Insecure updates are the rule, not the exception

Mixed Signals

Kurt looks at the practice of code signing and examines why so few upstream open source projects actually do it.

By Kurt Seifried

First, the good news: Most major Linux vendors sign their software and source code packages, thereby allowing end users and administrators to verify that the code has indeed originated from the vendor and hasn't been modified or tampered with after being signed.

Most tools like Yum and RPM check package signatures by default and will refuse to install or upgrade packages unless they are properly signed (which you can override manually). So, why am I worried about upstream code signing of updates? Because very few open source projects actually sign their code properly, if at all. This means that if an attacker breaks into a distribution site, they can modify existing source code packages or upload new ones with malicious code. Users cannot easily detect a problem unless they have a copy of the code for comparison or they compare the changes in the new version against an old version. And yes, people do break into major sites (e.g., the Linode hack earlier this year) [1].

Why People Don't Sign Code

Why don't upstream open source projects sign their code properly? Doing so would allow vendors like Debian and Red Hat and end users to verify easily that the code was signed by the project in question. There are several reasons: the first is that signing code correctly, even minimally, is a chore and requires some setup. But, more importantly, it requires ongoing discipline; you must protect the signing keys forever, you must sign and verify the signed code, and you must handle key management. Additionally, code signing only proves that a file containing certain content was signed; it does nothing to attest to the quality of the code (everyone has security flaws in their software). For all you know, the upstream project has taken code contributions containing a back door.

The third and, in my opinion, biggest reason is that unsigned code and all the

KURT SEIFRIED

Kurt Seifried is an Information Security Consultant specializing in Linux and networks since 1996. He often wonders how it is that technology works on a large scale but often fails on a small scale.

problems that come with it don't cost the code author anything. Virtually all the costs of distributing malicious code are borne by end users; the author rarely incurs any cost or penalty.

A perfect example is the Social Media Widget incident [2]. The company responsible for this WordPress plugin outsourced development to a company that inserted malicious PHP code. Fortunately (as far as is known), the code inserted just displayed an ad and didn't attack the server or users' systems. The WordPress team accepted this explanation and reinstated the plugin. So, the cost in terms of money, time, and effort was basically zero for the Social Media Widget plugin.

How to Sign Code

Signing code is easy; you just use a tool like GPG and create a signature, usually external, so you have `foo.tar.gz` and `foo.tar.gz.asc`. You then provide the signature file, typically by putting it into the same directory or download system as the files. Then, you need to make the signing key available in a secure manner. You can get your key signed, but the chances of getting it signed by widely trusted keys (e.g., vendor distribution keys) are minimal. Your best bet is to make the GPG signing key available on the project website via HTTPS so that users have a reasonable chance of downloading it without an attacker modifying or spoofing it. Simple, right?

Well, you also have to handle key security. The signing key must be as secure as possible, which ideally means placing it on a hardware module that is external to your computer and only plugging it in and using it as needed, thereby minimizing the window of opportunity for an attacker to compromise it.

In a perfect world, your signing server is offline so that attacks cannot be launched against it. The cheapest way to put your signing key into a secure hardware module is to use an OpenPGP-compatible card [3]. These cards are not too expensive; Kernel Concepts sells them for about US$ 20 [4], and you'll also need to buy a reader. You'll also need to choose a key lifespan. Note that keys should expire. This ensures that, say, 20 years from now, when attackers can crack 4096-bit keys on their quantum-enabled smartphones, the key you created cannot be used for nefarious purposes.

Attacks Against Signed Code

So, now the code is signed, and everything is good, right? Nope. Even if packages are signed properly and clients verify the signatures properly (which won't happen unless the process is automated and fails to close when the signature is invalid, as with Yum and RPM), a number of attacks are still possible. The simplest and most direct attack is to provide older signed packages that have known security issues. This attack is especially easy if the file signature doesn't also confirm that the filename has not been changed. Do you ever read `ChangeLog` or `VERSION` to confirm the package version?

Another attack is to prevent any updates of software by deleting new versions, or making downloads incredibly slow, or by feeding infinitely large files to clients. Unfortunately, this approach is more effective than you'd think. For example, when was the last time your update software warned you that no updates had been installed for a few months?

Addressing such attacks is not simple and depends on the client having information available to check. For example, Yum downloads a copy of the repository data and then gets the packages, which prevents replaying of older packages (unless the user forces a downgrade manually) and also lets the client know whether updates should be available. Note, however, that if the attacker can modify the RPMs, chances are they can mangle the repo data as well.

For software that doesn't take this approach, such as WordPress, I suggest keeping a copy of the software that you download so you can compare newer versions to it and make sure that the version you're downloading is actually newer than what you have.

The Update Framework

As with most things in open source, you don't have to reinvent the wheel. The Update Framework (TUF) [5] is available under an open source license (but not one I recognize) and is written mostly in Python, so it's pretty understandable. Even better is that all the above security issues have been taken into account for the system design and implementation. It even addresses possible problems like key compromise.

One basic thing to keep in mind about security, especially for software updates, is that you need to make your system as secure as you can, but you also need to make it possible to return to a known good state (i.e., you have removed all the compromised packages and so on from your update infrastructure).

TUF places all of the heavy lifting on the server and end client. Thus, the intermediary mirror systems don't even need to know about it, which in turn makes deployment possible (trying to get a major mirror site to install some software so they can securely serve updates is a battle you will lose). Unfortunately, TUF isn't perfect – the only client is written in Python, so integrating it with non-Python software or on systems that don't natively support Python (e.g., Windows) will be difficult, to say the least. For more information, an excellent lightning talk is available on YouTube [6] that covers all the basics of TUF using PyPI.

Conclusion

You can certainly use TUF to secure updates, but, unfortunately, deploying TUF is nontrivial. You're going to need at least two servers (in case one fails) and some keys that will require management. That means it's probably not going to be used; in fact, I'm not aware of any software that uses TUF for updates. So, unless people start demanding that organizations and vendors, like WordPress, Drupal, Joomla, RubyGems, PyPI, Hackage, CPAN, and so on, start providing secure updates for all the code they make available, it isn't going to happen. ■■■

INFO

[1] HTP Zine 5: *http://straylig.ht/zines/HTP5/0x02_Linode.txt*

[2] Social Media Widget remote file inclusion: *http://seclists.org/oss-sec/2013/q2/83*

[3] OpenPGP Card: *http://en.wikipedia.org/wiki/OpenPGP_card*

[4] Kernel Concepts Security/Smartcards: *http://shop.kernelconcepts.de/index.php?cPath=1_26*

[5] The Update Framework: *https://www.updateframework.com/*

[6] TUF lightning talk: *https://www.youtube.com/watch?v=2sx1IS6cT3g*