



Protecting Your Code Updates: How to Defend Against SSL Spoofing Attacks

By Larry Seltzer

Security Analyst and Writer

Table of contents

- Executive summary**1
- Introduction**2
- Code Signing**2
- Software Updates** 5
 - The Security Need.....5
 - Static Updates.....5
 - Automatic Updates..... 6
- The Security Threat**..... 6
 - Null Characters in a Domain Name7
 - The Auto-Update Attack Scenario..... 8
- Whitelisting** 8
 - Advantages Relative to Hashes 9
 - Important Implementations..... 9
 - Development.....10
- Conclusion** 11
- Appendix A - Public Key Encryption and Digital Certificates** 12
 - Certificate Chains of Authority 12
- Appendix B - Null Characters in Domain Names** 13

Executive summary

Security experts generally recommend applying security updates to software as soon as possible, as the overwhelming majority of attacks against vulnerabilities are against those that have already been addressed with patches from the software developer. But, published scenarios enable attackers to compromise the safety of these updates.

Some code distribution methods rely only on Secure Sockets Layer (SSL) offerings to protect the integrity of the update process, but authentication through conventional SSL can be weak and subject to man-in-the-middle attacks. Both static code distribution sites and built-in automatic update mechanisms are often vulnerable to these attacks.

The solution to this problem is code signing, a mature technology built into Windows and many other systems for years. Code signing allows users to ensure that a program was created by a named and authenticated organization. Programmatic updates can build this mechanism into their own algorithms to ensure that they are not being fed rogue updates.

Conventional SSL
can be subject to
man-in-the-middle
attacks.

Introduction

It is an axiom of computer security that users have to trust some aspects of the system. Yet developers of many essential software applications fail to take simple measures to validate that trust.

Code signing certificates are the standard for providing proof of origin for an executable software program. It is common for malicious software to masquerade as legitimate, and code signing has long been a way to protect against this threat. Many of the most sophisticated software companies rely on code signing, and for good reason. The liability and embarrassment that result from a compromise of the update process are devastating for a software vendor.

New research describes attacks against SSL that create opportunities to compromise the update processes of unsigned software. This paper will show how code signing works, how attacks can be mounted against unsigned software, (including auto-update software), and how real-world signing systems protect software vendors, enterprises, and end users.

Code Signing

Code signing is a process that uses Public Key Infrastructure (PKI) technology to create a digital signature based on a private key and the contents of a program file, and packages that signature with the file. Users combine the file and the associated public key to verify the identity of the file signer and the integrity of the file.

For some basics of public key encryption, and how it relates to certificates, see Appendix A: Public Key Encryption and Digital Certificates.

Code signing starts with a digital certificate. Users can create their own with many of the available free tools, or can purchase one from a trusted certificate authority (CA). The user provides a name for the entity, typically a company, and other identifying information. The CA provides a certificate to the user with public and private keys. The certificate is also signed by the certificate authority.

It is essential that users keep private keys secure and confidential, restricting access only to those who absolutely need them. Anyone who has access to the private key can create software that will appear to be signed by the owner of the certificate.

A code signing certificate, also known as a Software Publisher Certificate, has special fields and options particular to code signing. An SSL certificate, for example, cannot be used for code signing.

Certificate authorities offer different types of code signing certificates for different code types. VeriSign, for example, offers certificates for Microsoft® Authenticode®, for Java®, for Adobe® AIR™, for Microsoft® Office, and others.¹

A reputable CA that sells a code signing certificate will not just take the applicant's word for their identity. It will perform checks on the company names, phone numbers, and other information that is required to prove identity—a process that can take up to several days.²

With a Software Publisher Certificate, a programmer can digitally sign files distributed with the software. A general description of the process follows. For more detailed information on how developers actually work with the code signing process, see the Development section below.

Digital signing of software begins with the creation of a cryptographic “hash” of the file being signed. A hashing function is a mathematical process that creates a hash value, often called a digest, which has a 1:1 correspondence with the original data. This digest provides no hints of how to recreate the original data, and even a small change in the original data will result in a significant change in the hash value. The SHA-1 hash function, a very popular one, produces a 160-bit digest. The SHA (Secure Hash Algorithm) functions are governed by the National Institute of Standards and Technology (NIST).³

**A reputable CA
will not just
take the
applicant's word.**

The code signing program then uses the private key to sign the digest, meaning it generates a signature in the form of a string of bits. Good digital signature algorithms allow a user with the public key to verify the creator of the signature, but not allow someone who does not have the private key to generate a signature.

The next step in the code signing process is to copy the certificates from the Software Publisher Certificate into a new PKCS #7 (Public Key Cryptography Standards) signed data object⁴. In addition to the certificates themselves, and the signed digest, the object contains the serial numbers and information on issuers of the certificates used to create the signature. This object is embedded into the signed file.

A good digital signature will include dates for the signing and the expiration of the signature, although technically this is not required by the standard. The shorter the life span of a certificate, the more frequently the identity of the signer is verified by the certificate authority.

A certificate authority can revoke a certificate for a number of reasons. For example, the user may violate legal terms, such as by signing a malicious program, or the user may report to the certificate authority that the private key has become compromised. Client systems can check to see if a seemingly valid certificate has been revoked.

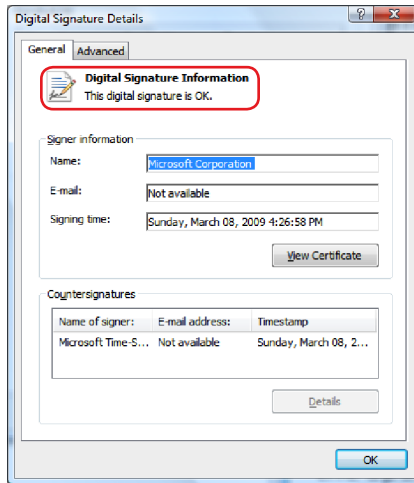
¹ VeriSign - Code Signing Certificates (<http://www.verisign.com/code-signing/content-signing-certificates/index.html>)

² VeriSign - Code Signing Certificate FAQ (<http://www.verisign.com/code-signing/information-center/certificates-faq/index.html#a4>)

³ NIST – Cryptographic Toolkit (<http://csrc.nist.gov/groups/ST/toolkit/index.html>)

⁴ RFC 2315 - PKCS #7: Cryptographic Message Syntax (<http://tools.ietf.org/html/rfc2315>)

Windows shows a signed file's certificate as being valid



The image is an example of how Windows provides a way to see details of the code signing certificate through the File Properties dialog box. The user can view the actual certificate details, including the issuer and when the certificate expires. Windows takes the signature, and certificates that are attached to the program, recalculates the hash, uses the certificate to obtain the public key of the publisher, and uses that public key to verify several characteristics of the file:

- The certificate is in valid form
- The digest in the certificate matches the one calculated by Windows

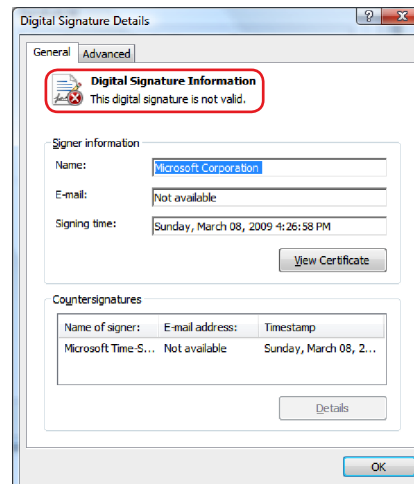
- The signature is valid, meaning that it was created with the private key associated with the public key
- The date of the signature is within the valid dates of the certificate
- The certificate has not been revoked by the certificate authority

In many cases, this information is checked automatically. Apple Software Update, for example, downloads updates and checks the signatures before installing them. Such automated update systems don't usually provide any positive feedback for users on their connections to servers. If the connection is successful, users rarely see anything more informative than "Connected to update server."

In the static case, checking the signature causes the software (Windows in the image) to recalculate the hash for the file to check it against the stored one. When even a single byte is changed in the file from the above "valid certificate" example, Windows informs the user that the signature is not valid.

It is worth noting that code signing signatures don't check for safety or quality. Code signing confirms who published the file, and that it has not been modified since it was signed. It is important not to trust software merely because it is signed, but to examine the signer and evaluate their trustworthiness.

After the file has been tampered with, the signature is no longer valid



Software Updates

Long ago, software updates came on physical media sent out by the vendor. There wasn't much reason to worry about the security of the updates. Today, updates come from the vendors, or even third parties, over the Internet. Users need a way to confirm the authenticity of the files they are downloading.

► The Security Need

The ease and low cost of distributing software through the Internet has led software vendors to distribute their updates on a more frequent schedule. These updates may implement new features, or deliver bug fixes, but the most common reason is to address security flaws. The imperative for users to install the updates, and install them quickly, is pressing and growing.

Some enterprises can take the time and money to test every update for problems that they might cause, but most users have to trust the developer, which means trusting the update. But is the update actually what it seems to be?

► Static Updates

The history of fake updates to Windows is long. It is one of the earliest e-mail scams for pushing malware. Sophisticated users can spot them right away, and they know that Microsoft never sends out security updates as attachments to e-mails. Microsoft also digitally signs all of their static update files, and they are distributed only through Microsoft's servers.⁵ Other major companies, including Apple, also digitally sign all of their updates, even though they haven't been a prominent target of malicious attacks.

In the open source software world, digital signatures are not popular and the best users can hope for is an MD5 hash of the files to prove the integrity, if not the authenticity, of the file - MD5, or Message Digest algorithm 5, is a hash function like SHA-1, but older and less reliable. There have been many compromises of such open source download sites. In 2007, the download servers for the famous blogging platform, WordPress, were compromised and an attacker replaced the version 2.1.1 source file with a copy that allowed unauthorized remote PHP access to the server.⁶

MD5 hashes were available, inconveniently, from WordPress for their files, but this was of little value in ensuring the integrity of the file: If the attacker was able to put a new WordPress file in the download directory, he could just as easily calculate a new hash and put it in the same directory.

Most users
have to trust
the developer,
which means trusting
the update.

⁵ Internet Storm Center – “Fake Microsoft Update Email”, October 10, 2008 (<http://isc.sans.org/diary.html?storyid=5159>)

⁶ WordPress Blog - “WordPress 2.1.1 dangerous, Upgrade to 2.1.2”, March 2, 2007 (<http://wordpress.org/development/2007/03/upgrade-212/>)

Virtually all mobile (phone) computing platforms use code signing to control the software allowed on the network. Developers typically must join a program through which their software is tested for adverse effects and then signed with a key owned by the mobile network operator. The operators are very sensitive about letting malicious software on to their networks. Handset manufacturers, like Nokia, and operating system vendors, like Google and Symbian, also have their own code signing systems.

For years, security experts have assumed that the malware plague would soon extend to mobile networks. However, while there have been malware incidents, and users on some networks “jailbreak” their phones in order to run unauthorized software, the problem has remained largely theoretical because of code signing.

One last important example: In 64-bit versions of Windows, all kernel mode software, including all device drivers, have to be signed by a code signing certificate issued by a trusted certificate authority. Kernel mode software, as opposed to user mode software, has unrestricted access to the system. Applications that run in user mode, typically end-user applications like Microsoft® Word, cannot change basic system settings or install other software.

This requirement goes a long way towards blocking rogue software. Microsoft does this for the same reason that mobile networks do it: software in the kernel has to be trusted, and it is more trustworthy if it is signed by an entity whose identity has been verified as a legitimate firm.

► Automatic Updates

Because many users don’t install important updates even when told to do so, automatic update systems have become a popular method for developers to keep their users’ installations up to date. The most famous of these is Microsoft Windows Update, but Apple, Sun, Adobe, and many others also have facilities in place to download updates automatically.

It is easy to assume that, because a product is installed and configured to receive automatic updates, the updates will come from the software publisher. In the case of a malware attack, this may not be the case and must be protected against.

► The Security Threat

The July 2009 Black Hat show in Las Vegas⁷ featured researcher Moxie Marlinspike⁸ enhancing his already famous “SSLStrip” and related attacks⁹. Marlinspike combined a number of discrete problems, not all related to SSL, to create a credible scenario in which users attempting to work with secure web sites were instead sent to malicious fake sites.

⁷ Black Hat USA 2009, Welcome page - <http://www.blackhat.com/html/bh-usa-09/bh-us-09-main.html>

⁸ Moxie Marlinspike, home page - <http://www.thoughtcrime.org/>

⁹ SSLStrip, description and code download. <http://www.thoughtcrime.org/software/sslstrip/index.html>

Marlinspike also demonstrated how the technique could be extended to automatic update systems to deliver rogue software updates to client systems. Some of these systems were protected only by SSL communications. Code signing could provide much stronger protection.

One of the core problems described by Marlinspike is the ability to embed null characters in the common name field of a certificate, designating a domain name. This can be used to trick software, including web browsers and auto-updating software, into recognizing a domain name different from the complete field name. The result is that software, and users, are shown the domain they expect, but communicate with a different one.

The remainder of this section will describe in more detail how the attack is performed.

Marlinspike's SSLStrip attack demonstrated a combination of several techniques to monitor, and even modify, the SSL communications of other users on the network. SSLStrip exploits both software weaknesses and social engineering to fool users and client applications into thinking they are using a trusted site or server, when in fact they are using a fake one. Two of the important techniques are the *man-in-the-middle*, and the *null character attack*.

A man-in-the-middle attack allows the attacker to monitor and modify packets on the network. SSL is normally an effective defense because the attacker only sees encrypted packets. Marlinspike demonstrated ways for an attacker to circumvent this protection.

If the attacker is on the same local network as the server being compromised, Marlinspike's techniques make it very possible to perform the man-in-the-middle attack. A number of popular techniques exist for this: A rogue wireless access point is one, DNS cache poisoning is another.

If the attacker is not on the same network, then he needs to get there in order to carry out malicious activity. This can be accomplished by installing malware on a relatively less-secure system on the same network. Attacks of this type are legion.

► Null Characters in a Domain Name

One of the key threats Marlinspike shed light on is the use of null (zero value, often designated "\0") characters embedded in a domain name.

Online purchase of inexpensive "domain-validated" SSL certificates is so automated that it is usually possible to buy one with an embedded null character, for example – "\0thoughtcrime.org." In the attack, the domain name of the certificate is combined to the right of the domain name to be spoofed, for example, "www.verisign.com\0thoughtcrime.org." (thoughtcrime.org is a domain owned by Marlinspike, and used by him in his examples.)

When processing strings of text, most software treats the null character as a string terminator. So, in the example, when SSL client software reads the certificate domain name it will stop at the null and treat the certificate as valid for www.verisign.com, as issued by the certificate authority.

For more detail on how this attack works see Appendix B: Null Characters in Domain Names.

► The Auto-Update Attack Scenario

In the auto-update part of his presentation¹⁰, Marlinspike focused on Mozilla’s Firefox browser and Thunderbird e-mail client, which use the same automatic update service. Firefox and Thunderbird rely on a TLS connection to their update server for all of their protection. (TLS, or Transport Layer Security, is the latest generation of SSL encryption.) As Marlinspike put it, the conversation goes like this:

Firefox: TLS connect to server (aus2.mozilla.org)
“Hello, do you have any updates for me? Here’s my product, version, build ID, locale, OS, locale, and channel.”

Update Server: “As a matter of fact, I do. Here’s an unsigned blob of data – you’d do well to install it.”

The updated software is returned from the server either as an image of the complete application or just the portions that differ from the version the client is running. By default, “minor” updates are installed silently and the user gets no notice until the installation is done and the user is instructed to restart the browser.

Marlinspike showed how to trick Firefox or Thunderbird into installing a rogue update. This involved setting up a man-in-the-middle attack, monitoring the network for Firefox or Thunderbird clients polling for updates, and then responding from a fake aus2.mozilla.org to feed the clients fake updates.

As the server reports the version number of the update, and determines if the client will install it, it is a critical component from a security perspective. So, the fake update server can force malware updates on to the clients. Furthermore, Firefox add-ons check the same server for their own updates, and so may be similarly compromised.

The newest version of Firefox, at the time of this writing, is version 3.6. It does not address this problem in the update process. Numerous less famous programs update themselves with security on-par with Firefox’s. If such vendors were to code sign their updates, they would be far less vulnerable to attack.

Firefox add-ons check the same server and so may be similarly compromised.

► Whitelisting

The old model of most security products, including mainstream anti-virus products, is blacklisting: the vendor identifies threats and then issues signatures for those threats. The client software uses these signatures to look for known threats, and blocks them when they are detected. But the sheer increase in the number of threats in recent years, and a corresponding lag in the protection against new threats, have hampered blacklisting products.

To many in the security industry, the future is based on whitelisting. Whitelisting turns the problem on its head: All programs are blocked unless they are on an approved list. Whitelisting products have been hampered by the difficulty in implementing such an approach. Especially in a large, diverse enterprise, the number of legitimate executable programs that one might want to run is immense.

But a new generation of whitelisting products gets high marks for management of the process, and one of their key tools is the digital signature.¹¹

► Advantages Relative to Hashes

Whitelisting products have traditionally worked with file hashes, much like blacklisting products. This means that the management system has to keep a database of all program files the system might properly encounter and their hashes. Vendors have gotten creative about ways to manage the problem. Trusted installers, for example, designate specific applications to install other software on the system; trusted update sources allow any software that comes from a particular server or directory to be installed. Both of these approaches put the burden on IT to control tightly what comes through these trusted channels, but none of these compromise solutions are necessary when code signed signatures are available.

Code signed signatures allow IT to trust all software coming from a particular vendor or, more precisely, a particular signer of a Software Publisher Certificate. Updates to those programs don't change the fact that the base program is still trusted.

Perhaps a bigger advantage is that in-house software can be trusted through digital signatures (see the Development section below for information on the code signing process for developers). If software is used only in-house, it may be possible for the organization to create an in-house certificate authority and add its address to the trusted roots (a collection of certificates from trusted certificate authorities) of in-house systems. In-house developers can then sign their software with complete control of the process.

► Important Implementations

With the release of Windows® 7 and Windows Server® 2008 R2, Microsoft has released a free tool called AppLocker¹² that implements whitelisting of applications through Group Policy based on digital signatures. Administrators can even specify a minimum version number that is approved for the application. AppLocker is powerful considering the price, but it is not as powerful as the many third-party solutions that also run on a broader range of clients. See the above-referenced InfoWorld review for a good selection, almost all of which support code signing certificates for policy decisions.

In-house software
can be trusted
through digital
signatures.

¹¹ InfoWorld - Test Center review: Whitelisting security offers salvation (<http://www.infoworld.com/d/security-central/test-center-review-whitelisting-security-offers-salvation-835>)

¹² Microsoft – Windows 7 Feature Walkthroughs, AppLocker (<http://technet.microsoft.com/en-us/windows/dd320283.aspx>)

► Development

Signing code isn't hard, but in most environments it is performed with command line tools. Microsoft has a suite of command line tools included with their development platforms for creating and using code signing certificates.¹³ They are analogous to the tools offered by other vendors.

Microsoft's `makecert.exe` creates a digital certificate. This is necessary in order to create an in-house certificate authority to sign in-house software, but most users don't need to touch this tool. `cert2spc.exe` converts a digital certificate into the Software Publisher Certificate, which is a certificate in code signing format. `pvk2pfx.exe` imports the private key and software publisher certificate into a `.pfx` file - the type of file a customer normally gets when he buys a certificate from a CA. `signtool.exe`, the actual code signing tool, takes the `.pfx` file as input.

The JDK (Java Development Kit) comes with a similar suite of command line tools. With Java, once the certificate is installed into the Java keystore, the `jarsigner` tool is run, specifying the JAR file to sign and the certificate to use. The signature is added to the JAR file.¹⁴

Apple's code signing tools are part graphical tool, part command line. There are graphical tools for generating certificates, but the code signing utility is a classic UNIX-style command line tool that performs signing as well as verification of files.¹⁵

Microsoft's Visual Studio also integrates code signing into the development environment. Using the Solution Explorer, the programmer can select a certificate from the Windows® Certificate Store, from a file, or have Visual Studio generate a test certificate. Thenceforth, code builds will automatically sign the software.¹⁶

Many other integrated development environments are capable of running the command line tools in an automated fashion as part of the build process. Once set up, code signing is easy.

¹³ `makecert` (<http://msdn.microsoft.com/en-us/library/bfskty3%28VS.100%29.aspx>). `cert2spc` (<http://msdn.microsoft.com/en-us/library/f657tk8f%28VS.100%29.aspx>). `pvk2pfx` - (<http://msdn.microsoft.com/en-us/library/dd434714.aspx>). `signtool` - (<http://msdn.microsoft.com/en-us/library/8s9b9yaz%28VS.100%29.aspx>)

¹⁴ How to Sign Applets Using RSA-Signed Certificates (http://java.sun.com/j2se/1.5.0/docs/guide/plugin/developer_guide/rsa_signing.html)

¹⁵ Mac OS X Reference Library – “Code Signing Guide” (<http://developer.apple.com/documentation/Security/Conceptual/CodeSigningGuide/>)

¹⁶ MSDN - How to: Sign Application and Deployment Manifests (<http://msdn.microsoft.com/en-us/library/che5h906%28VS.80%29.aspx>)

Conclusion

Designers try to make security solutions as bullet-proof as possible, but there always has to be an element of trust involved. This was demonstrated long ago by Ken Thompson, principal author (with Dennis Ritchie) of UNIX, in a famous speech: “Reflections on Trusting Trust.”¹⁷ The take-away lesson from the speech was that at some point users have to trust some software they’re running, or the programs used to create the software.

Enterprise IT is not always in a position to independently judge which software is trustworthy. They can decide to trust the author of the program, which is essentially the same thing: They can’t trust a program without trusting its developer. The identity of the developer can be verified with digital code signing certificates.

Larry Seltzer has a background in software development, product testing, management of software development, and industry analysis, and is the author of over 1000 published articles on computer topics. He was one of the authors of NPL and NPL-R, fourth-generation languages for microcomputers by the now-defunct DeskTop Software Corporation of Princeton, NJ. For several years, he wrote corporate software for Mathematica Policy Research and Chase Econometrics. After several years as a consultant, he joined NSTL (National Software Testing Labs) developing product tests and managing contract testing for the computer industry, governments and publication. In 1991 Larry moved to Massachusetts to become Technical Director of PC Week Labs (now eWeek Labs). He moved within Ziff Davis to New York in 1994 to run testing at Windows Sources. In 1995, he became Technical Director for Internet product testing at *PC Magazine* and stayed there till 1998. Since then, he has been writing for *PCMag* and numerous other publications, including *Fortune Small Business*, *Windows 2000 Magazine* (now *Windows IT Pro*), *ZDNet* and *Information Week*. He remains a Contributing Editor at *PC Magazine* and writes their Security Watch blog. He is co-author of *Linksys Networks: The Official Guide*, author of *ADMIN911: Windows 2000 Terminal Services*. He has a Bachelors degree in Public Policy from the University of Pennsylvania.

¹⁷ Thompson, Ken – “Reflections on Trusting Trust” Communication of the ACM, Vol. 27, No. 8, August 1984, pp. 761-763. <http://cm.bell-labs.com/who/ken/trust.html>

Appendix A

Public Key Encryption and Digital Certificates

Digital certificates are documents that combine a public key and an identity. They can be used to verify that the public key belongs to the group or individual that purports to hold them.

Digital certificates can be generated by freely available software and issued by anyone to anyone, but their real value in the marketplace comes when they are issued by a trusted authority. These trusted entities are known as CAs (certificate authorities), which are entrusted with verifying the identity information on the certificate. The CAs sign the certificates so that third parties can verify their authenticity. If the party trusts the authority, and its verification procedures, they can trust the certificate itself and, by extension, any data or software signed with the certificate.

Operating systems, and some applications that verify digital certificates, come with a list of trusted roots, which are certificates from trusted certificate authorities. Any certificate trusted by the system must be signed by one of these CAs or by an affiliate authority whose certificate is signed by one of the trusted CAs.

Digital certificates viewed by a user also include information on the authority that issued them, because this is an important element of a trust decision. There is also a date range for which the certificate is valid, and the user agent will normally warn the user when a certificate is out of this range.

Code signing certificates are special digital certificates designed to identify the publisher of a piece of software. They are used to generate a digest, which is a mathematical value uniquely correlated with the software being protected. The digest, also known as a hash, cannot be used to recreate the software, and even a small change in the software will result in a substantial change in the digest. It is this digest that is then signed.

The signature and certificate are attached to the software. Users can obtain the public key of the publisher based on the identity stipulated in the certificate, and use it to verify that the software was signed by the publisher.



The most common use of digital certificates is in secure web browsing. When a user connects to a secure site, such as a bank, with an *https://* prefix as opposed to *http://*, communications are encrypted and the signer of the certificate is identified in the browser if the user clicks on the “lock” icon.

Extended Validation SSL (EV-SSL) certificates strengthen the security in this process by standardizing the verification which a CA performs on the applicant’s identity at a very stringent level.¹⁸ The browser also changes in appearance by turning a portion of the address bar green to show the user that they are working with an EV-SSL site

SSL is also used in many embedded applications, both between clients and servers and servers to servers. In these cases, EV-SSL can also be valuable in blocking the use of rogue certificates because the actual identity of the certificate owner can be checked.

► Certificate Chains of Authority

Trusted certificates don’t normally need to be issued directly by a trusted root authority. Through affiliate relationships, other authorities can issue certificates on behalf of the trusted authority, and in fact these affiliates

¹⁸ EV SSL Certificate Guidelines Version 1.1 - http://www.cabforum.org/EV_Certificate_Guidelines_V11.pdf

can have their own affiliates. The result is a chain of trust: the certificate contains the names of the various issuing authorities, and each parent authority signs the certificates of its child authorities.

In order to verify the authenticity of the certificate, the client program (usually a browser) walks up the chain, verifying the signature of each authority up to the root, which needs to be one of the trusted roots.

Appendix B

Null Characters in Domain Names

The heart of the attacks demonstrated by Moxie Marlinspike at Black Hat 2009 was the use of null characters in a domain name in a digital certificate.¹⁹

When a client connects to a server in an attempt to use SSL, the server responds, in part, with its certificate. The client then looks at the Common Name field of that certificate, which should contain the name of the domain of the server, and will then compare that name to the domain it expects, such as `www.verisign.com`.

When using a solution that is not from a trusted CA, it is possible to trick the client into mistaking the name it expects for a domain name of a malicious site belonging to an attacker.

When a customer buys a certificate from a sub-standard certificate authority the process is usually automated. If an unauthorized party requests a domain-validated certificate (e.g., `abc.verisign.com`), the CA parses the base domain name (`verisign.com`) from the request, does a whois lookup on that domain, and sends a request for authorization to the administrative contact for the domain. Whoever is in charge of this process for VeriSign turns down Marlinspike, and other unauthorized parties.

The Common Name field is one component of the DistinguishedName data grouping in X.509 certificates. (X.509 is the standard for digital certificate formatting used by SSL and most other applications.) Other fields include an Organization, Organizational Unit, Country State and Locale. But most SSL implementations don't care about anything but the Common Name. It is standard practice, with conventional SSL, to ignore the other fields with respect to authentication. Browsers may display those fields when the user clicks the lock icon, but they are not used for authentication.

X.509 certificates are formatted using a notation system called ASN.²⁰, which allows many string types. One of them is called an IA5String, and it is formatted with a byte length prefix in the style of programming in Pascal, a language popular in the 1980s: The string is prefixed with a byte that defines the length of the string, followed by the string data itself.²¹

Pascal-style strings are not common in programming these days, as C language-style strings predominate along with C-influenced languages. It is common for software that reads certificates to do so with C, or another language that handles strings the way C does.

C strings have no length indicator and, instead, are null-terminated. One major advantage of this is strings can be larger than 255 characters, which is a limit for Pascal in using a single byte. It also means that strings cannot contain a null character (a zero byte, often written as `\0`), because the string reading code stops when it reaches that character.

¹⁹ Null Prefix Attacks Against SSL/TLS Certificates - <http://www.thoughtcrime.org/papers/null-prefix-attacks.pdf>

²⁰ Wikipedia, Abstract Syntax Notation One - <http://en.wikipedia.org/wiki/ASN.1>

²¹ In fact, strings were not part of the original Pascal language at all but are an extension defined as part of the popular UCSD Pascal. (Wikipedia, UCSD Pascal - http://en.wikipedia.org/wiki/Ucsd_pascal)

To exploit the problem, as described by Marlinspike in his Black Hat 2009 presentation, the attacker buys a certificate for `www.verisign.com\0.attacker.abc`, for example (assume, for this example, that the attacker owns and controls the domain `attacker.abc`). The certificate authority then contacts the owner of `attacker.abc` – the attacker – and checks to see that he wants to buy this certificate. He says yes.

Only an inferior certificate authority permits the use of null characters in a domain name. Reputable ones, including VeriSign, check for nulls and reject certificate applications that contain them.

The attacker then installs this certificate on his server. Assuming he can get a client to reach his server after attempting to reach `www.verisign.com` (something he can do with other well-known attacks), the SSL tests pass and his site authenticates as `www.verisign.com`. This is because most SSL implementations compare the two names and stop at the null embedded in the certificate Common Name.

But, it gets worse. Having to buy targeted certificates for each target site can be cumbersome and expensive. To address this, CAs sell “wildcard” certificates that allow users to match to anything; `*\0.attacker.aaa` matches anything as a prefix to the base domain name.

At his 2009 Black Hat presentation, Marlinspike provided a long list of SSL client programs that were vulnerable to this attack, including Firefox, Internet Explorer, Outlook, AIM, Citrix VPN, and more. The problem is probably fixed in the current versions of many of the products he listed, but users often take years to upgrade.

On October 13, 2009, Microsoft acknowledged this vulnerability and issued an update to their CryptoAPI that addresses this situation.²² On systems that have the update applied, CryptoAPI rejects certificate names that contain null terminators.

²² Microsoft Security Bulletin MS09-056 - Vulnerabilities in Windows CryptoAPI Could Allow Spoofing. <http://www.microsoft.com/technet/security/Bulletin/MS09-056.msp>