Protecting Free and Open Communications on the Internet Against Man-in-the-middle Attacks on Third-party Software: We're FOCI'd

Jeffrey Knockel
Dept. of Computer Science
University of New Mexico
jeffk@cs.unm.edu

Jedidiah R. Crandall
Dept. of Computer Science
University of New Mexico
crandall@cs.unm.edu

Abstract

In this position paper, we argue that the potential for man-in-the-middle attacks on third-party software is a significant threat to free and open communications on the Internet (FOCI). The FOCI community has many challenges ahead, from the failure of the SSL system to protect Internet users from states that control the Internet to the challenges inherent in measuring and cataloging Internet censorship. It is already well-known in the community that man-in-the-middle attacks are a threat, and such attacks are already being used by nation states.

In this paper we discuss our experiences discovering two vulnerabilities in software update mechanisms (in Impulse SafeConnect and Sun Java). What surprised us was the relative ease of finding such vulnerabilities and exploiting them. Our argument is that automated tools are needed to help users manage this threat more effectively because the threat involves many third-party applications from many small vendors.

1 Introduction

Vulnerabilities in systems will always exist, and there will always be so-called "arms races" between attackers and defenders in environments where information security matters. When large software and communications infrastructures are developed over many years and then suddenly exposed to threats from a new angle that was never part of the design, however, this moves beyond "arms race" and can become a significant source of attacks that persists for many years. For example, when Windows systems were suddenly thrust onto the Internet with routable IP addresses, it took almost a decade before a user could (arguably) put a Windows machine on the Internet with a reasonable level of confidence that the machine would not be quickly compromised through open ports using memory corruption attacks.

In the context of free and open communications on the Internet (FOCI), it is important for users to be able to protect their systems from state actors that control most of the networking infrastructure. It is well known that this control enables man-in-the-middle attacks, and it is also known that man-in-the-middle attacks can include attacks on the integrity of executable downloads and third-party software updates. Our argument in this paper is that this kind of attack has the potential to become a persistent and widespread threat to free and open communications on the Internet, because the vulnerabilities are relatively easy to find and exploit compared to other types of vulnerabilities. Because many small, third-party software vendors may be affected, automated tools are needed to help users protect themselves without relying on the vendors.

We will not argue in this paper that the capability to carry out man-in-the-middle attacks is something that state actors will increasingly utilize. We will point out, however, that among the SSL certificates that Iran is suspected to have forged were several software update servers [24]. In this paper we will focus on arguing that typical Internet users are vulnerable to attacks on software updates by third-party software.

We also will not explore in depth the incentives of vendors who do not properly secure their software against man-in-the-middle attacks. It is worth pointing out that even the largest vendors can take years to fix known vulnerabilities of this type. Apple took more than three years to fix the vulnerability in iTunes updates that FinFisher exploited [17], and this exploit is believed to have been used against Egyptian dissidents [25]. Microsoft was prompted by the Flame worm to fix a vulnerability in its Windows update mechanism [18] that was based on weaknesses of certificates that use the MD5 hashing algorithm that had been known for almost four years at the time of Microsoft's fix [21]. These examples were simply a matter of major vendors not taking the man-in-the-middle threat seriously enough.

Even if the major vendors take this threat seriously, however, there are many small third party software vendors whose software performs automatic updates with administrator privileges. Depending on what the profile of programs installed on the average user's computer is, this problem may run much deeper than Microsoft and Apple. One of the vulnerabilities we discovered and describe in this paper, for example, was for a software program that is marketed specifically to U.S. universities. Many dissidents, journalists, and others who may be targeted by governments have ties to U.S. universities, so even if a software program is used only by a small segment of the population it may still be a fruitful endeavor for a government to develop a man-in-the-middle exploit for that software, especially given the relative ease of finding and exploiting man-in-the-middle vulnerabilities in software updates.

To support the statement above that finding and exploiting this type of vulnerability is relatively easy compared to, *e.g.*, 0-day memory corruption vulnerabilities for web browsers, we first describe two vulnerable third-party applications. In Section 2, we describe a man-in-the-middle vulnerability in Impulse SafeConnect. SafeConnect is software that many universities require all users of certain campus networks to install. Then in Section 3 we describe a man-in-the-middle vulnerability in a more pervasive program: Sun Java. Section 4 summarizes our findings from a few other programs, followed by discussion in Section 5, after which we conclude.

2 Impulse SafeConnect

Impulse SafeConnect is software that many universities require users of their networks to download and install as a prerequisite for using the network. It performs network access control and can scan a user's machine for patches, peer-to-peer activity, and other information.

In July 2011, we discovered that the automatic updating built into SafeConnect 5036.223 is vulnerable to man-in-the-middle attacks. When it detects that it has an Internet connection, SafeConnect attempts to connect to 198.31.193.211, a hard-coded IP address that, when connected to the University's wireless network, routes to an on-campus server. If connected, it begins communicating with the server via XML over insecure HTTP encrypted using the 12-byte Blowfish key in ECB mode:

```
\x4f\xbd\x06\x00\x00\xca
\x9c\x18\x03\xfc\x91\x3f
```

Among other things, information regarding software updates is sent by the server to the SafeConnect client, including the version of the update and both URL's and hex-encoded MD5 hashes for each updated file. If the version is newer than that currently installed, each file is then downloaded, its MD5 hash verified against the XML-provided hash, and the files are verified to have an "Impulse Point LLC" Authenticode digital signature. If these verifications pass, then SafeConnect silently replaces its files with the downloaded files and restarts itself.

We can exploit this update process via a man-in-the-middle attack. We first leverage that SafeConnect only uses symmetric encryption to protect its network communication. Thus, by reverse engineering the encryption key used by a SafeConnect client to decrypt update information, we also knew the key used to encrypt the update information. We also leverage that we also had analyzed an older version of SafeConnect, version 4250.121, that had a similar update process, that was also signed by Impulse Point, but that performed no digital signature verification on updated files. Thus, when SafeConnect attempts to connect to 198.31.193.211, we then exploit SafeConnect 5036.223 by spoofing an "update" to version 4250.121. After this version is installed, we can then spoof an update containing any arbitrary executable.

This vulnerability has been fixed by version 5059.242, which now uses HTTPS to perform all network communication. However, one must be on-campus to receive this update, so, *e.g.*, students who have graduated and moved away will remain vulnerable indefinitely.

3 Sun Java

In September 2011, we discovered that the automatic updater built into Sun Java 6 and 7 for Windows is vulnerable to man-in-the-middle attacks that enable arbitrary code execution as an Administrator. We will describe the vulnerability for Java 6 (Java 7's vulnerability is analogous).

The automatic updater in Java 6 periodically downloads update information from the following URL via insecure HTTP:

javadl-esd.sun.com/update/1.6.0/map-m-1.6.0.xml

This XML file maps older versions of Java 6 to the URL of another XML file with update information. For instance, the update information for Java 6 Update 30 to Update 31 is retrieved from the following URL via insecure HTTP:

javadl-esd.sun.com/update/1.6.0/au-descriptor-1.6.0_31-b79.xml

This XML file includes a textual description of the update presented to the user, a URL used to download the updater executable and the command-line options with which to execute it, and a hex-encoded SHA1 hash of the updater executable.

If an update is available, then the user is prompted with the description of the update. If the user accepts the update, then the executable is downloaded. Its SHA1 hash

is then verified against the XML-provided hash. Then the downloaded executable is verified to have a "Sun Microsystems, Inc." Authenticode digital signature and to have a PE version at least as high as the present version of Java installed. If the executable passes all verifications, then it is executed with the XML-provided command-line options.

We can exploit this update process via a man-in-the-middle attack. When a user downloads the XML file containing the specific update information, we can provide an XML file with our own executable URL, command-line options, and SHA1 hash. However, we must still provide an executable signed by Sun Microsystems with a version at least as high as the present version of Java installed. Moreover, we would like to be able to run arbitrary code if we pass it appropriate command-line options.

We found that a file named `javaws.exe` installed with Java meets these requirements. This executable is used to launch Java Web start applications from the Web. Thus, when the Java updater inquires about updates, we spoof a URL to a `javaws.exe` with the same version as the inquiring version of Java installed and spoof the following command-line options:

```
-Xnosplash␣-J-Djava.security.policy=
     http://url/to/grantall.jp␣
   http://url/to/hello.jnlp␣-open
```

where `http://url/to/grantall.jp` is a URL to a file containing

```
grant {
    permission
    java.security.AllPermission;
};
```

granting all permissions, including to run native code, to the target Web start application. `http://url/to/hello.jnlp` is a URL to a Web start application running arbitrary native code. The `-Xnosplash` option causes the Web start application to be executed silently. The `-open` option is used to pass arguments directly to the Web start application. We use it to "eat up" additional options that are unconditionally passed to the downloaded executable that would otherwise be considered by `javaws.exe` as erroneous and cause it to fail.

This vulnerability has been fixed in Sun Java 6 Update 31 and Java 7 Update 3 [3]. These versions download both XML files via HTTPS. Although the executable itself is still downloaded via insecure HTTP, since its SHA1 hash is downloaded securely, the downloaded executable is securely verified.

4 Other third-party software

Here we describe other third-party software that we looked at. Exploiting VirtualBox's update mechanism on certain platforms was trivial, since there was no attempt to secure the update mechanism on these platforms. Adobe Flash appears to have a substantial attack surface, but we were unable to exploit it. Google Chrome has a relatively simple mechanism with a small attack surface, that we were also unable to exploit.

VirtualBox 4.1.0 queries over insecure HTTP

update.virtualbox.org/query.php

for available updates with a query string indicating the user's operating system and the currently installed version. If an update is available, a clickable link is provided to download the installer for the updated version using the user's default browser. This installer is also downloaded over insecure HTTP. After being downloaded, on Windows, a user can then personally verify Oracle's Authenticode digital signature before executing the installer. On Ubuntu Linux, the downloaded Debian package provides no digital signature.

The Adobe Flash 10.3.181.3 automatic updater queries over insecure HTTP

fpdownload2.macromedia.com/get/flashplayer/update/current/xml/version_en_win_pl.xml

for the latest version of Flash. If an update is available, the updater downloads from the following hard-coded URL via insecure HTTP:

http://fpdownload.macromedia.com/get/flashplayer/update/current/install/install_all_win_pl_sgn.z

This URL is expected to contain the latest version. Although this file has no Authenticode signature, it is signed using Windows' Crypt32 API, and, after it is downloaded, its signature is verified using a public key built into the automatic updater. Although the version number of the download is not verified by the automatic updater, all signed installers released since we began testing, once executed, check to see if a version of Flash newer than the one being installed is already present. We do not know if there exist older versions of the installer that are signed but that do not perform this version check after being executed.

Google Chrome 18.0.1025.168 performs updates according to the Open Client Update Protocol [2], which we have found to perform secure updates even over insecure HTTP. The essence of the protocol is that, for each update request, the client randomly generates a shared, symmetric key. It is encrypted using the update server's public key. The client sends its update request and the encrypted shared symmetric key to the update server.

The update server decrypts the shared symmetric key using its private key and sends a response signed with the shared symmetric key. As an enhancement, to avoid the computational cost of the server having to perform asymmetric decryption responding to future update requests from the client, the server also responds with a cookie that can be used by the client to make future update requests that will not require the server to perform asymmetric cryptography to generate a response.

We found that the responses returned by the update server were XML files containing whether an update was available and, if so, an insecure HTTP URL at which to download the updater and its base64-encoded SHA1 hash. After download, the hash of the updater is verified against the XML-provided hash before being executed.

5 Discussion

In this section, we first enumerate potential fixes to the problem of man-in-the-middle vulnerabilities on software updates, with some thoughts about each, and then we suggest a few avenues for further research that could be performed to understand the scope of this problem and begin to address it. We also briefly discuss the role of open source software and some ethical disclosure issues.

What was most surprising to us was the relative ease with which we were able to find man-in-the-middle attacks on software updates. Originally, we had been reverse-engineering SafeConnect for privacy concerns, but the potential for a man-in-the-middle attack of some kind became apparent upon viewing a TCP dump that showed no evidence of asymmetric cryptography. It took some work to find that arbitrary executables could be installed, but within a week of beginning our analysis of SafeConnect we had developed an exploit capable of installing a backdoor on the computer of almost any member of our university community (or many other universities that used the same software). Months later, researchers at the Electronic Frontier Foundation and the University of Michigan independently found the same vulnerability [11, 5].

After discovering the vulnerability and the process of alerting the vendor, university, and university community about the vulnerability in SafeConnect had played out [16, 9, 15], we became interested in man-in-the-middle attacks on software updates and within a week we had several exploits, including one for Sun Java which is one of the most commonly installed third-party applications on any platform. Although we had independently discovered this vulnerability, we next discovered that a similar (but not as powerful) exploit was part of a toolset [13] put together by security researchers in Argentina [14] for exploiting man-in-the-middle attacks on third-party software updates. At this point it became apparent to us that finding the vulnerabilities was not a challenging research problem and that the vulnerabilities were not generally unknown to vendors nor attackers, so we directed our efforts elsewhere.

The question that remains is, what can we do to protect users in environments where man-in-the-middle attacks by state actors are probable? Possible mitigations for this type of attack on software updates include:

- **Find vulnerabilities and pressure vendors to fix them:** This approach does not scale well because of the number of third-party vendors and the time-consuming nature of reverse-engineering. Also, the Digital Millennium Copyright Act (DMCA) appears to allow for this kind of research, but exemptions are not automatic and may require adherence to certain processes.

- **Give users tools to detect unsafe updates by third-party software on their machines:** Even just from the relatively small number of third-party programs we have looked at, it is apparent that distinguishing secure automatic updates *vs.* insecure automatic updates is an extremely challenging problem. Compare, for example, the sophisticated and insecure signature checking of vulnerable versions of Sun Java to the relatively secure scheme of Google Chrome, which in the end is downloading an executable binary over plain HTTP. A tool that used dynamic analysis on machine code and memory and was able to determine which of these schemes was secure and which not seems well beyond the reach of current dynamic analysis research. For efforts to automatically detect cryptography in binary code see [26, 12, 10].

- **Educate users:** There are already many efforts to educate users who are using technology in countries that control the Internet, such as the Tor project [23] or the Open Net Initiative [19]. These are very important and effective efforts that hopefully can reach many users, but within the specific context of man-in-the-middle attacks on software updates complementing these efforts with technologies that help protect users should be a priority for the FOCI community.

- **Re-think trust in distributed systems:** Efforts to re-think the SSL system [8, 6] or build trust from social networks may be very valuable in addressing the threat of man-in-the-middle attacks on third-party applications. The reason for this is that the SSL system, which is currently the most viable way for third-party software vendors to perform secure updates, creates a cost barrier and is based on less than ideal notions of trust. Any improvements in

this space will give third-party software vendors more options to do updates securely.

- **Some type of service that handles updates securely without cooperation from the third-party vendors:** It may be possible to use a host-based firewall or routing table configuration to re-route outgoing connections that may be related to updates and then tunnel and encrypt them and handle them in a secure way, even if the software trying to do the update is itself using plain HTTP. Note that Tor [7] can be configured to route all outgoing connections through the Tor network, but might not be the ideal solution in the specific case of handling third-party updates for day-to-day use of normal users. The Update Framework [20, 22] is a framework for securing software updates that is aimed at developers.

Potential avenues of research to understand and mitigate this problem include:

- **How vulnerable are average Internet users to man-in-the-middle attacks on third-party software updates?** This is largely a function of what the distribution of third-party software used by users is and how vulnerable various third-party programs are. In terms of the latter, we analyzed a relatively small number of third-party programs and found several serious vulnerabilities. For the former, this question can be answered statistically by collecting data at a large network gateway. This would have to be done in a privacy-preserving way, which is challenging because identifying an unknown third-party software update in HTTP traffic is not possible without saving many details about the HTTP flow. Note that simply downloading an executable EXE file puts users at risk, so a first-order approximation of how at-risk the average user is could be measured by searching for a signature of HTTP-encoded PE headers or other indications of executable binaries being downloaded.
- **What forensic procedures and mechanisms must be put in place?** There is a great deal of activity and expertise surrounding the problem of taking apart malicious binary programs and finding out about their purposes, origins, *etc.* Is there anything special we need to anticipate with respect to detecting and analyzing state-sponsored backdoors? One precedent for analysis of a state-sponsored backdoor was the Chaos Computer Club's analysis of Bundestrojaner, which was used by law enforcement in Germany [4].
- **What is or is not feasible with respect to informing users about potential insecure third-party software updates on their systems?** Dynamic analysis to determine the security of third-party updates with reasonable false positive and false negative rates seems infeasible, based off of the examples we have shown in this paper. However, it might be possible to combine dynamic analysis with visualization and research in human-computer interaction to make a best effort to inform users about what is happening on their systems.

Open source software projects tend to cooperate with one another so that open source operating system distributions, *e.g.*, Ubuntu, have a unified software distribution model where most software a user needs is installed and updated securely through a package management system. This does not mean that open source operating system users are not vulnerable to attack (the VirtualBox vulnerability was particularly eye-opening to us because we were ourselves vulnerable to man-in-the-middle attacks on our personal machines). It does mean, however, that open source operating systems make it much more feasible for a user, with the help of the open source community, to know what is happening on their machines and protect themselves against man-in-the-middle attacks.

Walled garden approaches such as that of Apple's iOS App Store enforce that all applications receive updates only securely through the App Store. However, walled garden approaches can also be used to restrict freedom on the Internet by enforcing censorship of what software you are allowed to run. Apple's iOS App Store guidelines explicitly prohibit applications that contain defamation, violence, objectionable content, pornography, mean-spirited or inaccurate religious references, or any content that is "over the line" [1]. These guidelines also explicitly prohibit applications that launch executable code or that do not use Apple's WebKit layout and Javascript engine. This precludes much of the software that we analyzed, including virtual machine technologies like Java and Flash and competitor browsers like Google Chrome.

An interesting issue that came up in our research is ethical disclosure. Leaving aside the many issues that ethical disclosure raises, one aspect of man-in-the-middle attacks that raised questions for us was: what kinds of vulnerabilities can really be "disclosed?" A very carefully crafted input that causes a remote buffer overflow does raise issues of "disclosure" since vendors, attackers, and affected users probably will not know this information on their own without investing resources and time into a research effort (*e.g.*, fuzz testing). On the other hand, is the lack of asymmetric cryptography in a software update something that can be "disclosed" to anybody but the affected users? It is well known that network communications that do not utilize asymmet-

ric cryptography are susceptible to man-in-the-middle attacks. The vendor should already know the behavior of their own software, and a skilled attacker can spot the vulnerability with relatively little effort in many cases.

6 Conclusion

We have argued that man-in-the-middle attacks on third-party software updates are a significant threat to free and open communications on the Internet that could prove to be persistent and widespread. This threat goes beyond major vendors, so that automated tools are needed to help users protect themselves from insecure third-party updates performed by the programs they install on their computer. More research is needed to understand the scope of this threat and start taking steps to address it.

Acknowledgments

We would like to thank the FOCI anonymous reviewers and our shepherd, Nicholas Weaver, for valuable feedback. We owe gratitude to several people at the University of New Mexico office of Information Technologies for helping with vendor notification. This material is based upon work supported by the National Science Foundation under Grant Nos. #0844880, #0905177, and #1017602. Jed Crandall is also supported by the Defense Advanced Research Projects Agency.

References

[1] App Store Review Guidelines. Retrieved May 3, 2012, from https://developer.apple.com/appstore/resources/approval/guidelines.html.

[2] Open Client Update Protocol. http://omaha.googlecode.com/svn/wiki/cup.html.

[3] Vulnerability Summary for CVE-2012-0504. Available at http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2012-0504.

[4] Chaos Computer Club analyzes government malware. http://www.ccc.de/en/updates/2011/staatstrojaner.

[5] COHN, C., AND SCHOEN, S. Safeconnect, Universities, P2P, Network Security and Risk: The Tangled World of "Policy Enforcement" on Other People's Computers. Available at https://www.eff.org/deeplinks/2011/10/safeconnect-universities-peer-peer-file-sharing, October 6 2011.

[6] Convergence. http://convergence.io/.

[7] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The second-generation onion router, 2004.

[8] Electronic Fontier Foundation SSL Observatory. https://www.eff.org/observatory.

[9] ERVEN, C. IT: SafeConnect glitch-free, don't uninstall. Daily Lobo, 30 August 2011, Available at http://www.dailylobo.com/index.php/article/2011/08/it_safeconnect_glitchfree_dont_uninstall.

[10] GRÖBERT, F., WILLEMS, C., AND HOLZ, T. Automated identification of cryptographic primitives in binary programs. In *Proceedings of the 14th international conference on Recent Advances in Intrusion Detection* (Berlin, Heidelberg, 2011), RAID'11, Springer-Verlag, pp. 41–60.

[11] GUERRA, C. How Safe is SafeConnect? The New School Free Press, 29 March 2012, Available at http://www.newschoolfreepress.com/2012/03/29/how-safe-is-safeconnect/.

[12] GUILFANOV, I. Findcrypt. http://www.hexblog.com/?p=27.

[13] INFOBYTE. ISR-evilgrade readme. Available at http://www.infobytesec.com/down/isr-evilgrade-Readme.txt.

[14] INFOBYTE. Java exploit demo for Windows 7. Available at http://www.infobyte.com.ar/demo/java_win7.htm.

[15] KNOCKEL, J., AND CRANDALL, J. SafeConnect now fixed through legal curiosity. Daily Lobo, 6 September 2012, Available at http://www.dailylobo.com/index.php/article/2011/09/safeconnect_now_fixed_through_legal_curiosity.

[16] KNOCKEL, J., AND CRANDALL, J. Security vulnerability plagues SafeConnect. New Mexico Daily Lobo, 29 August 2011, Available at http://www.dailylobo.com/index.php/article/2011/08/security_vulnerability_plagues_safeconnect.

[17] KREBS, B. Apple took 3+ years to fix FinFisher trojan hole. Available at http://krebsonsecurity.com/2011/11/apple-took-3-years-to-fix-finfisher-trojan-hole/.

[18] KREBS, B. Flame malware prompts Microsoft patch. Available at http://krebsonsecurity.com/2012/06/flame-malware-prompts-microsoft-patch/.

[19] The Open Net Initiative. http://opennet.net.

[20] SAMUEL, J., MATHEWSON, N., CAPPOS, J., AND DINGLEDINE, R. Survivable key compromise in software update systems. In *Proceedings of the 17th ACM conference on Computer and communications security* (New York, NY, USA, 2010), CCS '10, ACM, pp. 61–72.

[21] SOTIROV, A., STEVENS, M., APPELBAUM, J., LENSTRA, A., MOLNAR, D. A., OSVIK, D. A., AND DE WEGER, B. MD5 considered harmful today: Creating a rogue CA certificate, December 2008. 25th Chaos Communications Congress, Berlin, Germany.

[22] TUF: The Update Framework. https://www.updateframework.com/.

[23] The Tor project. https://www.torproject.org.

[24] TOR BLOG. DigiNotar Damage Disclosure. Available at https://blog.torproject.org/blog/diginotar-damage-disclosure.

[25] WIKIPEDIA. FinFisher. Available at http://en.wikipedia.org/wiki/FinFisher.

[26] WRIGHT, J. L., AND MANIC, M. Neural network approach to locating cryptography in object code. In *Proceedings of the 14th IEEE international conference on Emerging technologies & factory automation* (Piscataway, NJ, USA, 2009), ETFA'09, IEEE Press, pp. 1658–1661.