

CS691I Term Paper: Authentication Failure Attacks in Software Update Mechanisms

Anthony Bellissimo John Burgess
Dept. of Computer Science, Univ. of Massachusetts, Amherst, USA 01003
{twon, jburgess}@cs.umass.edu

December 14, 2005

1 Motivation and Contributions

Software update mechanisms are an integral part of distributing updated code to end users. They can deliver bug fixes, feature additions, and code exploit defenses. Without the presence of simple, often automatic, software update mechanisms, many end users are unlikely to download and install updates regularly. The rapid deployment of updated code is especially important when new exploitation opportunities are revealed and available to attackers at large. Unfortunately, the update mechanisms deployed in many common applications do not make use of accepted computer security practices and are therefore vulnerable to attack. We examined a variety of update programs' abilities to detect and avoid attacks, as well as demonstrated automated attacks against two products in particular.

2 Background / Related Work

To date, the only related paper we have discovered is one by Michael et al. [1] that discusses system design considerations for securely updating the software on software defined radio (SDR) systems. While their motivation had to do with the legal and safety issues surrounding a maliciously malfunctioning RF transmitter, the techniques they propose to do the updating apply to all automatic update mechanisms. The upshot of the paper is that signing updates with a private key whose corresponding public key is installed on the devices that are being updated is a simple, effective way of guaranteeing authenticity.

It is possible that the body of work on the subject is small simply because the fix is so obvious. Nevertheless, multiple systems we have examined fail to provide even this level of authentication, and with at least one system the authenticity of the updates themselves is entirely irrelevant given that the existing trusted code that installs them is driven using untrusted scripts that are downloaded and executed without verification.

3 Potential Targets

3.1 Operating Systems

Operating system updaters such as Apple’s Software Update and Microsoft’s Windows Update are clearly the most appealing targets for this sort of exploitation. Operating system vendors like Microsoft and Apple were some of the first proponents of “signed software” authenticity mechanisms. Both vendors claim that their operating system update systems are secure. We have verified this, as well as ensured that a naïve user cannot “OK” their way into running malicious code while updating their operating system software.

3.2 Root Access Services

Programs that run constantly and require root or other administrator-level access are a similarly attractive target for attackers targeting update programs. Their continuous operation makes them a viable target on a large percentage of computers, and their high privilege level make their exploitation a potentially catastrophic event for a system. Even if no arbitrary code execution is possible, malicious updates of virus definitions could allow the attacker to use the antivirus program to delete or disable programs performing critical tasks, including the antivirus software itself. Although many services such as antivirus programs are marketed as security tools, we will attempt to determine if their security emphasis extends to their software update mechanisms.

3.3 User Programs

Although user programs typically run with restricted privileges on a system running Unix or Mac OS X, they often run with administrative privileges on a Windows machine. On a Unix or Mac system, a compromise of these applications can damage anything to which the executing user has access, such as his or her data files. On Windows systems, users commonly run with full administrative privileges so that they can install software themselves. Execution of malicious code by any program in this environment can lead to catastrophic results for the whole system.

4 Preliminary Findings

Software	Platform	Auth. Connection?	Auth. Binaries?
Windows Update	Windows	yes and no	yes
Apple Software Update	MacOS	no	yes
Adobe Acrobat	MacOS	no	yes
Microsoft Office	MacOS	no	yes
Mozilla Firefox	Windows	yes and no	no
Fugu	MacOS	no	no
McAfee VirusScan	Windows	no	no
McAfee Virex	MacOS	no	no

5 Examined Software

5.1 Microsoft Windows Update

Windows Update appears very resistant to attack. When the software is invoked, either automatically or by a user's manual visit to windowsupdate.com, it invokes a script on Microsoft's webserver that lists all the updates available. It then compares the list of retrieved updates to the computer's contents to determine whether any packages should be updated. If so, it downloads signed packages from the webserver, verifies their authenticity, then extracts and installs them. Additionally, some binary data is exchanged with Microsoft during the update procedure out-of-band with respect to the browser. We were unable to determine the significance of this data, though we believe it to be encrypted.

When the software is invoked from a visit to windowsupdate.com, it instantiates an ActiveX control¹ that exposes the functionality necessary to download and install the update packages. The only potentially exploitable external interface of this control allows the limited instantiation of other controls already resident on the user's computer. However, it specifically whitelists the instantiation of COM objects² with names beginning with "Microsoft.Update.". All components necessary to download and install updates contain this prefix. Through experimentation, we verified that by modifying the update DLL to whitelist against a shorter string, we could convince it to instantiate COM objects with arbitrary prefixes. However, by the time an attacker has enough access to the host system to perform this sort of low-level modification of system files, he already has enough access to do whatever else he wants.

¹ActiveX controls are Windows-native binaries that can be embedded in (invoked from) web pages, and often serve as a bridge between interpreted script on the web page and real code on the user's machine.

²Component Object Model – ActiveX controls are the subset of COM objects that may be invoked directly from a web page.

5.2 Apple Software Update

MacOS Software Update employs a simple yet effective signature system to ensure the integrity of its updates. Each archive downloaded contains a file named “signature” containing a 1024 byte signature of the hash of the accompanying installation executable. Each installation binary is checked against its signature which may only be signed by the private key held by Apple Computer Corp. (whose public key is included on the operating system’s installation media). No encrypted connections are needed, nor verification that the IP address of the server being downloaded from belongs to Apple.

This scheme allows system administrators running MacOS Server to mirror updates on the local server for network clients to install without providing attackers an opportunity to introduce malicious updates to unsuspecting users. Because of this signature checking, MacOS Server administrators may only deploy updates retrieved from Apple, and not arbitrary code they might want installed on client machines.

5.3 Adobe Acrobat Reader

Acrobat Reader’s update mechanism also appears secure (we examined the OS X version). The software gets the locations of specially-formatted update files from a webserver, then downloads the update files it requires. Through extensive examination of the update file format, we have determined that it includes both a OS X-native disk image containing the update itself and a corresponding signature for the disk image. Though we were unable to determine the precise algorithm and key-length used to generate the signature, we located the corresponding public key in the Acrobat Reader application package.

5.4 Microsoft Office Update

Microsoft’s update mechanism for their MacOS Office products downloads an XML-like list of available product updates and product versions whose presence should trigger their installation. Software certificates signed by Microsoft are included in the XML-like file for each update that can be downloaded. Binaries downloaded for update are checked against the signature in the corresponding certificate and not executed if the signature does not match. Microsoft’s public key for verification of signatures is signed by Verisign, whose root public key is included on the MacOS installation media.

5.5 Mozilla Firefox 1.0

Firefox downloads its update packages from a mirror server over unencrypted HTTP, but the list of updates and their locations itself is downloaded over HTTPS. We experimented briefly with SSL man-in-the-middle and the Firefox updater, and Firefox notified the user in no uncertain terms that someone was doing something nasty with the update process. It did eventually allow the user

to accept our certificates and retrieve the list through the middleman, but only after clicking through two warnings. The first warning notifies the user that the certificate is not signed by a trusted CA, and the second warns that the domain on the certificate is not the same as the domain to which the connection is being established. It's possible that we could have reduced this to one warning about the CA, if we had created our own CA and then created a forged certificate for mozilla.org. This may be a moot point: Firefox 1.5 was released during the course of the semester, and instead of allowing users to "OK" through SSL warnings, it displays an error message indicating (falsely) that the update location file retrieved from the server is malformed. This closes this avenue of attack.

However, the list of updates and locations contains no file hash information. This allows either of two attacks: either we can provide false DNS entries for hosts that are known to be Firefox mirrors, or we can intercept requests for filenames known to be Firefox updates and return malicious content instead. Since these attacks did not fit cleanly into the framework we developed for carrying out our attacks on McAfee VirusScan and Virex, we did not have enough time to attempt them. Since we cannot falsely advertise the existence of an update (Firefox 1.5 ensures that we cannot be an SSL man-in-the-middle), this limits the pool of possible targets to those users with out-of-date Firefox installations. Therefore, the optimal time to execute an attack like this is clearly whenever a major update to the software is released.

5.6 Fugu

Fugu is commonly-used free sftp/scp agent for MacOS X. Fugu, like many Mac applications, has a "Check for Updates" function on its menu bar. The program downloads an XML document from a predefined location detailing the newest available application version along with the location where the program may download the latest version. Although a phony DNS response or XML interception-and-modification could easily cause the mechanism to download malicious code, only a disk image is mounted to the user's desktop. Execution of the downloaded program must be initiated manually by copying the binary from the disk image into the "Applications" folder and then calling the executable as normally required when running Fugu. The user may execute malicious code, but it will be restricted to the user's privileges unless the malicious application requested and obtained a root password from the user. An naïve user might be fooled into producing the root password for malicious code, but this attack is not substantially easier than posting a malicious executable on a web site for a user to download and execute manually.

5.7 McAfee VirusScan

When VirusScan is first installed, it installs and registers a number of ActiveX components. This must be done with the user's explicit consent, though once the controls are installed they may be re-instantiated freely. The main application's

dialogs are built from HTML and VBScript, and rendered using Microsoft's HTML engine. To start the update process, the client makes a HTTP POST request with version numbers of the component DLLs, executables, and virus definition file. It then parses the reply generated by the server to determine whether any components are out-of-date. If so, the client requests a number of documents from the server over HTTP, including VBScript instructions that control the behavior of the ActiveX components that the user authorized when the software was installed.

Although we cannot say whether the ActiveX controls expose more functionality than necessary, they certainly expose enough to be dangerous. One of the functions exported by one of the controls is a simple wrapper around the Windows API ShellExecute function (analogous to the POSIX fork/exec*/waitpid or spawn* family of functions). By injecting commands into the VBScript returned by the server, the McAfee Updater can be forced to execute arbitrary commands on the Windows machine with the access level of the updating user. This problem is compounded by the fact that the user must have administrative privileges to update the software. We exploited this functionality to download and execute code of our choice. We injected the commands using a simple program that sits between the victim and a Squid web proxy/cache which has the ability to modify the content of messages in either direction. We modify the inbound data to reflect that the client is out-of-date (whether or not this is the case), which causes the client to request the exploitable VBScript from the server. We then inject our commands into the desired place in the returned VBScript in order to trigger the desired behavior.

We briefly examined the "Enterprise" edition of McAfee's VirusScan software, intended to be deployed internally at institutions such as corporations and universities. When this product performs an update, no script is downloaded from the network to drive the update mechanism. Instead, it simply retrieves named files from an administrator-configurable location, determined at install time. Additionally, the updates themselves are encrypted and signed using public-key cryptography. We are unsure why this markedly more secure design was chosen for one product but not the other.

5.8 McAfee Virex

Virex's update manager asks the user to authenticate as an administrator, then proceeds to log on anonymously to McAfee's FTP server and checks the time of the files in a particular directory. This time is given in the filename, and has nothing to do with the timestamp on the server. If it finds an update newer than the last one it retrieved, it downloads and unpacks it. The update file contains an OS X installer package, which can contain arbitrary code, as well as pre- and post-installation instructions. Since Virex has already obtained the root access needed by the installer, it silently runs the install package in the background after it retrieves it.

We successfully convinced the Virex updater to install and run our arbitrary code merely by configuring our DNS server to provide a false address for

McAfee’s FTP server (its own). We mirrored the directory structure of the real server on our fake server, and placed a package “dated” January 1, 2006 (V7060101.gz) at the location that the updater searches for updates. The updater happily downloaded and installed the files in the package, and we used the post-install script in the package to execute the code that we installed. The code ran with full root privileges.

6 Suggested Improvements

For some of the software we observed to be insecure, fixing the problem is not complicated. In the case of Fugu and Virex, the inclusion of a signature file on the web/FTP server that could be retrieved at the same time as the update package would prevent the software from installing forged updates from an attacker (assuming a secure choice of signature scheme, sufficient key length, and the rest of the standard assumptions). McAfee’s “VirusScan Enterprise” product for Windows already does precisely this, so we are again unsure why this functionality was not carried over to Virex. A signature scheme could be implemented for Firefox, but an even simpler fix might be the inclusion of file hashes in the update list. Since Firefox 1.5 does not allow the retrieval of update lists from hosts with arbitrary SSL certificates, an attacker cannot modify the hash values received by the client.

Fixing VirusScan is a more difficult proposition. A short-term fix would be to transfer the scripts that drive the ActiveX control over SSL. However, because much of the software relies on Microsoft’s HTML engine for its network connectivity and display, it is possible that the handling of invalid SSL certificates may not be left to the application (as it is in Firefox). Therefore, an adversary might be able to convince a user to accept the forged SSL certificate, leaving the process as vulnerable as it is over plain HTTP. A better fix involves the rethinking of the level of access required by the ActiveX control involved.

6.1 ActiveX considerations

When an ActiveX control is installed, it is typically bound to a domain from which it is allowed to be instantiated, though this is not always true (plugins such as Macromedia’s Flash Player and Sun’s Java can be instantiated from anywhere). We were unsuccessful in invoking the vulnerable ActiveX control on another webpage, which indicates that this part of the installation process was done properly. However, this doesn’t solve the problem of malicious DNS replies indicating that an untrusted machine is the requested host. We successfully pointed “www.mcafee.com” at our attacker, then used the root page to instantiate the vulnerable controls and run the exploit. A more sophisticated attack would have used something like Apache’s native URL Rewriting to ensure that any requests for anything on that host would have pointed to a page which contained the malicious script, and possibly a “mcafee.com is down for maintenance” message to defuse user suspicion. Meanwhile, the code installed

and executed by the control would already be doing its work silently on the user's machine.

It is important to note that despite the amount of bad press Microsoft's ActiveX technology has received with respect to security, the central problem with the technology is that, even though it is being instantiated and scripted over an untrustworthy medium (the network), the user must trust the control not to abuse the very low-level control it has over her machine. In addition to our exploitation of McAfee's update control, there have been other, higher-profile ActiveX-related exploits in the news. One involved Sony's XCP DRM software; specifically, the original uninstaller they released came packaged as an ActiveX control that remained on the user's machine after the uninstallation of the DRM software was complete. Not only could this control be instantiated from pages on an arbitrary domain (unlike the McAfee control), it also exposed sufficient functionality to download and execute arbitrary code without verification that it came from a trusted source (in this case, Sony or their partner First4Internet, who developed the software). A similar problem of equivalent severity affected Sony's other DRM system, MediaMax, whose ActiveX uninstaller contained equivalent functionality and was also instantiable from an arbitrary domain. A Google search for "ActiveX security flaw" yields pages and pages of results, most of the form "control x enables a remote machine to invoke undesirable behavior y when given input z ."

6.2 Economy of Mechanism

Taken as a whole, this suggests that the principle of *economy of mechanism* applies especially to this style of application. Economy of mechanism is the idea that the design of a system should be as simple as possible to facilitate analysis of its correctness (this is a superset of the KISS principle³). A critical question to ask in analyzing a system this way is exactly where the simplicity comes from. We illustrate this by contrasting the exposed interface of McAfee's update control with Microsoft's Windows Update control, then examining a potential fix for the functionality provided by McAfee's excessively broad interface.

Recall that the reason McAfee's update control can be exploited is its inclusion of an interface to the Windows API ShellExecute function, which allows the execution of arbitrary commands. As far as we can tell, the only time ShellExecute is invoked from the legitimate VBScript is when the script tries to launch a particular URL. That line looks like this:

```
call gobjShell.ShellExecute(sURL,sArgs)
```

While arguably "simple" in some respects, the use of Windows functionality designed to analyze a command and arguments and do the context-sensitive "right thing" with it is complete overkill for such a simple operation. The impor-

³KISS – "Keep it simple, stupid."

tant observation here is that, rather than the update system having predictable control flow after invoking this function, literally anything could happen as a result of this function being called. Compare this with the limited exposure of Microsoft’s interface to its update control, in which the only command that accepts arbitrary, unstructured input immediately verifies the input against a whitelist of legal values, returning an error if it does not match the whitelist pattern.

We propose that the interface exposed by McAfee’s update control can be drastically simplified, bringing about a corresponding decrease in exploit potential. Since `sArgs` is empty here, the above call to `ShellExecute` could be shortened to a special-purpose call like:

```
call gobjShell.OpenURL(sURL)
```

`OpenURL` can validate its input to ensure that it is sensible before opening it. Furthermore, the contents of `sURL` here are the fixed location of the update script and parameters indicating the install type, the “branding” of the installation (in our case, Comcast), and whether or not to pop up an interactive window. We can go one step further with this and propose this function prototype:

```
call gobjShell.OpenUpdateWindow(branding,installType,doPopup)
```

or even

```
call gobjShell.OpenUpdateWindow(installType,doPopup)
```

to allow the control itself to fill in its own branding information. Here, even the potential to launch an arbitrary URL is mitigated by `OpenUpdateWindow`’s construction and launch of the URL itself, from parameters that can be trivially checked for safety.

7 Conclusion

Though the principles of secure, authenticated software distribution are well-understood, we have shown that these principles are not necessarily followed, even in popular, high-profile software. We have also observed that in some software designs, the authenticity of the distributed files themselves can be of secondary importance when compared with their delivery method. It is unclear what can be done to resolve this — even vendors that demonstrate their concern about the authenticity of updates to their software in some products ignore these principles in the development of other, similar products.

Like many software problems, this dilemma seems to imply the need for a standardized mechanism for software updates such that application developers need not "reinvent the wheel" each time a new program is deployed. It would be practical to have an extensible framework (possibly included with an operating system) which developers could register with for their update needs. Applications would need only supply the mechanism with a public key for updates to be checked against, along with the network location of an XML document describing updates and their download instructions for that application. The software updater itself could perform installations with necessary privileges, unburdening applications from needing individual root access for installation related tasks. Much like encryption algorithms, a good software update mechanism could serve as a single point of failure that could be scrutinized such that developers utilizing the module can be assured of its security properties provided they can comply with framework requirements.

8 Division of Labor

- John – Provided equipment, captured most network traces. Analysis: MacOS Software Update, Microsoft Office Update, Fugu, Virex
- Anthony – Created man-in-the-middle script, configured attacker/victim. Analysis: Windows Update, Acrobat Reader, VirusScan, Firefox
- Both – Writing, target enumeration

References

- [1] Lachlan B. Michael, Miodrag J. Mihaljevic, Shinichiro Haruyama, and Ryuji Kohno. A proposal of architectural elements for implementing secure software download service in software defined radio.