Kai Engert https://kuix.de kaie@kuix.de
kaie@redhat.com

Web security, and how to prevent the next DigiNotar.
(v2, added link to MECAI proposal)
---
Problem of web security: We want to transfer data from A to B, but nobody else should be able to read it or modify it.

This should be true at least for passwords, but remember, if you protect just the password, but not the session cookie that will be used for the remainder of a session, users are still at risk, because an attacker could take over an active session (see Firesheep).

http: no protection

The data that flows from browser to server must travel from your own computer to servers. Anyone along the route (e.g. wifi hotspot operators, people sharing your network, Internet service providers, government agencies) can easily look at the data passing, or even manipulate it.

Worse, an attacker can manipulate the DNS, and redirect users to any page the attacker choses. As we've seen 2 days ago, even large DNS providers such as Verisign are not totally immune against such attacks, although they claim that the attacker was not successul.

The common web approach for web security is to use https: http + encryption, for encryption we use the SSL/TLS protocol

SSL/TLS: data transfer wrapper, adds encryption + key ownership verification

Encryption works by using secret information. You take the input data, you use the secret information to transform the data, then transfer it. Only if the recipient knows how to revert the transformation the original data can be read.

We use "Public Key" encryption, which is based on math, and it uses functions that aren't easily reversible.

Behind the scenes this means, if you want to receive encrypted data, you create a pair of very big numbers. You tell your communiation partners: Use this function and this very big number (the public key) to transform the data. Only I will be able to undo the transformation and read the data (using my private key).

This means, encryption is about keys. The trouble is, you are at home, you connect to your bank's site, anyone on the route between you and your bank can send you false information. An attacker could send you a false key, and pretend it's the bank's key.

How to do you know you are really talking and encrypting

with your bank, not with an attacker (a Man-In-The-Middle) that controls any of the routers?

To solve this problem we must verify key ownership. At the very least we must verify that a key belongs to the domain owner (classic padlock, blue identity in Firefox). Or we can go one step further and verify ownership by a real world identity, like a registered corporation. (This is called Extended Validation (EV) and in Firefox it will be shown with green identity information.)

So, web security is about encryption and key ownership.

How can we verify key ownership?

Technically, a key and an owner name can be combined with a digital signature. This means, the person or organization who created the signature confirms that the key belongs to that owner. This combination is called Certificate.

But how do we trust the ones who create the signatures?

In the past, and until today, a simple strategy has been used:
- find organizations that we treat as trustworthy third parties (Certificate Authority, CA)
- add a list of CAs to our software
- every time we connect to a site, we want to see a key and a proof of key ownership
- if the proof was made by any of the CAs, we'll trust it

Most operating systems or Internet software today includes a list of CAs that are trusted by default.

This allows us to buy an empty computer, install an operating system, install a browser, open https://free-world.mail, and if the server operator has worked with one of the commonly accepted CAs to get a certificate, and if there is currently no MITM attacking our connection, the browser will display information about the connection's security status.

Each software vendor may have its own rules how a CA can get the status "trustworthy". For example, Mozilla.org defines its rules in the Mozilla CA Policy.

Once a CA has the status "trustworthy" and is pre-loaded into the software, a CA has a lot of power. They key owned by the CA can be used to proof the ownership of other encryption keys, either truthfully or abusive.

Once a CA has the status "trustworthy", they can easily share their power with additional organizations by creating empowerment certificates (power transfer certificates) (using the technical mechanism of so called intermediate certificates).

As of today, there is a large of set empowered CAs, as documented by the EFF's SSL Observatory.

What's the problem? The problem is that each of the CAs has unlimited power. As of today, if you own any CA, you can create a proof of key ownership (certificate) for any domain in the world.

That statement is also true for a hacker, who breaks into any CA and abuses the CA's power.

This happened in 2011, we know about two major events.

The infrastructure of a CA named Comodo got hacked and a couple of false certificates were created. Comodo immediately communicated the issue to Software vendors and software projects, including Mozilla. While it was a disappointment to learn that a CA can be hacked, Comodo acted in an honorable way, and therefore we decided to not revoke their status of trustworthyness.

The trouble was, there was no technical solution that would allow browsers to learn that there are bad certificates that should be ignored. The only choice we had was a software update, were we added the certificates to a blacklist.

A couple of months later, another CA got hacked, DigiNotar. Unfortunately, the people running the DigiNotar CA didn't behave in an honorable way. In my understanding they knew about the abuse of their CA, but they decided to be keep the event secret, probably in the hope to save their business.

It took about one month until we learned that false certificates were used by attackers to spy. The only reason why this was discovered was that a browser vendor added additional checks on top of the usual SSL/TLS trust checking. This only worked because the browser and the web site being attacked were controlled by the same organization. The browser had a list of allowed CAs for the web site, and the attacker's cert was from a different CA, and the browser reported the mismatch of the otherwise valid looking certificate. This approach is called CA pinning. It can be used only in a limited amount of scenarios, like when Firefox phones home for software updates, but it's difficult to use it in general.

After this event the decision was made to remove the status of trustworthyness from DigiNotar. Again, the only way to help our users was to create a software update.

It's believed this attack was not about money, but it was about politics. A government might have tried to identify citizens who are working against the government. It's absolutely possible that such people have been identified, and either have been tortured or even killed.

This is a good example that there are many reasons why we should work to improve the security of the web, and make it less likely such events will happen again. But we're not there yet, we need your help.

What's next?

Some people claim that the SSL/TLS security is essentially broken, no longer trustworthy at all. But we're still using it, because it's what most of today's software and the web still uses. Even if we already had a working, better replacement, we can't upgrade all software and all devices at once.

Some people have suggested to stop trusting any Certificate Authorities and ship software with *zero* trust.

But I don't see how that helps.
Whenever people connect to a site for the first time, they would get a security warning.
But how can users verify on their own if a site's key is really owned by www.my.bank?

I personally think that's too difficult. You would have to repeatedly check key ownership on your own, on a new computer, when trying a different browser, while using an Internet Cafe.

Some people suggested to replace CAs using a "web of trust" ran by individuals. The idea is, if other people trust the key ownership, you should trust it, too. The idea is to use the reputation of other people to decide whether you should trust their statements or not.

An example of a project that suggests to use a web of trust is the CAcert community. My criticism is, even if they have a

community of people who build trust, it's still just a single root key. If you are interested to hear their proposal, there will be a talk today, at 16:00, 4 pm, in the hardware crypto room. I will be there, too. If you're interested, we can have an open discussion about these topics.

Another approach suggests to replace today's world of CAs, it was announced in 2011, called: Convergence.

It proposes to completely replace all CAs and introduce notaries. Notaries are servers elsewhere on the web. They try to check what certificates are being used on a server from the Notary's position inside the web. The idea is, if a number of other people on the web see the same information that you see, it must probably be right?

Convergence suggests that users chose which Notaries they should trust.
But most users won't be able to make that decision on their own.

How can we come up with a default set of notaries that we trust, that most of the web would use by default?
Why is that better than trusting CAs? Why should we trust those notaries to give us correct information, and how can we know they are not part of a conspiracy network?
We would need a lot of notaries in order to have the bandwidth to satisfy the verification needs of everyone on the web, who is gonna run all those notaries?

It also requires the use of side channels to check trust, and it's difficult to make it work with Intranet sites.

(Note: some arguments stolen from Adam Langley's opinion on Convergence, see his blog post for details.)

At this point I don't think Convergence is a solution for the general public.

(For the record, there has been an older approach with a similar idea, the Perspectives Firefox Add-On, which has similar problems.)

Ok, so the question is, what else can we do to prevent things like DigiNotar to happen again?

In my opinion, *preventing* is very difficult. Whatever system we will decide to trust, it might always be possible to trick some users into accepting false data, at best we can achieve that false information will only be accepted for a short period of time after an abuse.

So, how are we able to improve anything at all?

I see three aspects.
Aspect (a) Prevent false certificates from being accepted at all.

Because (a) is very difficult, the alternative is:
Aspect (b) Make it possible to quickly detect that someone's powers were abused.

But detection is not enough, in addition we must solve:
Aspect (c) Make it possible that we can react very quickly, and inform everyone to not trust the false certificates.

Let's look at aspect (a), how can we prevent false certificates from being accepted all?

This could be done by limiting which certificates will be accepted to identify server.

One suggestion is DANE, which is based on the secure domain name system (DNSSEC), a system where all information in the DNS is digitally signed. The idea is that a domain owner can publish which CAs the domain owner has decided to trust. For example, the domain owner could decide it trusts CA_1 and CA_2, and publish this information in a record in the DNS. If a browser visits the domain, and the site presents a certificate from CA_3, then the browser should reject the connection. By allowing two alternative CAs, CA_1 and CA_2, the domain owner is able to change a CA customer/vendor relationship.

The problems with DANE are:
- while it reduces the *likelyhood* of being affected if some

other CA gets hacked, it doesn't help at all if the hacker is able to hack CA_1 or CA_2. Any CA could be hacked. You shouldn't assume you're safe by buying from the most popular or most expensive CA.
- how can we trust the information in DNSSEC? Because it's signed. But who signed it? The people who are in control of a country's DNS, or of a DNS zone (.com/.org/etc.). This means, DANE is not a solution against government agencies who want to spy on citizens.
- all trust ultimately points to a single root key. What if that key is stolen or abused? The secure web must probably be shut down for days to recreate the hierarchy of trust.

In my opinion DANE is not a complete replacement for what we have today, however it could be used to complement what we have today. If we have multiple trust statements, we can require that they are in synch, and we could reject keys if one of the trust sources allows us detect an inconsistency.

There is an alpha Firefox Add-On, Extended DNSSEC Validator. (I didn't have time to try it yet.)

Let's look at aspect (b), quick detection of false certificates.

If a single webserver is hacked, nobody might even notice. We depend on administrators to keep their systems save and watch for attacks.

If nobody notices that a server's secret key is stolen, an attacker could run MITM attacks for a long time, because everyone sees the expected certificate. I can't offer a solution for a single webserver attack. But in the end, this means, if a single server is hacked, the users of that webserver alone will suffer from the attack.

Let's look at a larger scale.

What happens if a CA's private key is stolen and abused? How can we quickly detect that?

As of today, we can't. It requires that either the CA notices and reports it to the public. Or it requires that someone actually notices that a bad certificate is being used, and reports it together with the false certificate.

We have seen various attempts and suggestions to help us to detect CA compromises more quickly.

There is the Certificate Patrol Firefox Add-On, which has been around for several years. It shows warnings to the user when certificates change. Unfortunately there are lots of valid reasons why certificates might change. Even for a security expert it's often difficult to decide whether a change is OK or not. This isn't a solution for the majority of users.

There are a couple of new proposals that try to help with quick detection.

The problem with all of them, they are completely new protocols, and it might take a long time, maybe years until they can actually be used by most users.

"Sovereign Keys", proposed by EFF

It proposes that we continue to use certificates from CAs, and verify them as usual. In addition each site or domain owner should create an additional key, the soverein key, and to publish it.
Public timeline servers host an append-only data structure (verifyable by everyone). The browser should query the timeline server, and the SSL/TLS handshake requires a signature made by the soverein key, in addition to the usual checks. The timeline of sovereign keys can grow to several hundreds of gigabytes, and there should be mirror servers that give out information about the timeline.

The idea was presented during a talk at the 28c3 conference in Berlin. People raised a lot of edge cases that would need to be addressed to make it work reliably. Peter from EFF had suggestions for the concerns, but I conclude this protocol still needs to be defined in much more detail.

The full protocol is complex. The EFF says it will probably take years until this can be deployed. But depending on its development, this might be an open, independent option for the long term future.

A proposal that has some similarities is "Certificate Transparency", proposed by Google.

I'm stealing a 10 second summary posted by Ben Laurie and Ryan Dahl:
"Certificates are registered in a public audit log. Servers present proofs that their certificate is registered, along with the certificate itself. Clients check these proofs and domain owners monitor the logs.  Anyone can check the logs: the certificate infrastructure would be fully transparent."

One criticism is, the certificates of Intranet sites would probably have to be added to that public log, too, which might not be wanted by some organizatons.

I don't know if we can expect all domain owners to automatically check the public log for bad changes.

This approach might work, but what I personally don't like is the following quote from the announcement:

"We are sacrificing decentralisation to make things easy on the server. As I've previously argued, decentralisation isn't all it's cracked up to be in most cases because 99.99% of people will never change any default settings, so we haven't given up much. Our design does imply a central set of trusted logs which is universally agreed."

I don't like the centralized approach of this proposal, especially not if it's supposed to be run by a very large company.

The proposal claims that no side channels are required, that a browser is not required to contact anyone else in order to learn about the correctness of a cert, but in my understanding the browser still must "phone home" (or phone Google) to learn about the latest trust. It must probably frequently download lots of new trust information. I'm not sure how that is supposed to work on mobile devices or for people who must work with little bandwidth. Also there will be a delay until everyone learns about the latest trust.

Finally, I have my own proposal, which I call Mutually Endorsing CA Infrastructure - https://kuix.de/mecai
I try to make small steps on top of what we have today.

My proposal is to combine the existing PKI with the Web-Of-Trust / Notary ideas.

Let's not build a new trust network from scratch. Instead of replacing the CAs, I ask the CAs start to run notary services.

The CAs are the ones who earn money by running a trust business. If today's infrastructure run by CAs isn't sufficient, then require *them* to contribute to improve web security.

Instead of introducing a new set of authorities to trust, let's

use the trust information that we already have in the browser. CAs should create digitally signed vouchers of what they see being used by servers.

CAs would create vouchers that contain DNS information (domain name to IP), what certificate is being used on a particular server, and also the most recent revocation information that can be found for the certificate being used.

Essentially the vouching service combines network information, current timestamps and adds a digital signature.

Daily, each server could obtain current vouchers, from each of the CAs that are known on the public web.

Whenever a browser connects to a server, it will look at its embedded list of trusted CAs, and will *randomly* select a list of 3 candidate CAs that will be accepted to vouch for this connection. Inside the SSL/TLS handshake, the server is expected to include a recent voucher that was issued by a CA that is *different* than the one that issued the server's certificate.

This allows the browser to compare its own view of the network with the notaries view.

With this approach, hacking any single CA and being a MITM to a victim will no longer be sufficient. The attacker would have to be able to control network routes between the

server and the vouching servers. The attacker probably can control only a small number of routes, not all of them. This means, because of the random selection made by the browser, a large percentage of users will get security warnings. There is probably a security expert in this large group of users who can report the issue.

If the vouchers are usually valid for 24 hours, this also allows us to globally revoke trust from CAs within one day. We inform the Vouching Authorities about the compromise, and they will immediately stop issueing vouchers for the related certificates. As soon as a browser no longer gets a valid voucher from a server, it can stop trusting the server's certificate.

Because no side channel is required, this approach would work with captive portals, such as pay-for wifi hotspots.

But still, the problem with all these approaches, they are ideas for the future. Someone must specify them in all the low level details, experiment with them, and we must wait for them to become standards.

This means, it doesn't help us in any way for the short term. But we really should prepare for the next attacks. So what else can we do?

This is aspect (c). Let's improve how quickly and easily we can react as soon as we learn about false certificates. This is about revocation.

Several kinds of attacks have actually been predicted by the designers of today's PKI, our security infrastructure. Unfortunately, the software we have today doesn't implement all the solutions that can help to protect us.

**Our immediate homework is to enhance our incomplete security software.**

If you are a C programmer and want to help, please get in touch with us.

Once a server key compromise is detected and reported to a CA, the design of PKI allows a CA to revoke the key/certificate and both parties can work to setup a new key pair and certificates. CAs are expected to publish information about revoked certificates. Clients who learn about the revocation can reject connections that use the old, revoked key, which is likely being used by a MITM.

(This is also the reason why certificates expire, to reduce the burden on CAs, so that their list of revoked certificates will not grow endlessly, but can be cleaned up from time to time.)

The revocation approach can also help with a lot of hacks of delegated empowerment certificates, the certificates that

transfer CA power to other organizations.

While the designers of PKI have invented the revocation mechanism, and while several large software applications implement parts of the mechanisms, many implementations are insufficient, including Mozilla's implementation. We simply don't have enough people to work on it and get everything done that's important.

Even though Mozilla software is able to retrieve revocation information, even though Mozilla will try by default to obtain current revocation information, that's not sufficient, for several reasons.

(1) We only support realtime mechanism called OCSP. But as of today, CAs are not required to make data available using this protocol. Many CAs do not.

(2) For technical reasons, we don't retrieve Certificate Revocation Lists (CRL) by default. If you are a C programmer and want to contribute, please get in touch, or search for the bug with alias "pkix-default". We are close to have this code ready, but we need help with fixing bugs.

(3) As of today, if a Mozilla client is unable to retrieve current revocation information (be it unavailibity or technical constraints), it will ignore this fact.

This means, a MITM attacker who wants to make use of a

stolen server certificate can simply block our attempt to obtain current information, and we will continue to accept it without warnings.

We should work on a persistent OCSP cache on disk. Some MITM attacks are only temporary, and if we cached revocation information on disk, the browser would be prepared for a later MITM attack involving the known revoked certificate.

Technically, we have the ability to require the availability of recent certificate status or revocation information, at least for those CAs that make such information available. You can already enable that strictness in Firefox/Thunderbird preferences. Unfortunately, there are usability issues that have so far stopped us from making it a default.

One problem is, the web is huge, and there are only a few CAs. This means the OCSP servers operated by CAs have to handle a lot of connections from all over the web. This means a lot of traffic for CAs. Sometimes a server might be unresponsive due to load. Or a CA might have a server outage. In such scenarios, if we decided to require OCSP by default, then users would get security warnings and be stuck more often.

We should work to get it done anyway, and we should work to make this bearable.

A partial remedy is OCSP stapling, a protocol that makes it unnecessary to open a separate connection to a server run by the CA. The protocol allows a server to obtain current information on its own, cache it, and include it in the SSL/TLS handshake between client and server. This reduces the load on uptime requirement's for OCSP servers.

We are working on OCSP stapling on the client side. We need help to implement better testing for OCSP (360420).

Because Mozilla has contributed money in the past to the openssl project, this feature may soon became default on Apache Servers (Apache bug 50740). If you can, please help to test this feature and lobby for getting it enabled by default.

For servers that don't support OCSP stapling yet, what happens if certificate status servers are down or cannot be reached? Until we can expect that OCSP information is more reliably available, we should probably implement browser user interface that allows users a temporary, short-lived override (proceed anyway).

Both features might be necessary before we can require OCSP by default, because a major scenario where we have problems with OCSP are paid-for Internet connections, such as wifi hotspots or hotel room connections. Before you can pay, you often cannot contact an OCSP server yet (340548), which results in a security error on the payment page. OCSP stapling and overrides could help.

You could also help by contributing to Mozilla's captive portal detection (bug 562917).

Unfortunately, there are still many large scale scenarios where the revocation mechanisms based on CRLs and OCSP don't help at all.

=== Revocation doesn't work if a CA's powers get abused. ====

Scenario (a) A government agency compels a CA to issue server certificates (or even empowerment certificates) that can be used for MITM purposes. This may be difficult to prove or detect, but in case it can be proven, the CA must be revoked. (Related: I created the Conspiracy Add-On, which could be used by experts to learn if an unexpected CA from country 1 was involved in issueing a certificate for an organization from country 2.)

Scenario (b) A hacker is able to break into a CA's infrastructure and is able to steal their private key. With this key, they can create any kind of certificate, even ones that don't contain information for revocation checking and certificates will be accepted without further checks.

At least in these scenarios, we must revoke all trust from a CA. At the very least, if the abuse is very well understood, it

might be decided that blacklisting the set of bad certificates is necessary. How can we do any of that?

As of today, our only possible reaction is to produce a software update that removes or blacklists those certificates. This has disadvantages:

First, a software update needs software engineering work from the security guys, then quality assurance work and then release engineering work. This makes it slow it deliver. We lose time while people are vulnerable.

Second, a software update cannot be quickly delivered to everyone. Some organizations roll out updates only after a delay and testing. Or a MITM attacker might block victims from connecting to Mozilla's update servers.

A better approach would be a mechanism, that simply delivers a list of the revoked CAs, overriding the information that is included in the software. That data could be small, it could be just some signed ASCII text with an ASCII signature, allowing people behind a governmental firewall, or shielded by an attacker, to distribute such critical revocation information using a peer-to-peer approach, such as social networking announcements or e-mail.

As a proof-of-concept, I created an experimental Mozilla Add-On named CA-Knockout, which parses a small text file and removes trust from affected CAs. But it would be better if

such a mechanism were built-in to Mozilla software directly. If you want to contribute, bug 647868 has suggestions. In my opinion this work should have a very high priority.

Another theoretical improvement is to limit the powers of CAs. We could restrict some CAs to issue certificates only for a single country's TLD. That might help, but how do we solve the problem that most CAs want to issue certificates for dot com, in order to make money with businesses? So such constraints are only of limited help.

Finally, there is one more issue that needs to be addressed, Too-Big-To-Fail.

If a major percentage uses certificates from a hacked CA, what do you do? If you revoke the CA, a large percentage of the web will immediately be treated as unidentified with security warnings, which users will have no choice as to accept and override, which makes it much easier for MITMs to trick users into accepting any MITM certificate for the affected sites.

This is why I believe it is very important to introduce redundancy into PKI. It should be possible to prepare for CA revocation, by obtaining two independent certificates, and configuring servers to have both alternatives available. That will require incremental work on SSL/TLS standards and

protocol enhancements, in order to allow such redundancy.

As you can see, there are a lot of issues and a lot of work needs to be done. Please help and please make it a very high priority to improve our security code.

If you would like to continue this discussion, have questions, or if you want to help but don't know how to start, please find me. My nickname on IRC is kaie. Also, you could go to the hardware crypto room for a talk at 4 pm, which will likely be followed by an open discussion about these issues.

Thank you

https://kuix.de/fosdem2012/