request with the username and the password to the proxy over a browser-to-proxy TLS connection, and the proxy captures the username and the password. However, since the username and the password are not enough to log in without an authentication code, THE PHISHING ATTACK HAS NOT BEEN SUCCESSFUL YET.
11. The proxy fixes the Host header and forwards the HTTP Post request to the website over a proxy-to-site TLS connection.
12. After the website verifies the username and password, it sends to the user an authentication code through an out-of-band channel such as SMS or email, which the attacker does not have access to. At the same time, it sends an HTTP response to the HTTP Post request over the proxy-to-site TLS connection of step 11.
13. The proxy receives the HTTP response of step 12 from the proxy-to-site TLS connection of step 11. The body of the response is the HTML source code of a page containing a 2FA verification form with a field for entering the authentication code. The proxy fixes the URLs in the HTTP response, causing the URL of the action attribute of the 2FA verification form to point to the proxy.
14. The proxy forwards the modified HTTP response to the browser over the browser-to-proxy TLS connection of step 10 and the browser renders the page containing the 2FA verification form.
15. After receiving the out-of-band message, the user enters the authentication code into the 2FA verification form and submits the form.

16. The browser sends an HTTP Post request with the authentication code to the proxy over a browser-to-proxy TLS connection. The proxy does NOT capture the authentication code, which wouldn't be useful since a different authentication code is used for each login request. Although the proxy has captured the username and password and could have captured the authentication code, THE ATTACK HAS NOT BEEN SUCCESSFUL YET.
17. The proxy fixes the Host header and forwards the Post request to the website over a proxy-to-site TLS connection.
18. The website verifies the authentication code, logs the user in, and sends an HTTP response with a login session cookie over the proxy-to-site TLS connection.
19. The proxy receives the HTTP response of step 18 from the proxy-to-site TLS connection of step 17. The cookie has a Set-Cookie header [3] whose value is the login session cookie being set. The phishing proxy captures the cookie, and the attacker can use it to impersonate the user as described below. THE ATTACK AS NOW SUCCEEDED.
20. The value of the Set-Cookie header comprises a string of the form <cookie-name>=<cookie-value> followed by cookie attributes, which typically comprise the HttpOnly predicate, which prevents JavaScript code from accessing the code, and the Secure predicate, which forbids sending the cookie over an unencrypted connection. The cookie may also have a Domain=<domain-value> attribute. If so, the proxy removes the attribute.
21. The body of the response that the proxy received in step 19 is the HTML source code of a web page telling the user that the login has been successful and containing links to private pages of the site that the user is now able to access. The proxy fixes the URLs and forwards the modified response to the browser.
22. The browser renders the page and stores the cookie received in the Set-Cookie header of the response. Since the Set-Cookie header had no domain attribute, the browser assigns the default domain to the cookie, which is the origin of the page, i.e., in this case, the domain of the proxy.
23. The user clicks on a link to a private page. Since the URLs have been fixed, the browser sends an HTTP Get request to the proxy over a browser-to-proxy TLS connection, and since the domain of the cookie is the domain of the cookie is the domain of the proxy, the browser sends the cookie with the request. The proxy will thus be able to continue monitoring the interactions of the user with the website.

Using the captured cookie for impersonation

The attacker can impersonate the user by installing the captured cookie in the attacker's own browser using a browser extension called a "cookie editor". Every browser keeps

track of the set of cookies that have been set by a given domain. In normal use of the browser, only servers in that domain can add cookies to the set. However, a cookie editor allows the user of the browser to visit a site, display the set of cookies that have been set for the site, edit them, and add a new cookie to the set by manually typing in its name and its value, and adding any desired attributes. The original description of the attack [1] referred to the EditThisCookie Chrome extension [4].

So, after capturing the login session cookie, the attacker can install a cookie editor in a browser, visit the home page of website, view the set of cookies set by the website, and add a new cookie to the set by copy-and-pasting the name and value of the captured cookie and any desired attributes. The attacker will then be able to visit private pages of the site without having to log in.

The value of the captured cookie is a reference to a session record in the database of the website, which itself is a reference to the legitimate user's record in the database. The session record will eventually expire, and the attacker will no longer be able to use the captured cookie to visit private pages. However, before that happens, the attacker can visit the profile page of the legitimate user and change the login method, thus taking over the account.

4.4 Cryptographic authentication at the application layer

As can be seen by the above details, the Evilginx attack can be readily implemented and will succeed against any website where the use logs in with username and password followed by a second factor sent over any out-of-band challenge. 2FA is therefore obsolete. The variety of cryptographic authentication initiatives surveyed in this book are a recognition that the time has come to try again cryptographic authentication after the false start of SSL certificates.

A reason to be optimistic about the chances of success of cryptographic authentication is that it uses a stronger defense against phishing attacks than 2FA. Two-factor authentication does not remedy the vulnerabilities of passwords. It adds a second factor, but the password used in 2FA is vulnerable to reuse, dictionary attack after breach of the password database, and capture by phishing attack. By contrast, the private key associated with a traditional public key certificate or a selective disclosure certificate, and the secret signature on user attributes by the issuer of an anonymous credential, are not vulnerable to capture because they never leave the credential wallet.

TLS client certificates are a transport layer authentication method. The client proves possession of the client certificate during the TLS handshake by sending the CertificateVerify message, which contains a signature computed with the private key associated with the certificate on the handshake messages that have preceded the CertificateVerify message. Verification of the signature therefore requires access to the handshake messages, so it has to be performed by the transport layer code of the network stack of the relying party, and success or failure has to be reported by the network layer code to the application code. By contrast, as we saw in Section 4.2, a username and a password are sent over an application layer channel, HTTP over TLS with unilateral server authentication, that is easy to use for the relying party.

The same application layer channel can be used for cryptographic authentication. For example, an X.509 certificate, of the kind that could be presented at the transport layer as a TLS client certificate, can instead be presented at the application layer as follows:

1. The relying party sends an HTTP Post request to a credential wallet asking for presentation of a credential. The request is sent over TLS with server authentication, i.e. with authentication of the wallet, but no client authentication. The request specifies the credential that the RP wants and includes a challenge and a callback URL.
2. The wallet asks the subject of the certificate for consent to present the certificate.
3. The wallet signs the challenge with the private key associated with the public key in the certificate.
4. The wallet sends an HTTP Post request to the callback URL containing the certificate and the signature.
5. The RP validates the certificate and verifies the signature.

Section 12.4.3.2 of Chapter 12 has an example of deployment of cryptographic credentials where credentials including X.509 certificates, selective disclosure certificates and anonymous credentials that are stored in a browser but are presented at the application layer through the HTTP-over-TLS channel.

4.5 Protection against man-in-the-middle phishing attacks

To be fair, two-factor authentication does provide protection against simple phishing attacks. It only fails to provide protection against a man-in-the-middle phishing attacks. And if no precautions are used, cryptographic authentication is vulnerable to man-in-the-middle phishing attacks as well. Any challenge-response protocol is vulnerable to an attack were a man-in-the-middle relays to the prover the challenge sent by the verifier, then relays to the verifier the signature on the challenge sent by the prover.

However, a countermeasure is readily available. The prover can defeat the attack by including in the signed data the destination where the message is being sent along with the challenge. In a man-in-the-middle attack, the prover will send the message to the attacker, therefore the destination included in the signed data will be the callback URL of the attacker. The verifier will detect the attack because it will expect its own callback URL to be included in the signed data instead. The destination URL is called "the audience" in the technology literature, and the "verifier designation" in the cryptography literature.

This countermeasure is included in the deployment example of Section 12.4.3.2.

References

[1] Gretzky, K. (2017, April 17). Evilginx - Advanced Phishing with Two-factor Authentication Bypass. Retrieved from https://breakdev.org/evilginx-advanced-phishing-with-two-factor-authentication-bypass/

[2] Gretzky, K. (n.d.). Evilginx v.1.1.0. Retrieved from https://github.com/kgretzky/evilginx

[3] Mozilla Developer Network. (n.d.). Set-Cookie. Retrieved from https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie.

<<< the end