New Standards for Browser-Based Trust
The Recent Acceleration of Improvements

Tom Ritter — tritter@isecpartners.com

iSEC Partners, Inc
123 Mission Street, Suite 1020
San Francisco, CA 94105
https://www.isecpartners.com

March 15, 2012

Abstract

Improvements to Browser Security, TLS, and Public Key Infrastructure have been appearing at an astonishing rate in the past few years. Standards are proposed in the IETF in Web Security, Public Key Infrastructure, TLS, and DNS Working Groups and similarly in the W3C Working Groups. Proposals for replacing Certificate Authorities, or improving them, are presented alternately on any one of two dozen mailing lists, at conferences on opposite sides of the globe, and in standards bodies. This paper first presents a survey of improvements being made to Browsers, HTTP, and Javascript, noting what will be effective and what won't. Methods for leveraging DNSSEC for improved authentication for different protocols are covered and improvements to TLS are presented covering protocol revisions, extensions, and techniques for using it to improve identity management. Finally a survey of all the proposals about replacing or improving CAs is done, and the commonalities and core concepts of them are drawn out and presented.

This paper is likely to undergo editorial and content improvements.
The most recent version will be available at http://ritter.vg/p/2012-TLS-Survey.pdf.

1 Introduction

When speaking with individuals participating throughout this ecosystem - administrators of web sites, the engineers behind browsers, researchers testing, breaking, and deploying protocol enhancements - a common phrase has been heard when speaking of the past few years: "Things are moving really fast." While it may seem that there is tremendous inertia of web servers not upgrading and Certificate Authorities not changing their business practices (and there is), literally dozens of improvements have been made in browsers and standards that incrementally improve security for users, and have the potential to evolutionarily change how we perform authentication on the web. While not encompassing all of the work done by the IETF and W3C (a book would be needed), the most user-centric improvements to Browser Security, the TLS Protocol, and Public Key Infrastructure are covered.

2 Browser Security

In the last several years we've seen an explosion of web-based vulnerabilities being exploited: the most common four being cross site scripting, SQL injection, cross-site request forgery, and clickjacking. Clickjacking, reflected cross-site scripting, and cross-site request forgery are attacks on a user, and therefore must be presented in a browser. All three can be prevented by the developers of the web applications, and while high profile websites are legitimately concerned about these flaws and take efforts to resolve them - innovation is actually more common in browser manufacturers. In recent years browsers have been working on several protection mechanisms, and have been aggressive about incorporating security

updates. While a long tail of users who do not upgrade their browsers can hold some of these protections back, overall bleeding edge versions of browsers often contain protection mechanisms even security professionals are unaware of. The number of protection mechanisms is eclipsed only by their complex interaction and implementation differences across browsers. Excellent resources are The Tangled Web[1] by Michal Zalewski and an IETF draft on Web Security[2].

2.1 Content Security Policy

Content Security Policy (CSP)[3] is a HTTP Header that specifies where additional content on a page is permissible to be loaded from. After receiving a web resource such as an index.html page, the browser will find references to other resources required - images, audio or video files, fonts, further pages in iframes, stylesheets, and javascript files. Additionally, running javascript may load these resources dynamically or make AJAX requests. CSP allows a server to dictate to the browser what origins are permitted when making requests. Each of eight content types may have their own settings: images, stylesheets, javascript files, iframe locations, fonts, <object> tags, media (<audio> and <video>), and finally AJAX requests. A sample Content Security Policy is:

```
Content-Security-Policy:
   default-src 'self';
   img-src *;
   script-src 'self', https://api.google.com
```

This example shows that images may be loaded from all locations, all resources except javascript files are permitted only from the original origin, and javascript files are permitted from the original origin and from resources loaded over HTTPS from api.google.com. Content Security Policy is near standardization, and experimental implementations exist in Chrome, Firefox, Safari, and Internet Explorer 10.

Besides disallowing content from being loaded from arbitrary locations, CSP has a reporting mechanism. If the setting `report-uri` is specified, a browser encountering a policy violation, in addition to blocking the content loading, will send a report to the URI specified containing details, as shown in Figure 1. This can be useful to detect targeted attacks as well as incidents of untargeted defacements, particularly common to off-the-shelf installs like Wordpress, where a bot will crawl the Internet looking for vulnerable installs to automatically infect with malware. A second header called `Content-Security-Policy-Report-Only` exists. In the presence of a Report-Only header, the browser will not block the violating content from loading, but will issue the same report violation. This allows server administrators to deploy a CSP policy to a production environment for testing, with no impact to the users, and refine the policy to determine what origins are in use that they may not have been aware of. Furthermore, it is possible to deploy both headers simultaneously, which can be used to enforce a broader policy but test a more restrictive one to see if it blocks legitimate resources.

```
{
"csp-report":
 {
 "request": "GET http://index.html HTTP/1.1",
 "request-headers":"Host: example.com
                    User-Agent: Mozilla/5.0
                    Accept: text/html;q=0.9,*/*;q=0.8
                    Accept-Encoding: gzip,deflate
                    Accept-Charset: *;q=0.7
                    Keep-Alive: 115
                    Connection: keep-alive",
 "blocked-uri": "http://evil.com/some_image.png",
 "violated-directive": "img-src 'self'",
 "original-policy": "allow 'none'; img-src 'self'"
 }
}
```

Figure 1: A sample Content Security Policy report violation that would be issued by a browser.

Two settings of `script-src` should be mentioned. In addition to specifying web origins, it is possible to specify `unsafe-inline` and `unsafe-eval`. When unsafe-inline is *not* present in the script-src set, a browser is disallowed from executing javascript contained in inline <script> tags, as well as attribute-based event handlers such as onclick or onmouseover. Most websites are not architected such that inline javascript can be wholly disallowed, thus the setting will be enabled on most deployments. However, if the effort to make a website unsafe-inline compliant is done, the risk of an exploitable XSS flaw goes down considerably because HTML injection flaws cannot be trivially exploited. Unsafe-eval prevents the javascript runtime from executing any string variable as javascript code, which (as far has been determined) is possible in only four places: the commonly known eval() function, setTimeout, setInterval, and the Function constructor. This setting will significantly strengthen a site against javascript malware, which by default uses heavily packed and obfuscated code that makes use of eval()

[1] http://nostarch.com/tangledweb.htm
[2] https://trac.tools.ietf.org/html/draft-hodges-websec-framework-reqs
[3] http://www.w3.org/TR/CSP/

extensively.

Content Security Policy can be likened to Data Execution Prevention (DEP), an exploitation mitigation setting for binary executables. DEP prevents code in a read-only memory page from being executed - most commonly preventing shellcode injected through a stack or heap buffer overflow from being run. CSP (with unsafe-inline disallowed) likewise prevents injected javascript from being executed. DEP has a bypass called Return-Oriented Programming, or ROP. In a ROP attack, existing executable code is reused to perform actions desired by the attacker. It's possible to circumvent CSP by using a technique similar to ROP: reusing existing javascript on the page or allowed domains to perform actions unintended by the developer. Other methods of bypassing CSP include exploiting DOM-based XSS, and taking advantage of user-controlled files on the server. Additionally, javascript can still be obfuscated and packed without the use of eval(). CSP should not be considered a complete solution to XSS, but is a good tool for exploit mitigation.

2.2 Caja

Caja[4] is a mechanism to sandbox untrusted javascript and have it run safely within a web page. Ordinarily if a website wanted to host modules or third-party code in its site such as games, it would embed the content in an iframe and have the browser enforce the same-origin policy. Caja however runs the untrusted javascript within the context of the page itself. Ordinarily this would allow the untrusted code to poison the javascript runtime, but after untrusted javascript is *cajoled*, a process that is similar to compiling, it is unable to alter the global runtime of the page.

Caja uses an object-capability security model. In an object-capability model, capabilities are granted by providing access to objects encompassing those capabilities. A cajoled module starts sandboxed, but gains capabilities as objects are passed to it - as an example ordinarily a module would be unable to access the XMLHttpRequest object to make AJAX requests, however it may be passed into the module and used. To restrict capability, it is also possible to pass a class that wraps XMLHttpRequest and only allows requests to a specific domain.

2.3 Javascript Cryptography

There have been a number of attempts recently at producing javascript-based cryptography. Without going into details on any single implementation, all suffer from the same flaws with regards to the javascript runtime in browsers. A number of security professionals have come out explaining why javascript-based cryptography will ultimately never be trustworthy. An attempt at enumerating the problems faced follows.

1. Third-Parties: Any third party supplying javascript to a page can alter the runtime
2. XSS: Any XSS flaw can alter the runtime
3. MITM: Without SSL the code can be easily modified in transit
4. SSL: SSL Authentication is currently not reliable
5. Modified: With each new page load, the runtime is rebuilt and must be revalidated
6. RNG: Not all implementations of javascript have a secure random number generator
7. Implementation: Cryptographic primitives are often implemented with subtle bugs that undermine all security
8. Keystore: Where do you keep your keys?
9. Side Channels: Keys or sensitive state may be exposed by timing and other attacks
10. Coercible: Has the entity you used to trust still trustworthy?
11. Caching: Has your cache been tampered with or poisoned?

It's clear that javascript cryptography will continue to be attempted and implemented, so the W3C has formed a working group[5] aimed at solving some of these problems. The working group is in its early stages and has not created a charter as of yet.

2.4 Strict Transport Security

HTTP Strict Transport Security (HSTS)[6] is a HTTP header that indicates a site should only be contacted over HTTPS. A browser will store this policy, and all HTTP requests to the server will automatically be routed over HTTPS. This header is designed to prevent SSL stripping attacks, like those performed by sslstrip[7]. In a SSL stripping attack, an attacker will perform a man in the mid-

[4] https://code.google.com/p/google-caja/
[5] http://www.w3.org/2011/11/webcryptography-charter.html
[6] http://tools.ietf.org/html/draft-ietf-websec-strict-transport-sec
[7] http://www.thoughtcrime.org/software/sslstrip/

dle attack on traffic between a client and server, speaking HTTPS to the server, but HTTP to the client. All references to SSL, including stylesheets, javascript files, links, and the Secure cookie attribute, will be removed or rewritten. Because no error will be presented to the user, only the most astute will notice that their connection is not using SSL. In testing, the percentage of people who notice stripping attacks is low. A sample header looks like:

```
Strict-Transport-Security: max-age=31536000;
     includeSubDomains
```

2.5 Public Key Pinning

Public Key Pinning[8] is a mechanism for a server to specify what public keys (contained in x.509 certificates) are acceptable for the server. Hashes of the public keys are stored on the client for a server-specified length of time, and subsequent TLS negotiations will contain a whitelisting check against those previously-specified keys. A sample header[9] looks like:

```
Public-Key-Pins: max-age=31536000;
     pin-sha1="4n972HfV354KP560yw4uqe/baXc=";
     pin-sha256="LPJNul+w4m620714DsqxbnJecwQzYpOLmCQ="
```

This header specifies two hashes of public keys (encoded in base64), and a caching time of 31536000 seconds (1 year). The server operator has the ability to pin any public key in the certificate chain. Pinning leaf certificates allows a server operator to whitelist specific certificates they are committed to using. Pinning an Intermediate or Root Certificate effectively ties the server to a single Certificate Authority. Public Keys, instead of certificates themselves, are pinned so a new certificate may be issued with the same public key. Additionally, some proposed changes to TLS involve recreating a certificate regularly, changing its fingerprint.

Public Key Pinning has already won a battle in the war on fraudulent certificates. Google Chrome ships with an implementation of pinning - certain Google domains are hardcoded to only accept certain certificate authorities. In July, 2011 Diginotar, a small Dutch Certificate Authority, was hacked or otherwise tricked into issuing over 500 fraudulent certificates, including certificates for high-visibility organizations like Google, Yahoo, Mozilla, Wordpress, and the Tor Project. A particularly widely encompassing certificate for *.google.com was used in August, 2011 in Iran against hundreds of thousands of individuals. One individual in Iran using Google Chrome with an understanding of TLS detected the fraudulent certificate thanks to Chrome's key pinning and posted about it on a Google support forum[10] which kicked off a wide and fast investigation that ultimately resulted in the Dutch government taking over operational control of Diginotar's systems, the company being distrusted in every major browser, and declaring bankruptcy.

However, the ability for server administrators to effectively perform a denial of service attack against their users using Public Key Pinning is high. If a single key is specified that should be used for a year, when the SSL certificate must be renewed, it must be renewed using the same public/private keypair as before, or the new certificate will be rejected by the regular visitors for the next 364 days. If the key is compromised and an immediate replacement must be put in place, that same denial of service is in effect. Therefore for browsers to comply with this header, two hashes must be provided. A second key is required and intended to be a certificate kept as an offline backup able to be deployed if needed.

The current draft specifies a HTTP header that will be sent to browsers. However, key pinning is a concept that can be applied to TLS in general - not just in browsers. A rough proposal to move Key Pinning into the TLS protocol itself exists in TACK[11], or Tethered Assertions for Certificate Keys. This outline was authored by Moxie Marlinspike and Trevor Perrin, but has not been proposed to the standards bodies as of yet.

2.6 Bootstrapping Trust

Both Strict Transport Security and Public Key Pinning rely on sending information down the same channel (HTTP) they are ultimately trying to secure. If a client contacts a server for the first time, in the presence of an active attacker, the attacker is able to remove both headers and completely circumvent the intended security. Google Chrome will preload keys and sites as HSTS currently, although this practice is not scalable. Aside from this preloading, neither HSTS nor Key Pinning attempt to addresses this deficiency; instead relying on the fact that active attacks are relatively rare and it's likely that sites will be visited over a trustworthy network at some point.

[8] http://tools.ietf.org/html/draft-ietf-websec-key-pinning-01
[9] Some hashes throughout the document have been abbreviated to prevent wrapping.
[10] http://www.google.co.uk/support/forum/p/gmail/thread?tid=2da6158b094b225a&hl=en
[11] https://github.com/moxie0/Convergence/wiki/TACK

Additionally, as these are mechanisms for caching data on the client, they can be used for tracking users across sessions, even if cookies are cleared. A creative server could use dynamic subdomains to store HSTS information or Pinned Keys, and retrieve that information later. With HSTS, the attacker would look for the request occurring in a TLS connection or not; with Public Key Pinning, sending an invalid certificate and seeing if a TLS connection is established or refused. Currently, browsers are opting to discard HSTS or Pinned Keys at the end of a Private Browsing Session. However, if HSTS and Pinned Keys are discarded regularly, an attacker has more opportunities to attack a user when they have no stored security information for the site. This is a serious concern for users who use a Private Browsing mode by default, or read-only operating systems[12] that are designed to increase security.

Finally, corporations often install a trusted root in the browsers or operating systems of its employees. After a network appliance is installed, a man in the middle procedure is initiated, and the installed root will be used to create trusted certificates for secure sites communicated with by users; no untrusted certificate warnings will be triggered. Without debating the ethicacy of the practice, current Key Pinning implementations allow locally installed roots to override Public Key Pins - the standard remains silent on the matter.

3 DNSSEC

The general technique people refer to when talking about using DNSSEC for authentication is that when a DNS resolver is contacted to retrieve the IP address for a server (A or AAAA records), other record requests will be made as well, and a certificate or public key fingerprint or hash will be retrieved. DNSSEC is used to ensure the responses were not tampered with. The certificate (or fingerprint thereof) is obtained through two channels, authenticated through different trust chains. An adversary seeking to impersonate a server would need to circumvent the trust authorities of two different channels.

3.1 DNSSEC-Verified Fingerprints

The most well known proposal relating to DNSSEC is DANE[13], or DNS-based Authentication of Named Entities, which proposes to put certificate hashes of websites into DNS. The client, most commonly a browser, will assert the certificate obtained over the HTTPS channel matches the one obtained over DNSSEC. Ideally DANE will allow a server administrator to assert the validity of a self-signed certificate, without needing the signature of a Certificate Authority. That is only one use however, DANE can also be used to assert the validity of a CA-signed leaf certificate, the Certificate Authority itself, or even a Certificate Authority unknown to the client.

Another mechanism for asserting the validity of a certificate provided over HTTPS via DNS is something dubbed DNSSEC Stapled Certificates[14] which is only present in Chrome, with no formal commitment to support it long-term. More complicated than the DANE proposal, DNSSEC Stapled Certificates requires a DNS record be created with certificate information, same as DANE. But after the record is created, the certificate must be rebuilt and have the DNSSEC chain embedded inside the certificate itself as an extension. Because DNSSEC information expires, the certificate will have to be rebuilt frequently and reloaded into the webserver. Frequency depends on DNSSEC expiration, but weekly is common. Although more complicated, one advantage of embedding the DNSSEC chain in the certificate is the assertion of two channels inside a single one - requiring no additional lookups or a DNSSEC resolver. Currently the DNS record used is actually an overload of the CAA proposal covered in Section 5.2, although this may change to a proper DANE record in the future.

Besides the certificate for a webserver, there are several other types of fingerprints we can assert with DNS. RFC 4255[15] allows assertion of SSH fingerprints using DNS. When used with `ssh -o "VerifyHostKeyDNS yes"` this will suppress the request to verify the fingerprint seen when connecting to a server for the first time *if* the fingerprint provided via DNS matches the one sent by the server. There is a way to similarly retrieve OpenPGP keys via DNS[16], and allow people to find the key via the command line argument `gpg --auto-key-locate pka -r user@example.com`. Finally, for individuals

[12] An example of a designed-for-privacy operating system is Tails https://tails.boum.org/.
[13] http://tools.ietf.org/html/draft-ietf-dane-protocol
[14] http://www.imperialviolet.org/2011/06/16/dnssecchrome.html
[15] http://tools.ietf.org/html/rfc4255
[16] http://www.gushi.org/make-dns-cert/HOWTO.html
[17] http://tools.ietf.org/html/draft-hoffman-dane-smime

using S/MIME, there is a draft[17] to assert S/MIME certificates via DNS.

3.2 DNSSEC Trust Chain

When speaking about performing trust decisions via DNSSEC, it is important to note the implicit trust decisions in DNSSEC itself. The fingerprints are signed with the server's Zone Signing Key, which in turn is signed by the server's Key Signing Key. But the server's key is authenticated by the organization that runs the top-level domain the domain sits under - example.com being under .com run by Verisign, example.org in .org run by the Public Interest Registry, example.info in .info is run by a company called Afilias. Country-code TLDs such as .vg (British Virgin Islands) are run by companies that are generally within the jurisdictions of the country in question (although not always.) These organizations are in turn authenticated by the root key, controlled by ICANN, a US-based company. Therefore by trusting any DNSSEC validated information, ICANN and the organization running the TLD are also trusted not to have subverted the information.

4 TLS

TLS began as SSL, pioneered at Netscape in the 1990s. Regrettably, the earliest deployed version of the protocol, SSL v2, can still be found supported in some dark corners of the web. But TLS 1.0 was standardized in 1999 with minor changes from SSL v3, and has been widely implemented. The overall protocol has undergone several revisions and dozens of extensions and improvements. TLS 1.1 and 1.2 are covered primarily because they have not been widely implemented. Other TLS extensions, such as the Server Name Extension, are omitted not because of lack of impact (indeed SNI is arguably the most important TLS extension deployed), but because with the exception of the TLS renegotiation bug, only the more recent TLS improvements are aimed at improving securing of the protocol, as opposed to adding features.

[18] https://tools.ietf.org/html/rfc4346
[19] http://ekoparty.org/2011/thai-duong.php
[20] https://tools.ietf.org/html/rfc5246

4.1 TLS Protocol Improvements

4.1.1 TLS 1.1

TLS 1.1 was defined in April 2006 in RFC 4346 [18] and was a relatively lightweight set of changes to the TLS protocol. The most significant changes were around mitigating risks in CBC mode. A theoretical flaw in the CBC implementation of TLS 1.0 (and SSL 3.0) relating to predictable IVs was put forward earlier in the decade, and solved resolutely in TLS 1.1. However this flaw did not gain prominence until Sept 2011 when a practical attack was demonstrated by Juliano Rizzo and Thai Duong at the ekoparty security conference [19], where they were able to decrypt parts of the TLS stream exposing authentication cookies.

TLS 1.1 also improves session resumption performance by allowing resumption even if the previous connection wasn't closed cleanly. Finally, it dictates that export ciphers (40-bit RC4 and DES) must not be permitted. It also adds security notes relating to timing attacks on RSA operations and the necessity of blinding, Diffie-Hellman small subgroup attacks, and TLS' mac-then-encrypt scheme.

4.1.2 TLS 1.2

Compared to TLS 1.1, TLS 1.2, defined in Aug 2008 in RFC 5246 [20], contained a significant number of changes. The simplest change were revising the default must-implement ciphersuite from RSA-3DES-CBC-SHA1 to RSA-AES-128-CBC-SHA1. It also added SHA-256-based ciphersuites, providing an upgrade option from SHA-1. Although TLS 1.1 did close the CBC vulnerability, TLS 1.2 finishes the hardening by requiring a fully random IV and uniform error codes.

TLS defines a psuedorandom function (PRF) for secret expansion. Prior to TLS 1.2, the PRF was a combination of MD5 and SHA-1; in TLS 1.2 it is simplified to be SHA-256. A significant change between TLS 1.1 and 1.2 was the addition of the signature_algorithms extension, which is sent by a client to indicate what signature and hash algorithm pairs may be used in digital signatures. This extension adds considerable confusion to the protocol, as a server may have a single certificate signed using RSA, but a client may choose to omit RSA as an acceptable signature.

The last significant change in TLS 1.2 was the addition of AEAD cryptographic modes of operation, and the data structure changes necessary to support them. AEAD stands for Authenticated Encryption with Associated Data. Authenticated Cipher Modes can tell when the ciphertext has been manipulated or corrupted in transit during the decryption operation; compared to CBC mode which can make no assertions about the integrity of the data, and requires an outside message authentication code to provide integrity. While TLS already provides integrity by MAC-ing the data prior to encryption, AEAD ciphermodes add another layer of protection and may remove side channels relating to adaptive ciphertext attacks. Prior to TLS 1.2, the only secure ciphers available were RC4 (a stream cipher) and block ciphers using CBC mode. The support for AEAD cipher modes allowed the later addition of new block cipher modes (covered in Section 4.1.4).

Experts believe one of the reasons for the extremely low public deployment of TLS 1.2 is the number and breadth of changes which do not provide significant benefit[21].

4.1.3 TLS Deployment

TLS 1.1 and 1.2 are not implemented or deployed widely. Of major libraries, support exists only in the schannel implementation in Windows 7/2008R2, gnuTLS, and OpenSSL 1.0.1 (currently beta). Server support therefore requires IIS 7.5 on Windows Server 2008R2, or Apache using mod_gnutls. Among browsers, only Opera and Internet Explorer (on Windows 7/2008R2) have support, and it must be enabled manually. Notably, support is missing from NSS, the cryptographic library used by Firefox and Chrome.

	TLS 1.0	TLS 1.1	TLS 1.2
XP	✓		
Vista/2008	✓		
Win7/2008R2	✓	✓	✓
OpenSSL	✓	1.0.1	1.0.1
gnuTLS	✓	✓	✓
NSS	✓		

Table 1: TLS Version Support in Libraries

	TLS 1.0	TLS 1.1	TLS 1.2
IIS6	✓		
IIS7	✓		
IIS7.5	✓	Manual	Manual
mod_ssl	✓		
mod_gnutls	✓	✓	Manual

Table 2: TLS Version Support in Servers

	TLS 1.0	TLS 1.1	TLS 1.2
IE XP	✓		
IE Vista/2008	✓		
IE Win7/2008R2	✓	Manual	Manual
Opera	✓	Manual	Manual
Safari	✓		
Chrome	✓		
Firefox	✓		

Table 3: TLS Version Support in Browsers

A much more complete comparison of TLS libraries with individual feature comparisons is available on Wikipedia[22].

Because of the lack of support in libraries, especially OpenSSL, deployment is minimal. There are a few sources of data, including surveys done by Opera in April 2010[23], the EFF Observatory [24], and Qualys in April 2011[25]. All contain slightly different numbers, but all round to 0% of servers supporting TLS 1.1 or 1.2.

Besides lack of support in libraries, there are misbehaving implementations that reduce the security of TLS deployment. An excellent summary of TLS practices is present in [26] and includes TLS servers that are intolerant of TLS versions greater than 1.0, compression algorithms, and a relatively high number of servers intolerant to any extensions (7%). If an initial handshake fails, nearly all HTTPS clients will enter a fallback mode using SSLv3, with no compression or extensions. This is particularly damaging to the server_name extension, where a client may be presented with a different SSL certificate that does not match the hostname requested. From a security perspective, this behavior will always enable a downgrade attack. A client is unsure if the handshake failure was a result of a misbehaving server or if an adversary inserted the failure alert to provoke the client to

[21] https://mail1.eff.org/pipermail/observatory/2011-August/000281.html
[22] https://en.wikipedia.org/wiki/Comparison_of_TLS_Implementations
[23] http://my.opera.com/securitygroup/blog/2010/04/06/how-secure-is-the-secure-web-ssl-tls-server-stats-part-1
[24] https://mail1.eff.org/pipermail/observatory/2011-August/000290.html
[25] http://blog.ivanristic.com/Qualys_SSL_Labs-State_of_SSL_InfoSec_World_April_2011.pdf
[26] http://www.imperialviolet.org/2011/02/04/oppractices.html

retry with less secure options. Until HTTPS client remove the downgrade behavior this vulnerability will always exist, and the security benefits of negotiating TLS 1.1 or 1.2 will not be realized when the threat model includes an active attacker.

4.1.4 TLS Ciphersuites

TLS has approximately 300 different ciphersuites defined, containing combinations of Key Exchange algorithms, Ciphers, Cipher Modes, and Hash Functions. (The definitive list of standardized ciphersuites is maintained by the IANA[27].) There are a number of less common ciphers present that may be negotiated depending on client and server implementation: including the SEED, ARIA, and Camellia block ciphers. The set of Russian algorithms known as GOST, consisting of a block cipher, hash function, and digital signature algorithm, was proposed but never standardized, although may be enabled in OpenSSL[28]. Ciphersuite additions are regularly proposed; most recently CLEFIA, a lightweight block cipher[29].

Key Exchange algorithms have also been added, for example RFC 4492[30] that added support for Elliptic Curve Cryptography. A less algorithmic key exchange method added was Pre-Shared Keys, in RFC 4785[31]. Finally, block cipher modes have also been added, with more regularity. RFC 5288[32] added GCM, a new cipher block cipher mode for AES - much desired with the general uneasiness around CBC mode. GCM mode is the first AEAD mode added to TLS, and will provide integrity for the data encrypted with it as part of the cipher mode. Likewise, the proposed AES-CCM draft[33] will also add an authenticated block cipher mode. However, because these are AEAD modes, they will only be available in TLS 1.2. Hash Functions in use are almost universally SHA-1, although SHA-256 and SHA-384 ciphersuites are defined for newer ciphers.

TLS was designed to provide three things: Authentication, Confidentiality, and Integrity. It is interesting to observe that in pursuit of a single protocol that provides everything, the option to ignore either Authentication, Confidentiality, or both is present. Although not widely used, TLS is defined with the DH_anon key exchange algorithm, which performs no authentication. One could say you're not sure who you're talking to - but you know it's encrypted! Similarly, there are NULL ciphersuites defined that provide no confidentiality, but do provide authentication and integrity. These NULL ciphersuites (along with the Export ciphers), have formed the basis for a relatively common line item in vulnerability reports.

4.1.5 False Start

False Start[34] is a change in the TLS handshake designed to speed up page loading for clients. Ordinarily a TLS handshake requires the client to wait twice: once after the ClientHello, and once after the ClientKeyExchange. This means to begin receiving secure page content for a website, a client must wait for four responses from the server: the TCP SYN/ACK, two responses in the TLS handshake, and the final HTTP Response containing the page contents. Removing one of the waits for the TLS handshake has a measurable improvement in page load time.

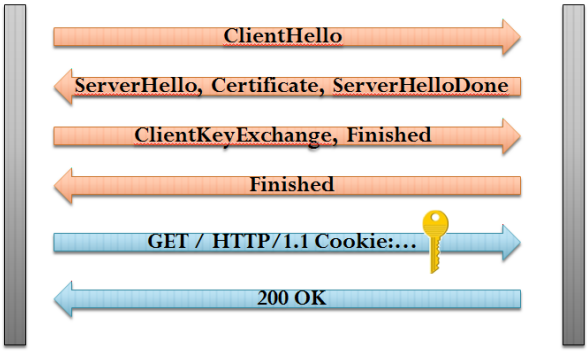

Figure 2: Ordinary TLS Handshake

The last piece of the TLS handshake is the server sending an encrypted Finished record. This record is used to verify the handshake was not tampered with, and therefore is critical to the security of the protocol. False Start reorders the handshake so the client sends application data prior to receiving the server's Finished message.

[27] http://www.iana.org/assignments/tls-parameters/tls-parameters.xml#tls-parameters-3
[28] http://www.openssl.org/docs/apps/ciphers.html#GOST_ciphersuites_from_draft_chu
[29] https://datatracker.ietf.org/doc/draft-katagi-tls-clefia/
[30] http://tools.ietf.org/html/rfc4492
[31] http://tools.ietf.org/html/rfc4785
[32] http://tools.ietf.org/html/rfc5288
[33] http://datatracker.ietf.org/doc/draft-mcgrew-tls-aes-ccm/
[34] https://tools.ietf.org/html/draft-bmoeller-tls-falsestart

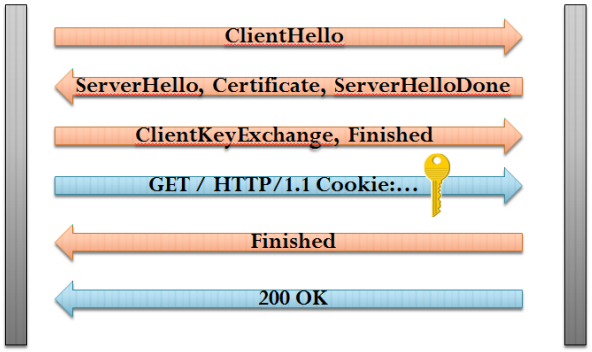

Figure 3: False Start Handshake

An active attacker is able to manipulate the client or server responses without either knowing until after the encrypted data has been sent by the client. The most practical attack is to downgrade the ciphers supported by the client (or similarly, the cipher chosen by the server). If the client supports the weaker ciphers, like 40 Bit RC4 or 56 Bit DES, they will weakly encrypt the first application request (often containing cookies or a username/-password). The attacker is then able to quickly brute force the decryption key and retrieve the encrypted data. Depending on the choice of the attacker, the user will receive a timeout or an error in the browser.

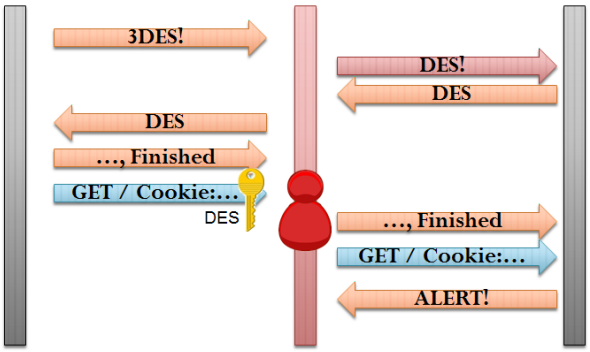

Figure 4: Cipher Downgrade Attack

To defend against this attack, False Start is dictated to not support sending data using any cipher with a key less than 80 bits. Additionally, a small minority of TLS implementations are compatible with false start. Sites using these incompatible implementations are maintained in a publicly available blacklist in the Chromium source code repository[35]. A different revision to the TLS Handshake named Snap Start was put forward by Google at the same time as False Start, however Snap Start proved too complicated and was withdrawn. An analysis of Snap Start is present on the Root Labs blog[36].

4.1.6 Next Protocol Negotiation

While many application protocols are layered over TLS, until recently browsers only layered HTTP over TLS. However two new protocols are becoming more popular: WebSockets and SPDY. Browsers can provide a great deal of control to a determined attacker, and relatively strange attacks have been carried out by inventive security researchers, including Chosen Protocol Attacks. In a Chosen Protocol Attack, an attacker tricks the browser into speaking to a service that speaks a different protocol than the browser thinks it is. This can result in security bypasses or information disclosure.

```
POST / HTTP/1.1
Referer: http://i8jesus.com/stuff/xps/test.html
Content-Type: multipart/form-data;
              boundary=--------------------7da70534
...
Cookie: <snip>
--------------------7da70534
Content-Disposition: form-data; name="doesntmatter"

USER anonymous
--------------------7da70534
Content-Disposition: form-data; name="doesntmatter"

PASS a@a.com
```

Figure 5: A Chosen Protocol Attack where a browser is tricked into sending valid FTP commands.

A Chosen Protocol Attack is shown in Figure 5, where a browser is tricked into POSTing data to a FTP server. Most FTP servers will ignore commands they do not recognize, so the HTTP headers will be ignored, but **`USER anonymous`** will be interpreted by the FTP server as a legitimate FTP command to log in anonymously.

Next Protocol Negotiation[37] is a TLS addition that eliminates the possibility of Chosen Protocol Attacks when TLS is the wrapper for another protocol and is negotiated prior to data being sent. It adds a TLS extension and a handshake message type that specify what the next protocol (layered inside of TLS) will be.

[35] http://src.chromium.org/viewvc/chrome/trunk/src/net/base/ssl_false_start_blacklist.txt?revision=123482&view=markup

[36] http://rdist.root.org/2012/02/27/ssl-optimization-and-security-talk/

[37] http://www.ietf.org/mail-archive/web/tls/current/msg05593.html, http://tools.ietf.org/html/draft-agl-tls-nextprotoneg

4.2 Identity Management in TLS

A full accounting of all that falls under the broad term of "Identity Management" would require a binding for the paper rather than a staple; instead we are focusing only upon improvements to TLS related to managing identity. Pointers to other new forms of identity management across multiple websites are provided.

4.2.1 Channel Binding

Channel Binding is a cryptographic concept (explained partially in RFC 5056[38]) that allows applications to assert that two endpoints communicating at one layer are the same endpoints communicating at a lower layer. Consider a scheme similar to HTTP Basic Auth, where a client authenticates to the server by sending a hash of their password to the server with every request. The server knows the plaintext password, performs the same hash operation, and compares their hash with the client-provided hash. It is trivial to perform a man in the middle attack against this protocol; simply read the packet as it crosses the wire - it is not encrypted. Now layer the protocol inside of TLS. The protocol itself is no more secure, only the transport mechanism adds the security of TLS. However, if an attacker can middle the TLS traffic, for example through a forged certificate, the protocol can be middled once again. The endpoints of the imaginary protocol (client and server) are not the endpoints of the TLS protocol (client and attacker, attacker and server.)

To improve security through channel binding the client will send the password and the client's view of the TLS Finished Record concatenated together and then hashed, instead of just sending the hashed password. In a secure channel, both the server and client will have the same Finished bytes, and thus can do the same hash calculations. In a middled channel, the client has one sequence of Finished Bytes with the attacker, and the server has a different sequence of Finished Bytes with the attacker. The hashes will not match, and the authentication will fail. This authentication is thus bound to the lower layer. Channel Binding is a concept independent of TLS, but it can be used with TLS, and RFC 5929[39] defines three channel binding types for TLS: tls-unique, tls-sever-endpoint, and tls-unique-for-telnet.

There's been some discussion recently about using channel binding to protect cookies in a web browser; however, because cookies are not an authentication mechanism in the sense of channel binding, the protections afforded are not as strong. If a cookie was bound to a TLS Session (specifically using tls-unique which is the TLS Finished message), that cookie could not be used in any other TLS session. This means if the cookie was stolen, for example through a Cross Site Scripting attack, the cookie would be useless to the thief, because the server would attempt to validate the channel binding, and reject the cookie because it was not bound to the TLS session used by the thief.

However, there are two problems with binding cookies to a TLS Session. The first problem is it does not work on first contact. We are authenticating to the site using a username and password and this authentication is *not* channel bound. An attacker performing a man in the middle attack sees the credentials go across, and then the cookie is set in response to a valid authentication. That cookie will be bound to the *attacker* rather than us, and when the attacker passes the cookie, the server will see the binding as valid. With web cookies, there is no way to validate the binding on the client side. The second problem is a bound cookie does not protect against server impersonation. If a client contacts an attacker thinking it is their bank, the attacker will receive the channel bound cookie. The attacker will be unable to use the cookie for authentication with the legitimate server, but all the attacker has to do is present the user with a login page and the message "For your security, we've logged you out from inactivity, please login again." The user will enter their credentials and send them off to the attacker.

Furthermore, if a cookie is bound to a TLS session, the cookie can only be used in that TLS session, which breaks persistent cookies. There are other options for binding however. One possibility, not defined in an RFC but still possible, is binding the cookie to a client certificate. A cookie bound to a client certificate could be used across TLS sessions, because the same client certificate will be used in the TLS protocol. And a cookie thief or man-in-middle would be unable to create a TLS session using a client certificate because they would need the corresponding private key - which is a much more difficult thing to steal. However, client certificates are not widely used, and for good reason. Client Certificates require a confusing enrollment process, have privacy implications, portability problems, and poor browser UI (coupled with the inability for a site to customize that UI).

[38] http://tools.ietf.org/html/rfc5056
[39] http://tools.ietf.org/html/rfc5929

4.2.2 Origin Bound Certificates

A proposal to improve Client Certificates is BrowserAuth.net[40]. It covers many of the problems inherent in client certificates[41], and proposed a modification to them called Origin-Bound Certificates[42]. An Origin Bound Certificate (OBC) has no enrollment process - it is created on-demand by the browser and is self-signed. This means the OBC can make no assertions about the user's identity. An OBC is tied to an origin - to a single website. Therefore, the potential for tracking is reduced but not eliminated entirely. Finally, because the OBC is created on-demand it is also selected on-demand, and silently. The user is never presented with a list of client certificates - if one exists, it is used, if not, it is created and then used.

OBC is not intended to change the way websites work now. With an ordinary client certificate, the certificate is almost always used to authenticate a user and automatically log them in. When using an OBC, they will not be logged in (because it makes no assertions about identity.) The user will still need to log in and have cookies set as the internet currently works. However, if a cookie was bound to an origin bound certificate (which is precisely what BrowserAuth.net proposes[43]), we have a solution that works with persistent cookies, and elegantly solves cookie theft. Any cookie thief would not be able to steal the origin-bound client certificate, and therefore would be unable to create a TLS session that the cookie would be valid in.

However, the original problems of server impersonation and first contact still exist. On first contact with an unknown server, a client could be attacked by a man in the middle, and never detect it. The attacker would generate their own origin-bound certificate, and the authentication cookie would be bound to *it*, with no way for the client to detect that the cookie wasn't bound to its. The client would save this cookie and if they tried to use it later with the server over an error-free-channel, it would fail! Secondly, an attacker able to impersonate a server to a user could perform the same attack to steal credentials - simply telling the user they've been logged out "for their safety" will trick most users into reauthenticating. Finally, OBC is another form of web tracking, albeit localized to a single site. Even when logged out, the OBC will be able to identify a client as a particular user if they've ever logged in while using that origin-bound certificate. It's also worthwhile to note that the HTTPOnly and Secure cookie attributes, when used with TLS, go a long way towards solving cookie theft currently. Although OBC and Channel-Bound Cookies is an elegant approach to solving the problem, it seems to be a tremendous amount of effort to defend against something we already have good defenses for now.

4.2.3 Using Binding Today

Channel Binding refers to a specific cryptographic concept, and none of the following recommendations are technically channel binding, but rather another form of binding. It would be possible for an organization to cryptographically bind a cookie to prevent cookie theft. Consider a cookie formed by encrypting the SessionID, browser, and operating system of the user, and authenticated with a message authentication code (MAC). When a server receives the cookie from a client, it will verify the MAC to ensure the ciphertext has not been tampered with, then decrypt it, find the user's SessionID, and the previously seen Browser and Operating System. If that browser and OS do not match the currently supplied values by the client (in the User-Agent Header) - something strange has happened. There's no reason a cookie should migrate from Internet Explorer to Firefox, or from a Linux machine to a Windows one (a browser version could conceivably upgrade, but that's handled easily enough.) If the server detects some anomaly like differing Operating Systems - they can invalidate the session and require the user to re-authenticate. It would likewise be possible to bind the SessionID to an IP address, a single TLS Session, or other information. Each binding point has drawbacks: using an IP address could break mobile clients or populations behind a proxy server, a TLS session breaks persistent cookies. Any decision should be weighted carefully, and may only be useful in limited deployment scenarios.

4.2.4 Secure Remote Password

Secure Remote Password is a protocol standardized by the IETF in RFC 2945[44], having been developed in the late

[40] http://www.browserauth.net/
[41] http://www.browserauth.net/tls-client-authentication
[42] http://www.browserauth.net/origin-bound-certificates
[43] http://www.browserauth.net/channel-bound-cookies
[44] https://tools.ietf.org/html/rfc2945
[45] http://srp.stanford.edu/

1990's. (Stanford has a good resource for SRP at [45].) SRP allows a client and server to prove to each other that each knows the password to a user account, without sending the password across the wire. As a nice property, the server stores the password hashed.

SRP can be layered inside of TLS, but there are no benefits from this. An attacker can still perform a man in the middle attack on the TLS connection, and observe the SRP protocol. They won't learn the password, because it's never sent, but they will observe any cookies that are set, and can impersonate the user to the server. But RFC 5054[46] defines a way to use SRP to establish a TLS connection. The TLS connection cannot be established unless both the client and server know the password. This means an attacker performing server impersonation or a man-in-the-middle attack will be unable to complete the handshake with the client, as the client will ultimately begin encrypting data that can only be decrypted if the other endpoint has the user's password.

SRP suffers from similar problems as client certificates when used on the web. A separate process must be created for enrollment, and a user cannot have a TLS session without being logged into the site. The alternative is worse: if TLS-SRP did not log the user into a site automatically, they would still be tracked but would also have two usernames and passwords. Because the username and password is used for TLS instead of the site, the browser UI is confusing.

TLS-SRP in the browser suffers from the problems listed, but other applications make use of TLS and a username/password. Any thick-client that runs on a computer, talks to a server over TLS, and authenticates using a username and password is a potential use of TLS-SRP. If an email client authenticated using TLS-SRP, it could never be attacked by a man in the middle, and an email provider would never need a Certificate Authority-signed certificate - they wouldn't need a certificate at all. Dozens of common applications could make use of TLS-SRP, from line-of-business apps to twitter clients. All of these applications gain the protection against man-in-the-middles because their authentication mechanism (SRP) is channel bound into the lower TLS protocol. (Another good resource for TLS-SRP is [47].)

SRP has been the subject of "is it or isn't it" patent debates. When SRP was being standardized the IETF investigated the relevant work and found three stakeholders: Stanford, Phoenix, and Lucent[48]. Stanford publishes a license stating "SRP is royalty-free worldwide for commercial and non-commercial use."[49]. Phoenix states it's patent on SPEKE *may* apply to SRP, but has committed to make licenses available on reasonable and non-discriminatory terms. Lucent has decided not to make any statement about applicability of the EKE patents to SRP. Most experts do not believe use of SRP should be a concern.

4.2.5 Other Identity Management Proposals

Other Identity Management solutions that are unconnected to TLS have been proposed and are being actively developed, including OpenID[50], BrowserID[51] (headed by Mozilla), WebID[52] (by the W3C), and Facebook Connect[53]. Additionally, OAuth[54], intended for application-to-application authentication.

4.3 More TLS Improvements

A full accounting of the features of TLS would encompass a paper twice this length, but the following are some recent or noteworthy additions to the TLS protocol.

4.3.1 Encrypted Client Certificates

Client Certificates are used to assert the identity of an individual to a server, and thus in practice often contain the individual's personal information - their full name and email and at other times can include they home address or other private information. Furthermore, client certificates are often used to connect to secure VPN services - often used when a user may not wish to alert the network operator of tunneling, or disclose their personal information freely. However, client certificate contents are sent in the clear as part of the TLS handshake. (Server

[46] https://tools.ietf.org/html/rfc5054
[47] http://trustedhttp.org/
[48] http://www.pdl.cmu.edu/mailinglists/ips/mail/msg09292.html
[49] http://srp.stanford.edu/license.txt
[50] http://openid.net/
[51] https://browserid.org/
[52] http://getwebid.org/
[53] https://developers.facebook.com/
[54] http://oauth.net/

certificates are likewise sent in the clear; however, these do not typically contain personally identifiable information.)

A current proposal before the IETF[55] will have a client send an "encrypted_client_certificates" extension in it's ClientHello, which will be echoed by the server in the ServerHello, if supported. If the client and server agree on using Encrypted Client Certificates, the Certificate and CertificateVerify messages are moved from before to after the ChangeCipherSpec message. This will cause them to be encrypted with the encryption key and algorithm negotiated during the handshake, effectively hiding them from a network operator. It is important to note that if the client software is not configured correctly, it may be possible to perform a downgrade attack by tricking the client into believing encrypted client certificates are not supported.

4.3.2 Datagram TLS

DTLS, or Datagram TLS, provides a version of TLS that will operate over datagram protocols like UDP, defined most recently in RFC 6347[56]. DTLS is currently at version 1.2 in parity with TLS - there is no DTLS 1.1. TLS cannot handle records that arrive out of order or are skipped, because it cannot decrypt arbitrary records - it requires all the records prior to the target to decrypt. Similarly, the TLS handshake will break if the handshake messages are lost. DTLS solves the second problem by requiring handshake messages be in a specific order, saving latter ones for processing until prior ones are received, and retransmitting after a timer expires. To provide the ability to decrypt an arbitrary record, DTLS must ban stream ciphers (because the stream cipher context would get out of sync in a reordered or lost packet), and add an unsigned integer containing the sequence number to the record - without it the message authentication code does not have all the information needed to verify the record. DTLS is obviously a useful protocol whenever a trustworthy encryption protocol is desired, but a datagram protocol (e.g. UDP) must be used for other engineering reasons.

4.3.3 Minor TLS Additions

Several other TLS additions have been standardized recently (one while I was writing this very section of the paper.) RFC 5705[57] defines a method to use the keying material produced by the TLS handshake in channel binding or as a seed to a random number generator. One use is if the TLS handshake is trusted to have used a good psuedo-random number generator and its entropy is not in question, an implementor may wish to use it to bootstrap their random number generation; the other use being similar to the bind types defined for channel binding to TLS. A heartbeat extension was added to TLS and DTLS in RFC 6520[58]. Previously, if a client wished to check with the peer to determine if it is still alive (most commonly in DTLS, but possibly also in TLS) the only mechanism to do so would be with an application-level heartbeat or no-op message, or with a costly TLS renegotiation. The HeartbeatRequest and Response messages indicate the peer is still present and holding the session open. RFC 4279[59] defines TLS authentication performed via Pre-Shared Keys. TLS-PSK can be advantageous to avoid public key operations on embedded devices, among other scenarios.

There are three additional items to note around certificate transportation, including two older standards, to provide a full accounting of Key Exchange Algorithms. A current draft[60] will allow a server to send the raw public key of its certificate, instead of the entire certificate container and chain. This can be advantageous in embedded scenarios or other situations where the client is already in possession of an authenticated public key for the server, for example if it was obtained via DANE or another mechanism. Finally, it is worth noting an older addition to TLS that is not widely used: OpenPGP. Although not widely used or implemented, it is possible[61] to send an OpenPGP key in place of an x.509 certificate. Finally, a very old addition to TLS is RFC 2712[62] which specifies how Kerberos can be used within TLS.

[55] http://tools.ietf.org/html/draft-agl-tls-encryptedclientcerts
[56] http://tools.ietf.org/html/rfc6347
[57] http://tools.ietf.org/html/rfc5705
[58] http://tools.ietf.org/html/rfc6520
[59] http://tools.ietf.org/html/rfc4279
[60] http://tools.ietf.org/html/draft-wouters-tls-oob-pubkey
[61] http://tools.ietf.org/html/rfc6091
[62] http://tools.ietf.org/html/rfc2712

5 Public Key Infrastructure

Public Key Infrastructure is a tremendously large topic with more corner cases than most individuals are aware of. The most commonly thought of implementation is the CA infrastructure and public websites. But corporations have their own Certificate Authorities with their own infrastructure and revocation mechanisms. Revocation occurs not just for server certificates, meaning the client must verify - but for client certificates too, meaning the *server* must perform revocation checking. The Federal Bridge CA is used provide a single Public Key Infrastructure amongst several US government departments and has been described as the 'Most Complicated PKI Infrastructure on the Planet', containing several cross-signed roots. But it is almost never thought of by individuals when they first become interested in PKI and begin exploring.

The following section is not designed to perform even a cursory explanation of the intricacies of PKI as it exists today, but rather focus on the most common case: validation of public sites in a web browser, for average users. In the past few years, spurred on by several repeated high-profile incidents at Certificate Authorities like Comodo, Diginotar, and Trustwave, there has been a growing wave of discontentment with Certificate Authorities, and there have been many proposals to change the system, or eliminate them entirely. These proposals, in addition to revocation, are covered with an eye towards drawing out underlying commonalities and properties, ultimately culminating in a list of concepts that can be used to evaluate any proposal, or perhaps design yet another.

5.1 Revocation

5.1.1 Certificate Revocation Lists

A Certificate Revocation List (CRL) is a list of Serial Numbers of certificates a Certificate Authority has revoked, along with issue and next-issue dates, signed by the CA. CRLs are updated every couple days, and are cached on the client for up to a week. CRLs often must be explicitly installed on clients - while it is possible to put a CRL location into the certificate itself via an extension, this is not always done. Besides problems with locating them and timeliness - CRLs can grow to be very, very large - which is a concern particularly with mobile clients. Techniques to segment them by revocation reason or perform delta CRLs have been proposed, but not widely adopted.

5.1.2 Online Certificate Status Protocol

The Online Certificate Status Protocol (OCSP) is a mechanism that's designed to be more timely. When a client receives a certificate from a server they query an OCSP responder, which is almost always run the issuing CA, to check the revocation status of the certificate. On one end of the spectrum, this could provide much better security, as a CA could potentially respond with an affirmative "This certificate is still valid as of now, and I recognize it" (a whitelist approach), but on the other end of the spectrum - OCSP responders are often fed by CRLs and thus inherit the same timeliness problem.

A major problem with OCSP responders is the additional latency added to a page load - for each new certificate seen (several on a page with third party resources) a browser must make an additional request after the TLS handshake prior to the request. Additionally, an OCSP responder is another single point of failure for a site, one outside of their control. If the CA's OCSP responder goes down, so to does the site. Because of this reason, no browser will perform a hard-fail on the lack of an OCSP response. Dubbed 'soft-fail OCSP', this provides no security, because an attacker performing a man in the middle attack can simply block OCSP requests or responses. There are other issues: OCSP status codes are ambiguous at times, it too is driven off serial numbers instead of certificates, and it also leaks the client's browsing habits to the operator of the OCSP responder.

5.1.3 OCSP Stapling

OCSP Stapling attempts to address the two largest problems with OCSP. Instead of a client obtaining an OCSP response, the server will obtain the OCSP response in advance and provide it to the client. Because the response is signed by the CA, it is fine for the server to provide it to us, and this eliminates the single point of failure, the additional round-trip required, and also the privacy leakage. OCSP Stapling does have a few practical problem, perhaps most notably that it's not available to most people because it's not widely supported by CAs or software. Additionally, the way OCSP is standardized currently - only a single OCSP response can be stapled. But most certificates have a chain of three: the root, an intermediate, and the leaf. The root is hardcoded as trusted, and thus would really need a patch to properly distrust. But an intermediate cert can be revoked, and thus should have a revocation check. Practically, intermediate certs are rarely revoked, which provides some leeway.

5.1.4 Revocation Proposals

During the writing of this paper, Google has announced[63] that Chrome will be moving off OCSP queries and will instead push CRLs down to the browser using a system similar to its autoupdate mechanism - although a browser restart won't be required. This technique is very similar to a software patch, which is what has ultimately been done in nearly all major certificate compromises: Code-Signing Certificates for Microsoft, misissued certificates from Comodo, the hundreds of certificates issued from Diginotar, and the intermediate certificate issued from Trustwave. When revocation was needed most: all browsers had to push a software patch to their users - Certificate Authority-based revocation wasn't enough.

Another proposed solution to revocation is to do away with it altogether and use short-lived certificates. If a certificate is only valid for a week, revocation will be achieved through expiration. Although a certificate can be resigned if no problems are present, this ultimately requires close coordination between Certificate Authorities and customers, and automatic online signing.

5.2 Certification Authority Authorization

Certificate Authority Authorization (CAA) is an IETF draft[64] likely to be standardized soon that is designed to be a mitigating control for Certificate Authorities so they are not tricked into issuing certificates for domains to fraudulent applicants. If a server operator wishes to restrict what Certificate Authorities are able to issue certificates for their domains, they can enter a CAA DNS record that states the specific CAs that are able to issue for that domain. This value is not intended to be consumed by clients (and actually may lead them astray), but rather by the Certificate Authorities themselves.

A Certificate Authority, when receiving a request to sign a certificate for example.com, will check the CAA DNS record for example.com and ensure that if it exists, it contains their name. If it does not, they will not issue the certificate. This process is obviously entirely opt-in on the part of the CA - a malicious or negligent CA could never perform the check or ignore it. Proposed by a CA itself, it's designed to be a mitigating control for Certificate Authorities who want to voluntarily improve their security.

CAA also includes a reporting mechanism, so a domain owner who wishes to receive reports from complying CAs can put in their e-mail or web service address. A CA that abides by CAA and receives a fraudulent certificate request can then send the incident report to the domain owner.

5.3 Extensions for Server Operators

A couple of proposals have been made about certificate extensions intended to benefit site operators. One commonly known as Seen Chains[65] allows browsers to report to server operators past certificate chains the client has seen for this host. While some operators would find this information valuable, most would probably find it overwhelming and furthermore all the exact same. Although Seen Chains is unlikely to be adopted or revised further, the mechanism of reporting anomalies is a good feature, and was what led to the discovery of the Diginotar attack (and that was reported manually - automatic reporting is even better.)

Another certificate extension that will soon be drafted according to its author is a flag that indicates the certificate should *always* have an OCSP staple attached to it[66]. As the ecosystem stands now, this extension does not add much to security - if an attacker breaks into a CA, they'll opt to not have this flag set. But in the future, if a CA chose to *only* issue certificates that had this extension set, that CA's certificates would be more secure. The attacker would target a new CA of course, and eventually if *all* CAs moved to this flag, we would have an all-in approach to fixing revocation and the security gains of a working system.

5.4 Security Policy Learning

Peter Gutmann has given a presentation entitled "PKI as Part of an Integrated Risk Management Strategy for Web Security"[67], that despite its dry name contains an extraordinary number of interesting concepts. One observation is that the browsers have produced a User Inter-

[63] http://www.imperialviolet.org/2012/02/05/crlsets.html
[64] http://tools.ietf.org/html/draft-ietf-pkix-caa
[65] http://tools.ietf.org/id/draft-weimer-tls-previous-certificate
[66] http://www.ietf.org/mail-archive/web/pkix/current/msg30258.html
[67] http://www.cs.auckland.ac.nz/~pgut001/pubs/pki_risk.pdf

face that shows almost the same 'trustworthyness rating' to sites with *and without* a SSL Certificate, but very scary looking warnings for sites with a self-signed certificate. Similarly, there is only two forms of trust: Not Trusted, and Trusted (the UI indicators for Extended Validated certificates notwithstanding.) But a more nuanced approach could indicate "probably safe" and "probably unsafe".

When the extremes are filled in and a sliding scale of trustworthiness is present, there are more indicators available that can contribute to that scale than merely the presence of a certificate. A browser may review the continuity of the Certificate, Certificate Authority, Geolocation of host, IP address, and AS. Similarly, there's a wealth of time information: registration dates of the AS, DNS name, certificate, the TTL of the DNS. Geographic information where China or Eastern European-based services are less safe, and the browser could see if the IP Address is a residential connection. Perhaps most powerfully: how often do I visit this site? Have I routinely visited it and provided my credentials, or is this the first time? With this host of information, a browser can potentially warn people away from spear phishing attempts that no safebrowsing list has ever seen before.

But browsers aren't the only ones who can take this approach - it's entirely possible for a Certificate Authority to as well. Is this an American site, hosted in America, with a request for a certificate coming from Iran? Where does the request originate? Is a current Extended Validation certificate being replaced by a Domain Validated certificate? Are certificates nowhere near expiry being replaced? Is the certificate authority changing? Is any of this occurring for an Alexa Top Million website? Top Hundred? Any of these checks would have stopped the bogus certificates issued by Comodo in early 2011.

5.5 CA/Browser Forum

The Certificate Authority / Browser Forum[68] is a "voluntary organization of leading certification authorities (CAs) and vendors of Internet browser software and other applications". The most notable part about the CAB Forum is that being composed of CAs, any requirements or restrictions adopted by it must be amenable to the CAs and thus makes it much more likely to be adopted. In December 2011 new Baseline Requirements were adopted to govern the issuance of certificates and requirements for CAs. These guidelines are not revolutionary, as can be imagined, but do provide several useful guarantees from CAs. Certificate Issuing policies must be public, as must security audits, which must be conducted annually. Some form of revocation, either CRLs or OCSP, must be provided, and must be able to be done in under 24 hours in the case of a high-priority issue. Revocation also must be available to the certificate owner. Finally, publicly available problem reporting must be available. These requirements take effect July 1, 2012. Other clauses, such as forbidding issuance of certificates to private addresses, such as 10.0.0.0/8, take effect much later.

The CAB Forum is a private group, and the only source of insight is discussions that occurs on the mozilla.dev.security.policy forum[69]. A certain degree of discontent with the CAB Forum has arisen due to it's closed nature. As this paper was being authored, an Organization Reform Working Group has been announced. For the most up-to-date information, refer to the organization's website[68].

5.6 Network Perspectives Based Trust Decisions

A man in the middle attack is isolated to the networks downstream of the compromised router. If an attacked user is able to gain the perspective of another network unaffected, they would be able to see the anomaly. That is the basic premise behind several Network Perspectives-based revamps of the Certificate Authority system.

5.6.1 Perspectives

Perspectives[70], a project at Carnegie Mellon, is the oldest project in this space. The proof of concept was a Firefox Add-On; when a user initiated a connection to a secure site, the add-on would contact servers called notaries and find what certificate the notaries saw for the site; and how long they had seen it. In a best-case scenario, they will have all seen the same certificate as the user, and for a long time, and the certificate would be considered trusted.

However, if a site rotates SSL certificates, or uses multiple SSL certificates presented from different load balanced

[68] http://cabforum.org/
[69] https://groups.google.com/group/mozilla.dev.security.policy/
[70] http://perspectives-project.org/

servers, Perspectives will report an anomaly that may be indistinguishable from an attack. This is commonly referred to as the 'Citibank problem', as they were the highest profile site that did this. Each server had its own SSL certificate, different Perspectives notaries would contact different servers, and each notary would report different certificates for the same domain. Additionally, because notaries poll the website periodically, there is a problem when the server has updated its certificate, but the notary hasn't seen it yet - this lag time looks very similar to an attack from the user's point of view. Finally, privacy for the user is not preserved as notaries are informed whenever a user visits a site.

5.6.2 Convergence

Convergence[71] is a project from Moxie Marlinspike designed to fix the deficiencies in Perspectives and provide more agility. Convergence is also demonstrated in a Firefox Add-On, and replaces all SSL verification that occurs within the browser. It fixes the privacy loss by caching certificates locally if they've already been trusted, and by creating something called notary bouncing. With notary bouncing, one notary will proxy a request to a second; the first notary doesn't know what site is being visited, and the second notary doesn't know who is visiting the site. A notary performs online validation instead of polling if it doesn't recognize the certificate seen, so notary lag is eliminated.

The most extensible part of Convergence is that a notary is able to validate a certificate in any fashion it wishes. The default operation is to perform the same network perspectives approach as Perspectives, but it is also possible to verify certificates from the Google Certificate Catalog. Although the code is not written, it is also possible to run notaries that verify certificates through a set of trusted Certificate Authorities, through DNSSEC, the EFF SSL Observatory, or any other practice desired. The ability to change which notaries to trust, and therefore which validation schemes, is dubbed Trust Agility.

Two issues that plague both Perspectives and Convergence are internal servers and captive portals. If a notary can't reach a server - because it is inside an organization inaccessible to the internet - it's unable to validate it. And captive portals - or the clickthrough agreements in hotels and airports - prevent any request from escaping until the user has paid and/or agreed to the terms. Convergence is unable to validate the payment or agreement gateway because the client cannot contact the notaries.

Another potential problem with both Perspectives and Convergence is scale. Google Chrome has stated[72] this is a very significant reason preventing them from implementing it. Similar to OCSP Hard Fail, the notaries become points of failure, not just for a single site, but for the entire browser. Hundreds of millions of users will hit the default set of notaries and never change off them - those notaries must perform quickly with 100% uptime. In the case of commercial browser entities like Internet Explorer and Chrome, the respective companies will likely be required to run the notaries to ensure their browser is not rendered unusable - eliminating the privacy gains.

5.6.3 CrossBear & MECAI

Crossbear[73] is another Firefox Add-On that makes Network Perspectives based trust decisions. The notable difference; however, is that Crossbear is designed to report any anomalies and have more users attempt to validate the finding and determine what network the attack is isolated to. A similar but less aggressive feature is available in the latest version of HTTPS Everywhere[74] a Firefox Add-On that automatically takes you to the SSL version of a site if it is available. If enabled, HTTPS Everywhere will report certificates to the SSL Observatory[75], which can detect man in the middle attacks.

MECAI, or Mutually-Endorsing Certificate Authority Infrastructure[76], is a proposal that borrows concepts from several others and aims to remove the scale problem raised with Convergence and Perspectives. Similar to Convergence, a client has some choice in whom to trust - when a client contacts a server, they specify which authorities they would like to vouch for the server. The server will ideally have these vouchers ready, signed by the different authorities, with a relatively short expiry time (on the order of a day), similar to OCSP stapling. The vouching authority is stating that it has seen this certificate, and that according to an OCSP responder, the

[71] http://convergence.io/
[72] http://www.imperialviolet.org/2011/09/07/convergence.html
[73] https://twitter.com/#!/crossbearteam
[74] https://www.eff.org/https-everywhere
[75] https://www.eff.org/observatory
[76] https://kuix.de/mecai/

certificate is not revoked. The final piece of MECAI is that the vouching authorities are actually other Certificate Authorities - trying to eliminate the scaling problem under the assumption that Certificate Authorities should be able to scale.

5.7 Public Accountability Based Trust Decisions

Currently, a Certificate Authority can sign any certificate they wish, and it will be trusted even though the cert has never been seen before. The reputation of the Certificate Authority is relied on. The opposite of this approach would be to trust a certificate on its own reputation - to require a certificate be public before it is trusted. An advantage inherent in this approach is domain owners can look at all the trusted certificates on the internet and see if there are certificates issued for the owner's domain they are unaware of.

5.7.1 Sovereign Keys

Sovereign Keys[77] is a proposal by the EFF that boils down to creating a key, dubbed a Sovereign Key, that will be the authority for the domain going forward. A Sovereign Key is posted to an append-only Timeline server, and all certificates presented by a domain will be signed by that domain's Sovereign Key.

Soverign Keys is actually considerably more complicated: the Timeline servers must be verifiably append-only, and replicated to mirrors. There are multiple timeline servers, and they include data from each other by reference. Timelines have the set of Certificate Authorities they trust at the current time embedded in them (because a Sovereign Key is only valid if it is CA-signed, to bootstrap validation through a trusted third party.) The mirrors and clients may detect a timeline server has misbehaved, and there is a protocol to automatically distrust that timeline. A Sovereign Key owner is able to specify entities that are able to revoke and recreate a new Sovereign Key for the domain. The full spec is available on the EFF's git repository[78].

Besides the considerable complexity, Sovereign Keys may lack the ability to distrust core Timeline Servers depending on the implementation. Additionally, Sovereign Keys introduces significant side channels when contacting mirrors to find Sovereign Keys.

5.7.2 CA Transparency and Auditability

Certificate Authority Transparency and Auditability (CT)[79,80,81,82] is a proposal originating from Google that relies on individual certificates being registered in public logs. A certificate is not trusted until it is present in the log, and a server will present proof of registration along with the certificate, similar to OCSP stapling. The logs are verifiably append-only using a cryptographic primitive called a Merkle Tree, and this property is routinely confirmed. Domain owners watch the logs and ensure that any certificate that appears for their domain is controlled by them.

It's possible for this to be implemented so domain owners have to perform no additional work, the Certificate Authority they register with will take the effort of registering the certificate in the log for them. Additionally, there is no side channel and no privacy leakage and it's possible to handle private subdomains - where an organization doesn't wish to publish internal domains within an organization. Similar to Sovereign Keys, there is the concern that a log is unable to be distrusted.

Revocation is not addressed yet, in the initial paper a system similar to DNSSEC proof of nonexistence is briefly described. A server operator would provide, or a client would query and receive, a list of revoked certificates, sorted in an order. The client would look in the list for the certificate they are concerned with, and if certificate is not present, it would not be considered revoked.

5.8 Simple & Unusual Approaches

5.8.1 Certificate Patrol

Certificate Patrol[83] may be the oldest Firefox Add-On in the group, and is certainly the simplest. It remembers what certificates have been seen for a domain, and if the

[77] https://www.eff.org/sovereign-keys
[78] https://git.eff.org/?p=sovereign-keys.git;a=blob;f=sovereign-key-design.txt;hb=master
[79] http://www.links.org/files/CertificateAuthorityTransparencyandAuditability.pdf
[80] http://www.links.org/?p=1212
[81] http://www.imperialviolet.org/2011/11/29/certtransparency.html
[82] http://www.ietf.org/mail-archive/web/pkix/current/msg30146.html
[83] http://patrol.psyced.org/

certificate changes, presents that change to the user for review. Of course, a user who is unfamiliar of SSL would have no idea what to do with this knowledge, but an informed user, like those reading this paper, can review the change and see if it makes sense to them.

5.8.2 Certificate Authority Penalties

Mozilla has proposed[84] adding support in the NSS library to distrust Certificate Authorities based on time. If a Certificate Authority misbehaves or does not follow proper procedure, all certificates issued by them after a certain date can be distrusted, without breaking existing certificates. Similarly, Sub-CAs or Intermediate Certs could be limited.

5.8.3 MonkeySphere

MonkeySphere[85] is almost the oddest proposal covered, beaten only by the next section. Monkeysphere aims to bring the PGP Web of Trust to SSL Certificates. When a new SSL certificate is received, Monkeysphere will attempt to find a web of trust path from the user's PGP key, to a PGP key with a UID matching the site, and a key matching the certificate's. As currently implemented, Monkeysphere only performs this trust check for a certificate that is not trusted under the current hardcoded Certificate Authorities.

5.8.4 YURLs

Finally, YURLs[86] invert the trust decision entirely. A concept known as Zooko's Triangle[87] (illustrated in Figure 6) was defined in 2003; summarized it explains that there are three properties of names of participants in a network protocol, and it is not possible to satisfy all three properties. A name is human-meaningful if it is **memorable**, such as a domain name. **Global** or decentralized means there is no central authority defining the names. Finally, **secure** refers not to the transport mechanism, but rather the assurance that the user is communicating to the endpoint they believe they are communicating with.

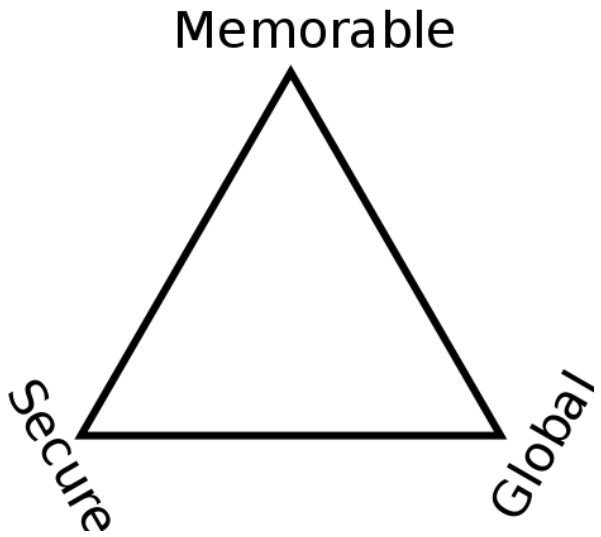

Figure 6: Zooko's Triangle

HTTPS URLs aim for Secure and Memorable. Domain names are obviously human meaningful, and we achieve Secure through the use of a central authority (Certificate Authorities) - ideally the central authorities will not issue a certificate for a domain to anyone but the domain owner. YURLs, as well as Tor Hidden Services, achieve secure by embedding the key, or fingerprint of it, in the URL itself. A user can never be tricked into communicating with a different server if they already know the key of the server they want to communicate with. If the server has the corresponding private key, it's the correct one, if not, it can't be. Embedding a key in the URL immediately removes human meaningful; YURLs take the form `httpsy://*cl7h3f7jwyj3fvmw7jpnjfvf2xlcmayi@yurl.net/`. Without human meaningful, they aim for decentralization - the domain name at the end of the YURL is not a definitive place to look, but rather a hint for the *first* place to look. It is possible to be redirected to a new server; a chain of redirections may occur until the server with the corresponding private key is found.

5.9 Concepts of Certificate Authority Enhancements and Replacements

When evaluating any proposal to change or enhance Certificate Authorities or revocation, a number of design decisions and trade-offs should be considered. Bootstrap-

[84] http://groups.google.com/group/mozilla.dev.security.policy/browse_thread/thread/dc63e870965f38fb/eb7edebc5dc4512d
[85] http://web.monkeysphere.info/
[86] http://www.waterken.com/dev/YURL/
[87] https://en.wikipedia.org/wiki/Zooko's_triangle

ping trust is difficult, often impossible, as illustrated with Public Key Pinning and Strict Transport Security. Security Policy is ultimately dictated by the site, and mechanisms that allow a site to define what is anomalous provide a great deal of power - seen again in Public Key Pinning, Strict Transport Security, and also DNSSEC-based assertions like DANE and CAA. When a user detects an anomaly, that can be reported to the site to great benefit, like in Content Security Policy and CAA. And also, if security policy is not explicitly dictated, it can often be inferred through rational decisions about what may be strange or what is reassuring.

From an engineering standpoint - anything that requires extra round trips is bad. To a lesser extent, anything that increases packet size such that window sizes are overflowed (and thus require a roundtrip at the TCP layer) is also bad - although this is very low level and often ignored by all but the largest sites. It's important to remember that any signed date requires the client to have a correct clock. And that users will never change the defaults, so those defaults may become points of failure. From an implementation standpoint, who has to change: the browsers, servers, or CAs; and how difficult is that change?

The ability to change who is trusted is desirable, both for power users, but also so browsers can revoke trust without disabling large swaths of the internet for their users. And privacy matters - who receives browsing history in form of validity checks? Finally, a Network Perspectives approach doesn't say what's valid, only what *is*; and a public accountability approach doesn't state what's correct, only what's public.

6 About iSEC Partners

iSEC Partners, an NCC Group company, is a full-service security consulting firm that provides penetration testing, secure systems development, security education and software design verification. iSEC Partners' security assessments leverage our extensive knowledge of current security vulnerabilities, penetration techniques and software development best practices to enable customers to secure their applications against ever-present threats on the Internet.