No more revocation - an applicable Short-lived certificates framework

This post was co-authored with Marios Galanis, Nobelle Tay, Dan Boneh.

The modern Web Public Key Infrastructure (PKI) facilitates the secure electronic transfer of information for network activities. Unfortunately, some components of the PKI ecosystem are still insufficient. How to efficiently revoke misissued or compromised certificates poses a problem that has been discussed actively. In this article, we will discuss the problem and propose our solution.

The problem

Web servers use Public Key certificates to prove their identity and establish encrypted and authenticated communication with clients (e.g., browsers) via Transport Layer Security (TLS). Within TLS, the public key on the issued certificate is used to exchange a secret encryption key. Therefore, it is important to ensure the certificate is legitimate and the information provided is authenticated. However, in the past years, we have seen several cases where even correctly issued certificates have turned out problematic and required to be revoked. In 2011, Diginotar, one of the root Certificate Authorities (CA) trusted by the major browsers, got compromised and issued many rogue certificates for high-value domains, including Google, Yahoo, Mozilla and others [1]. After the attack is discovered, all certificates issued by Diginotar are not trustworthy anymore and need to be revoked. In another case, when the private key of a web domain's certificate is leaked to an attacker, the attacker can impersonate the certificate owner and decrypt all the transferred information. Thus an update of the public/private key pair and revocation of the old certificate are necessary. In these cases, stopping browsers from trusting the out-of-date certificates is crucial to protect users from visiting malicious websites. Nevertheless, all solutions of revoking certificates currently face some problems.

Certificate revocation list (CRL) is a list of PKI certificates that are issued by CAs but have been revoked. A certificate must be checked against the CRL before being trusted. The effectiveness of this mechanism requires clients to obtain the complete and latest list of revoked serial numbers. In the case of Diginotar, the list contains every single certificate ever issued by the CA. The amount of data puts the burden on network and client resources. And if clients fail to obtain an updated CRL, the attacker can abuse the rogue certificate and impersonate the server.

Online Certificate Status Protocol (OCSP) is an Internet protocol used to report the revocation status of a certificate. Instead of letting each client maintain a list of revoked certificates, an OCSP responder maintains the latest one and returns a signed response about the status of the

queried certificate. The messages are communicated through HTTP. Although OCSP requires less network bandwidth because of less data transmitted, it poses other problems. When an attacker is able to interfere with the OCSP queries, clients fail to receive certificate status and load the web page. Worsely, most clients would silently ignore OCSP if the query timeouts, which leads to no protection from using revoked certificates. Moreover, OCSP discloses to the responder the web domain the client visits upon receiving the query, posing a privacy issue potentially could be used to track clients [2].

Solution

The online revocation checks, as discussed above, are vulnerable to attacker's interference and failures in getting the revocation status of a certificate. Therefore, we aim to prevent online checks and construct the certificate as a caching mechanism.

We propose to reduce the validation period of the certificate to a few days and simply stop issuing new certificates with the old public key to achieve revocation. Specifically, each issued certificate is only valid for 4 days, which we call a short-lived certificate. For a web domain on a given day, there are 4 short-lived certificates available to use. The first certificate is valid from 4 days before and expires on this day. The second certificate is valid from 3 days before and will expire the day after this day, and so on. Each day, the web domain uses a new certificate that expires at the end of the same day. This design has three benefits. First, it prevents single-point failure of using certificates. If CA or other parties are not able to issue new certificates as expected, the cache of 4 certificates still enables the web domain to operate for another four days while issuers could fix the problem. Second, when the certificate is compromised, the attacker can only exploit it for a maximum of 4 days. To revoke certificates for a web domain, the CA stops issuing any new certificates with the old public key. Third, this cache solves the problem of unsynchronized clock time and ensures available certificates for the next 4 days.

In this solution, the damage caused by compromised certificates is diminished because of the validity period of 4 days. Clients only need to verify the legitimacy and validity of the certificate itself without worrying about whether the certificate is revoked or not. This greatly reduces the communication latency and network bandwidth being used. No third-party is introduced meaning we do not transfer the trust to elsewhere and not introduce any privacy concerns. Large amount of certificate revocations at a time can be easily handled without extra cost in memory or computation,

Implementation

We designed the framework based on modern PKI so that our proposal is easy to incorporate with today's work, including the operation of CA, Certificate Transparency (CT) and clients. We

also aim to not create much computation and memory costs for each party so the design is practical and scalable.

Our project involves three parts: a CA responsible for issuing certificates, a middle daemon responsible for storing certificates and generating verification proof, and a website daemon responsible for obtaining and serving daily certificates for web domains. Here, we set the validity period of each short-lived-certificates to be 4 days and the issuance of multiple of them accounts for a total validity period of 365 days. This period can be changed based on the web domain's choices. Next, we explain the operation details for each component.

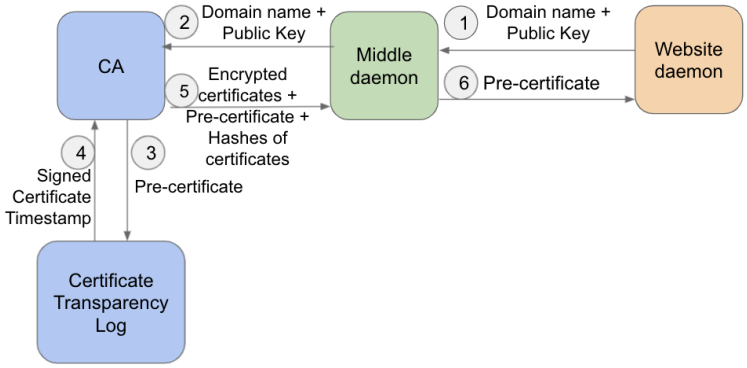

Figure 1. Certificate generation during initialization phase

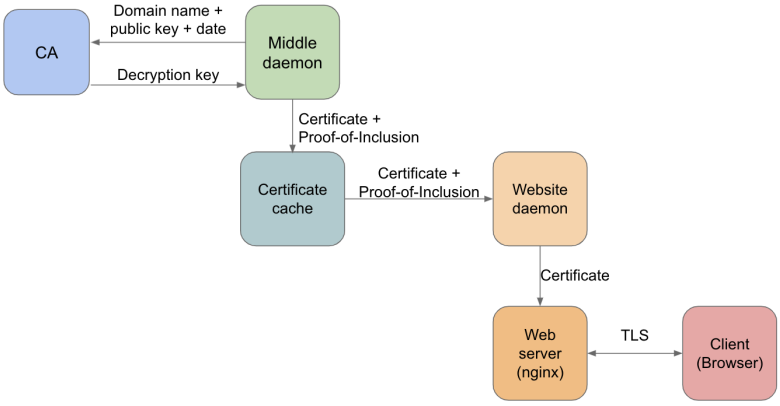

Figure 2. Certificate issuance on daily basis

1. Certificate Authority (CA)

For a given web domain, CA still generates and signs certificates. Instead of a one-year certificate, CA generates 365 short-lived certificates. Each certificate has a different starting date and lasts for 4 days. CA generates a cryptographically secure key for each certificate from a pseudorandom generator using its secret key, the domain name, domain public key and certificate starting date, and encrypts the certificate correspondingly. This is to ensure future

certificates cannot be used until the time when CA provides the decryption key. If the certificate is compromised, attackers can only obtain future certificates in encrypted form. The decryption key does not need to be stored since it can be regenerated, which saves memory cost.

To help the middle daemon generate proof-of-inclusion, CA builds a Merkle tree from the 365 certificates and sends the leaves, which are the hash values of the 365 certificates, to the middle daemon. To record the issuance of the certificates to Certificate Transparency (CT), CA generates a pre-certificate and attaches the Merkle root as an extension. This pre-certificate is sent to CT to obtain a Signed Certificate Timestamp (SCT) which is regarded as a promise of logging the pre-certificate along with the Merkle root. After receiving the SCT, the CA attaches it to the pre-certificate, signs on it and forwards to the middle daemon. The pre-certificate has a validity period of the total short-lived certificates (1 year) and the poison extension is not removed so it cannot be used as a legitimate certificate. As the last step in the setup phase, the CA sends the encrypted certificates to the middle daemon. And it discards all the generated values but its own secret key. The certificate issuance process is shown in Figure 1.

Throughout the year, every time CA receives a request from the middle daemon for the decryption key of a domain's certificate, the CA regenerates the key from the PRF, and forwards to the middle daemon. This is shown in Figure 2. Details on operations of Middle Daemon are discussed in the next section.

2. Middle Daemon

Middle Daemon starts by storing 365 ordered and encrypted certificates (when a certificate is decrypted and delivered it can be deleted), one pre-certificate with an attached SCT, and the leaves (hash values of the 365 certificates) of the Merkle tree. On a daily basis Middle Daemon requests and receives the decryption key for the current day's encrypted certificate of each domain, decrypts the certificate and creates its Proof-of-Inclusion using the Merkle tree. Finally, the decrypted certificate along with its Proof-of-Inclusion are sent to the certificate cache available for web domains.

Middle Daemon is architected to run in containers created from an image, and can be deployed to Amazon ECS on Fargate. The certificate cache is hosted on Amazon Simple Storage Service (Amazon S3) by Middle Daemon and accessed by the corresponding Website Daemon. Middle Daemon refills the cache each day with one certificate and Proof-of-inclusion and ensures there are four available ones. Website Daemon retrieves one certificate with proof each day which expires at the end of the day from S3 bucket.

3. Website Daemon

Every day, Website Daemon downloads a new short-lived certificate from the cache, along with a Proof-of-Inclusion for the Merkle tree. Then, Website Daemon updates the path to the new short-lived certificate in the web server configuration. Finally, the Website Daemon reloads the web server to pick up the new configuration.

This logic is similar to the certificate renewal process today on web servers, except that the Proof-Of-Inclusion for the Merkle tree is downloaded alongside the new short-lived certificate. The Website Daemon merely automates the certificate renewal process to ease the transition to the short-lived certificate program for the domain. The design also allows the domain to continue serving HTTPS requests without taking the risk of putting the web servers offline for a significant amount of time.

4. Client Verification Process

- **TLS Handshake**

During the TLS handshake process between the website server and the client, the client verifies the signed short-lived certificate and the SCT in pre-certificate as it does today. To mitigate the overhead, the client can cache the pre-certificate with a reasonable time-to-live because it is the same for the 365 certificates. If the client would like to validate the short-lived certificate is logged to CTLog, the client can send an additional request to the server to get the Proof-of-Inclusion for the Merkle tree. The validation process is the same as the auditor's process discussed next.

- **Certificate Transparency**

To audit the short-lived certificate as part of the Certificate Transparency framework, the auditor needs to verify two conditions: the pre-certificate with the Merkle root has indeed been logged, and the short-lived certificate that the server is using has indeed been included in the Merkle tree. To verify the first condition, the auditor needs to get the logged pre-certificate from the server so that it can asynchronously contact the CT to verify the pre-certificate has been logged, which is not any different from the auditor we have today. To verify the second condition, the auditor needs to get the Proof-of-Inclusion for the Merkle tree from the server and perform the verification. In the client case of verification, it can batch the requests to the server and store the logged pre-certificate for each domain in a cache with a reasonable time-to-live.

- **Short-lived Certificate Enforcement**

Once a domain is enrolled in the short-lived certificate program, the client should not accept long-lived certificates from the server. This can be done by adding a new HTTP response header

type, similar to Expect-CT/HSTS response header that allows the client to enforce certificate transparency requirements. However, this enhancement requires an update to the HTTP protocol, and thus we decided this is beyond the scope of this project.

Source Code

https://github.com/gongfchen/short-lived-certificate

Reference

[1] Josephine Wolff, How a 2011 Hack You've Never Heard of Changed the Internet's Infrastructure (2016) Future Tense.
[2] E. Topalovic, B. Saeta, L.S. Huang, C. Jackson, and D. Boneh, Towards Short-Lived Certificates (2012) In proceedings of IEEE Oakland Web 2.0 Security and Privacy.