Global Authentication in an Untrustworthy World

Martín Abadi, Andrew Birrell, Ilya Mironov, Ted Wobber, Yinglian Xie
Microsoft Research, Silicon Valley

Abstract

With the advent in the 1980's of truly global hierarchical naming (via the Domain Name Service), security researchers realized that the trust relationships needed to authenticate principals would often not follow the naming hierarchy [1,13]. The most successful non-hierarchical authentication schemes are based on X.509 and RFC 5280, as used for example in TLS and Authenticode. These are extremely widely deployed, and are trusted for most people's everyday use of the Internet. Unfortunately several incidents in the last few years have proved that this trust is misplaced [9,14]. We explore the weaknesses of this machinery, helped by a large database of X.509 certificates, and we offer an analysis technique and a suggestion for how the trust could be enhanced.

1. Introduction

The most widely used scheme for authenticating unrelated third parties on the Internet is X.509, using the X.509 "profile" defined by RFC 5280 [5]. This is the machinery underlying web requests via the HTTPS method, or secure email access (POP or IMAP over TLS, or SMTP with TLS), or secure installation of code ("Authenticode"). The RFC 5280 authentication decision is made by the "relying party" after it obtains a sequence of certificates. For example, when the relying party is an HTTPS client in a web browser, it receives the certificates during the TLS handshake, and appends certificates from its store of trusted root certificates. The intention is that each certificate in the sequence is signed by the public key of the following certificate in the sequence, up to a top-level "trust anchor". If the relying party decides to trust this sequence of certificates, then the authentication succeeds. The relying party then believes that the public key in the initial certificate "speaks for" the server [12].

It is important to realize that the relying party's authentication decision implies that it trusts all of the parties that wrote the certificates involved. Any one of them, if misbehaving, could cause the decision to be incorrect.

Our goal in this research is to allow relying parties to make better-informed judgments in this authentication decision. First, however, we will explain quite how badly judged the current decisions can be.

We will concentrate on the example of a client using a web browser to make an HTTPS request. The *de facto* state of the art is that a browser arrives with an initial set of trusted certificates chosen by the browser vendor, and the browser uses these as the trust anchors in the above authentication protocol. On success it displays a padlock or some such graphic in its window.

As delivered by the vendor, Microsoft's web browsers trust roughly 337 root certificates; Firefox trusts about 180. In both cases these certificates are trusted because the top-level CAs that are the subjects of the certificates have met some criteria specified by the browser vendor. These include security guarantees and third-party auditing requirements. The criteria read nicely, but the resulting security is analogous to the oxymoron of a secret shared by 337 people.

But it's worse than that. Each of the top-level CAs can freely certify intermediate CAs, to which they delegate their full power. For example, the top-level trusted CA of the Government of Korea has signed at least 194 intermediate CA certificates that are valid at the date of writing. Each of the 194 intermediate CAs could write a certificate for an arbitrary domain name and it would be trusted by your copy of Internet Explorer. Similarly, DFN, the German research and education network, has signed at least 290 valid intermediate CA certificates that Internet Explorer would trust for arbitrary uses. In total we have observed 1,510 intermediate CA certificates, still valid at the date of writing, and trusted by the web browsers for certifying across the entire name space.

This design, even if correctly implemented, leads to many opportunities for incorrect trust decisions, which can be encouraged by various economic or political incentives. Perhaps the most famous failure was the incident in which the top-level CA DigiNotar was compromised and was used to write trusted certificates for google.com (among others), which were deployed and allowed man-in-the-middle attacks on

Our work in this area has benefitted from discussions with our colleagues, particularly Moisés Goldszmidt, Muhammad Umar Janjua, Anoosh Saboori, and Kelvin Yiu. Martín Abadi is also affiliated with UCSC.

users in Iran [9]. There have been several other examples: the Flame malware relied on a compromised certificate; the compromise was possible because an intermediate CA was using inadequate cryptography (an MD5-based signature [10]) and was effective because of inappropriate use of the code-signing capability [14].

The bottom line is that a global system where trust is based on uniform acceptance of any of 1,510 CA certificates will inevitably produce incorrect decisions: incorrect globally, or incorrect for a particular user, or a particular enterprise, or a particular location.

In the remainder of this paper, we will describe our progress so far in allowing clients to make better authentication decisions. In particular, we advocate:

- Acquiring a thorough understanding of the reality and failings of the current public PKI (section 3).
- Enabling local policies for trust decisions, in place of the current purely global ones (section 4).

These approaches are synergistic, as we discuss below.

2. Related Work

There are several existing proposals to improve the security of X.509-based authentication, some still being designed and some that have been deployed at prototype or even beta-test levels. We mention here only a selected few of them. Throughout, we focus on approaches that remain close to X.509 and the existing infrastructure, as these seem most likely to have immediate impact. The literature contains several other interesting but largely untested suggestions with different starting points, such as SDSI and SPKI [7].

RFC 5280 itself describes the "name constraints extension", which limits what names a CA certificate may sign. This is powerful, but it also conflicts with the desires of the organizations that run the CAs: they want the freedom to certify names over a wide range, especially the retail CAs. We have seen only 36 certificates containing name constraints

"Extended Validation" (EV) certificates provide more reliable information about the principal behind the certificate, displayed in the user interface [2]. Like most security features that rely on user behavior, they tend to be ignored by most users, who are neither willing nor qualified to make the requisite judgment calls.

"Public Key Pinning" is a mechanism used in Google's Chrome browser, comparing newly received certificates with known trusted public keys [8]. It was notable in detecting the DigiNotar compromise.

"DANE" is an IETF work in progress [11]. DANE restricts the certificates that should be trusted for a host with a particular domain name by adding records in the corresponding DNS entry. It is a powerful mechanism, and could prevent many security problems. Unfortunately, the DANE design, by adhering to the DNS hierarchy, prevents the non-hierarchical trust that many users and administrators demand. DANE's security also relies on the security of DNS, and therefore on DNSSEC (which is by no means globally accepted). More fundamentally, DANE continues to leave the relying party out of the trust decision.

The "Perspectives" system [15] allows a client to compare the certificates that it has received from a server with those seen from the perspective of an independent notary. This is aimed at detecting targeted attacks. For example, if your web browser is being subject to a man-in-the-middle attack, such as was used in the DigiNotar compromise, then your client could detect this if the Perspectives server is not subject to the same attack. This style of service provides valuable, though fallible, input to a rational trust decision.

Several proposals take the form of comparing the received certificates with a repository of previously received certificates. These are typically implemented by DNS queries matching received certificates' fingerprints with the database. Again, this is valuable information, but fallible. Unlike Perspectives, these plans rely on history. They are limited in accuracy (because things change) and recall (because the underlying crawl of the Internet is incomplete).

3. Understanding the World

One cannot improve the security of a large-scale authentication system without first understanding it.

Understanding protocols and mechanisms is not especially difficult, but understanding how they are deployed in the Internet is challenging. Fortunately, in 2010 EFF published recordings of TLS handshakes gleaned from an enumeration of the public IPv4 address space [6]. We used this data as our starting point. We extracted the certificates, parsed them, and constructed a SQL database.

The original EFF data from 2010 provided us with 5,606,345 distinct certificates, from 11,349,715 IP addresses. During 2012 we enhanced this by fetching current certificates from those addresses and from other sources, so that we now have 7,760,225 distinct certificates from 11,493,324 distinct IP addresses. 3,282,181 of the certificates are self-signed. Of the remainder, 3,412,615 are server certificates that would be (or would have been at some previous date) trusted by a browser. We have retained all the historical data, including the origins and sequencing of the certificates.

This origin data allows us to reconstruct or analyze the sequences of certificates that were returned during 22,654,969 TLS handshakes.

While not "very large" in the sense of VLDB, the database is quite complex, representing the certificate structures and signing relationships in a lot of detail. The database has about 13 GBytes of table data, and the on-disk size including indexes is about 39 GBytes.

Our database is rather more elaborate than the one used by EFF, designed for more detailed search, browsing and analysis. For example, we include a table that enumerates signing relationships among the certificates' public keys, and we can enumerate the exact signing chains that browsers will trust.

The most obvious use of this database is to perform *ad hoc* queries, for education or in support of decision-making. For example, it's easy to check for certificates with known deficient cryptography (such as MD5 signatures or 512-bit RSA moduli) – this was used recently to consider the impact of rejecting these certificates in the web browser. More elaborately, we can check for signing chains where a CA is using a weaker key than the key in the certificates that it signs. We can evaluate the quality of a CAs behavior in actual practice, enabling for example a comparison of that with the CABForum's "Baseline Requirements" [3]. Or we can find signing chains where the certificates involved have subject names from more than two countries. Recently, we determined the popularity of signing CAs for the server certificates used by the HTTPS servers from the Alexa "top million" sites.

While such *ad hoc* queries are educational and sometimes important, the total amount of data involved can be overwhelming, and cries out for more systematic analysis. We have performed one such analysis by clustering the certificate chains from the database.

There are endless possible criteria for clustering the chains; we provide just one example here, and recognize that it is a preliminary attempt. The metrics used should have the effect that the clusters indicate similarly trustworthy (or untrustworthy) certificate chains, and the outliers should represent surprising and probably suspicious certificate chains.

The input entities for the clustering analysis are the certificate chains from the database: each entity represents one TLS handshake. We consider only the set of chains that would lead to a trusted root in either IE or Firefox. Each chain is a sequence of certificates $\{C_0, C_1, C_2 \ldots C_{n-1}, C_n\}$, where C_0 is the server's certificate and C_n is the certificate of a trusted root. (The chain we analyze has the certificates up to and including the trusted root, regardless of whether they were all included in the actual TLS handshake). We use the following five features from each of C_0, C_1, and C_{n-1} (substituting C_n for C_{n-1} if "n" is 1):

- Key-Length: length of the public key;
- Year: year number of issue date ("notBefore");
- Is-CA: the certificate is flagged as a CA;
- Country: subject's country code ("C=" field);
- Host: certificate's domain name ("CN=" field).

Three features represent the overall certificate chain:

- Root-Key: the public key of C_n;
- Chain-Length: n+1;
- C-Match: the subjects have the same country.

Thus we have a total of 18 features for each certificate chain. Among these features, Is-CA and C-Match are boolean features; Key-Length, Country, Host, Root-Key, and Chain-Length are categorical features; Year is a numerical feature. For Key-Length, we quantize the length into several ranges and each range is a category. For Host, we consider three categories: normal, missing (no hostname in the subject field), or irregular (e.g., "plesk" or "localhost").

The distance between two certificate chains is computed as the weighted sum of the distances of all feature dimensions. For boolean and categorical features, if the values of two features are identical, we define their distance as 0; otherwise the distance is 1. The only numerical feature is the certificate issue year, where we normalize the Euclidean distance of two feature values by the maximum range of the issue year across our dataset. The Root-Key dimension is given a weight of 2 to give it more importance empirically. All other dimensions have a weight of 1.

We used the "K-means++" algorithm to cluster the chains (using modes for categorical features like country names). We customized it in our setting so that the clustering runs iteratively. Initially, we select K centroids that are distant from each other. We then apply the K-means algorithm to compute K clusters. If any big cluster (with at least 100 certificate chains) is already tight (i.e., the majority of the data points are close to the centroid), we output this cluster except the set of outliers that are far away. For all outliers and the remaining data points, we select new centroids and iteratively apply the K-means algorithm until no such big clusters exist, or until we reach an arbitrary maximum number of iterations (100). We found that the clustering results are not very sensitive to the particular thresholds we chose (cluster size, tightness), since big, tight clusters come out naturally in most settings. The results are relatively stable.

Specifically, we analyzed 2,762,551 chains, obtaining 261 clusters. Of these, 216 had at least 10

chains. 1,771,105 of the chains ended up in "tight" clusters (ones with mean distance less than 1), suggesting that they have clear group patterns. There were very few outliers: 135 chains further than distance 3 from their cluster's centroid. The running time for this experiment was about one hour on a 2.6 GHz 4-core Xeon CPU using about 10GB of memory.

In one test of the clustering's predictive value, we correlated clustering with certificates' revocation status, which was not an input to the clustering algorithm. We found that they aligned remarkably well. For example, 90% of all revoked certificates in the database were assigned to just 21 clusters. We manually examined a small cluster with an atypically high revocation rate, and discovered that sites in the cluster serving non-revoked certificates were unreachable, or offered expired certificates, or had recently moved to a different CA (thus leaving the cluster).

In sum, although our exploration of clustering remains preliminary and somewhat *ad hoc*, we see indications that this approach has the potential to help us understand the large, varied space of actual certificate chains. This understanding may contribute to thwarting attacks. It can also inform the design of new policies and mechanisms, such as those that we discuss in the next section.

4. The Policy Engine

Given the above, it seems unlikely that simply evolving the existing mechanisms will be satisfactory. So we offer here in outline an alternative: allowing a client to make its own decisions about what certificates to trust. This is a fundamental shift. It enables the client to make client-specific decisions about trust, suitable to this client, rather than just to a global authority. For example, a client (or the enterprise managing a client) might decide not to trust certificates with RSA keys shorter than 2048 bits; or not to allow a foreign CA to certify an enterprise's internal email system.

Both the diversity of certificate chains in our database and the variety of proposals discussed in section 2 suggest that clients may benefit from a fair amount of flexibility in making those decisions. Some clients might have only basic cryptographic requirements, while others might wish to remember known keys or known CAs, or both, and perhaps apply DNS-based checks. Similarly, a single client might reasonably want to apply different criteria in different parts of the domain name space.

We have designed and prototyped a "policy engine" that offers improved ability for a client to make trust decisions based on the certificate sequence received from a server. The design is based on a "policy", which can be adopted by an end-user or by an organization. We do not envisage normal end-users designing policies, and only rarely would they modify them. Rather, an end-user would obtain a baseline policy that suits the user's desires, from an organization trusted by the user: for example, the user's ISP, or IT department, or somewhere independent like Consumer Reports. We would not expect policies to come directly from browser vendors: end users' trust requirements are too diverse for that too work.

A database such as the one described in section 3 is important for such a policy engine: it allows a client to evaluate the impact of a policy decision (e.g., how much of ".com" would be excluded if the client required 2048-bit RSA keys?). The database will also be an underlying mechanism for many of the policies.

The input to the policy engine is a received sequence of certificates (e.g., from a TLS handshake) and a desired target (e.g., the server's domain name in an HTTPS connection).

There are many possible languages and structures that could be employed for a policy engine; here we offer one design.

The most primitive part of this design consists of "judges". A judge is a function taking the input and delivering a boolean result. Each judge has its own criteria by which to judge the input, and its result is "true" iff the input meets this judge's criteria. The criteria might indicate goodness (e.g., this certificate chain is the same as you saw every day for the last month), or badness (e.g., 99% of the certificates issued by this CA were for a different country than the server you're contacting).

Judges are grouped into "panels". A panel is a function taking the input and delivering a set of (name, boolean) tuples. This function attempts to invoke each of the panel's judges. The panel's result is one tuple for each of its judges, each containing the name of the judge and the judge's result; plus one tuple with the name "failed". The "failed" tuple's boolean is "true" iff the panel failed (for example, because of inability to access a remote data source). Panels serve two purposes. First, they allow multiple judges to consult a single data source in a single transaction. Second, they are the unit of failure. If the panel indicates failure, all of its judges' booleans are false.

For example, we would expect one panel to be purely local: its judges look at the input and use criteria such as cryptographic strength, or requiring that an enterprise resource must be authenticated by an enterprise CA, or interpreting certificates' "Policy" extensions [4]. A more elaborate panel might additionally have judges that consult local history: is

this the same certificate as the server proffered last time, or is this certificate signed by the same CA as the server used last time. It would also be possible to have a panel whose criteria are based on the results of our cluster analysis.

Similarly we would expect to have a panel whose judges use criteria based on queries on our database: is the CA the same as the one recorded for this server in the database; or is the CA one that is normally used almost exclusively for servers in some other country?

Finally, we would expect to have panels accessing the other proposed improvements to X.509 security: DANE, Perspectives, and the various certificate repositories. (We could also use our database to implement a panel with Perspectives-like semantics.)

The top level of this design is the policy itself. A policy is a function from the input to a single integer. A policy is defined by a set of tuples. Each tuple is either (panel-name, judge-name, integer) or (panel-name, "failed", integer). The policy's result is the sum of the integers for judges whose boolean was "true" in the result from the corresponding panel, plus the integers for the "failed" tuples for panels that returned a failure indication. The integers here should be viewed as weights: positive, indicating goodness, or negative, indicating badness. A policy might use only a subset of the available judges and panels.

The three levels of this design have quite different characteristics. The judges and panels are highly technical, specialized to their criteria and often making sophisticated use of complex data sources such as our database (in which case the judge is likely to be written in SQL). Panels that access only the input are trivial; but ones that access history or external data sources can be quite complex, dealing with connection and resource management, caching, and failures. We expect judges and panels to be viewed as opaque software distributions, created by experts for use by other people's policies.

The policy itself on the other hand is conceptually very simple, although designing one from scratch is a skilled task (much as in the analogous mechanism of SpamAssassin). The judges and panels offer no opinion about the final trust decision, but the policy embodies the trust strategy of the client. As discussed earlier, policies would be adopted by individual clients or groups to reflect their own needs.

Note that the policy is informed of failures and decides what to do with them by the weight it gives them: zero to ignore them, or a very negative integer to treat them as fatal. Judges and panels have no opinion on the significance of failure, they merely report it.

The final part of the policy engine design is its integration into the client workflow, acquiring the input and acting upon the resulting integer. We see three places in the system where this could be done.

First, it could be a layer on top of the system's cryptographic library (CAPI in Windows, OpenSSL in most other systems). We have not explored this in detail, because of the complexity of these interfaces. Experts advised strongly against it.

Second, it could be inserted into the client's workflow as a web proxy. This is quite easy to do, and allows the policy engine to intercept the TLS Handshake Protocol when used for HTTPS requests. We have prototyped this, but do not favor it. The only way to report a judgment to the client is by failing the connection. Also the technique applies only to authentication from a web browser. It doesn't help with other uses (such as code signing).

The third technique, which we have also prototyped, is to monitor the log written by the Windows CAPI library. This approach works for all usages, including code signing. For example, applying a suitable policy based on this log would have detected the Flame malware. The downside here is that the results are offline: the relying party has made its decision asynchronously. But from the point of view of an enterprise administrator, this provides information about attacks and failures at least as well as the other techniques, and the increased coverage is very valuable.

5. Conclusion

Global computing requires a global authentication system with trust relationships that do not follow a single global hierarchy. The X.509 PKI offers an appropriate structure for that, but as presently deployed is demonstrably lacking in security.

Standards bodies (such as the CABForum or IETF) can help, by setting requirements for proper behavior by CAs [3], and enforcing them in all the popular web browsers (and other clients). But that fails in two ways:

- The system is large and dynamic. Unless we can analyze and understand it, and use that understanding to adjust authentication decisions, our security will fail.
- Authentication decisions are not absolute. Different users or organizations have different trust relationships, and the authentication system must support these. The relying party's trust decisions must be made locally, not globally.

We believe the ideas outlined here are a start towards providing more secure and more flexible global authentication.

References

1. A. Birrell, B. Lampson, R. Needham, M. Schroeder. A Global Authentication Service without Global Trust. In *IEEE Symposium in Security and Privacy*. 1986.
2. CA/Browser Forum. Guidelines For The Issuance And Management Of Extended Validation Certificates, v.1.4. https://cabforum.org. 2012.
3. CA/Browser Forum. Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates, v.1.1. https://cabforum.org. 2012.
4. S. Chockani, W. Ford, R. Sabett, C. Merrill, S. Wu. Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework. http://tools.ietf.org/pdf/rfc3647.pdf. 2013.
5. D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. http://tools.ietf.org/pdf/rfc5280.pdf. 2008.
6. P. Eckersley, J. Burns. The EFF SSL Observatory. https://www.eff.org/observatory. 2010.
7. C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, T. Ylonen. SPKI Certificate Theory. http://tools.ietf.org/pdf/rfc2693.pdf. 1999.
8. C. Evans, C. Palmer, R. Sleevi. Public Key Pinning Extension for HTTP. http://datatracker.ietf.org/ 2012.
9. Fox-IT BV. Black Tulip Report of the investigation into the DigiNotar Certificate Authority breach. http://www.rijksoverheid.nl/bestanden/documenten-en-publicaties/rapporten/2012/08/13/black-tulip-update/black-tulip-update.pdf. 2012.
10. D. Goodin. Crypto breakthrough shows Flame was designed by world-class scientists. http://arstechnica.com/security/2012/06/flame-crypto-breakthrough/. 2012.
11. P. Hoffman, J. Schlyter. The DNS-Based Authentication of Named Entities (DANE). http://tools.ietf.org/pdf/rfc6698.pdf. 2012.
12. B. Lampson, M. Abadi, M. Burrows, E. Wobber. Authentication in distributed systems: theory and practice. In *ACM TOCS* 10(4). 1992.
13. S. Mendes, C. Huitema. A New Approach to the X.509 Framework: Allowing a Global Authentication Infrastructure without a Global Trust Model. In *Proceedings of the Symposium on Network and Distributed Systems Security.* 1995.
14. Microsoft. Flame malware collision attack explained. http://blogs.technet.com/b/srd/archive/2012/06/06/more-information-about-the-digital-certificates-used-to-sign-the-flame-malware.aspx. 2012.
15. D. Wendlandt, D. Andersen, A. Perrig. Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing. In *Proc. USENIX Annual Technical Conference.* 2008.