Most software already has a “golden key” backdoor: the system update

Ars Staff ⋮ 9-11 minutes ⋮ 2/27/2016

Software updates are just another term for cryptographic single-points-of-failure.

Leif Ryge is an artist, hacker, and journalist living in Berlin. He used to tweet as @wiretapped but is on hiatus until Twitter stops suspending Tor users' accounts.

In 2014 when *The Washington Post* Editorial Board wrote "with all their wizardry, perhaps Apple and Google could invent a kind of secure golden key they would retain and use only when a court has approved a search warrant," the Internet ridiculed them. Many people painstakingly explained that even if there were somehow wide agreement about who would be the "right" people and governments to hold such an all-powerful capability, it would ultimately be impossible to ensure that such power wouldn't fall in to the "wrong" hands.

Yet, here is a sad joke that happens to describe the reality we presently live in:

> Q: What does almost every piece of software with an update mechanism, including every popular operating system, have in common?
>
> A: Secure golden keys, cryptographic single-points-of-failure which can be used to enable total system compromise via targeted malicious software updates.

I'll define those terms: By "malicious software update," I mean that someone tricks your computer into installing an inauthentic version of some software which causes your computer to do things you don't want it to do. A "targeted malicious software update" means that only the attacker's intended target(s) will receive the update, which greatly decreases the likelihood of anyone ever noticing it. To perform a targeted malicious software update, an attacker needs two things: (1) to be in a position to supply the update and (2) to be able to convince the victim's existing software that the malicious update is authentic. Finally, by "total system compromise" I mean that the attacker obtains all of the authority held by the program they're impersonating an update to. In the case of an operating system, this means that the attacker can subvert any application on that computer and obtain any encryption keys or other unencrypted data that the application has access to.

A backdoored encryption system which allows attackers to decrypt arbitrary data that their targets have encrypted is a significantly different kind of capability than a backdoor which allows attackers to run arbitrary software on their targets' computers. I think many informed people discussing *The Washington Post*'s request for a "secure golden key" assumed they were talking about the former type of backdoor, though it isn't clear to me if the editorial's authors actually understand the difference.

From an attacker perspective, each capability has some advantages. The former allows for passively-collected encrypted communications and other surreptitiously obtained encrypted data to be decrypted. The latter can only be used when the necessary conditions exist for an

active attack to be executed, but when those conditions exist it allows for much more than mere access to already-obtained-but-encrypted data. Any data on the device can be exfiltrated, including encryption keys and new data which can be collected from attached microphones, cameras, or other peripherals.

Many software projects have only begun attempting to verify the authenticity of their updates in recent years. But even among projects that have been trying to do it for decades, most still have single points of devastating failure.

In some systems there are a number of keys where if any one of them is compromised such an attack becomes possible. In other cases it might be that signatures from two or even three keys are necessary, but when those keys are all controlled by the same company (or perhaps even the same person) the system still has single points of failure.

This problem exists in almost every update system in wide use today. Even my favorite operating system, Debian, has this problem. If you use Debian or a Debian derivative like Ubuntu, you can see how many single points of failure you have in your update authenticity mechanism with this command:

```
sudo apt-key list | grep pub | wc -l
```

For the computer I'm writing this on, the answer is nine. When I run the apt-get update command, anyone with any one of those nine keys who is sitting between me and any of the webservers I retrieve updates from could send me malicious software and I will run it as root.

How did we get here? How did so many well-meaning people build so many fragile systems with so many obvious single points of failure?

I believe it was a combination of naivety and hubris. They probably thought they would be able keep the keys safe against realistic attacks, and they didn't consider the possibility that their governments would actually compel them to use their keys to sign malicious updates.

Fortunately, there is some good news. The FBI is presently demonstrating that this was never a good assumption, which finally means that the people who have been saying for a long time that we need to remove these single points of failure can't be dismissed as unreasonably paranoid anymore.

I won't write much about the specifics of the FBI/Apple situation, because there are already plenty of in-depth accounts of the many details of the case. The important thing to understand is that the FBI is demanding that Apple provide them with a signed software update which will disable an iPhone feature which deletes data after a certain number of failed attempts at guessing the PIN (which, along with a per-device secret, is the seed from which the encryption key is derived). On iPhones with relatively short PINs, this effectively "breaks" the encryption because a small key space can be quickly searched.

(On my Debian system, such a feature doesn't even exist. If someone has my encrypted hard drive, they can freely attempt to brute-force my disk passphrase—but hopefully most people's disk crypto passphrases on computers with keyboards are stronger than a short PIN. If an attacker can convince my computer to run arbitrary code while the disk is decrypted, the key can be exfiltrated and the strength of the passphrase becomes irrelevant.)

So when Apple says the FBI is trying to "force us to build a backdoor into our products," what they are really saying is that the FBI is trying to force them to use a backdoor which already exists in their products. (The fact that the FBI is also asking them to write new software is not as relevant, because they could pay somebody else to do that. The thing that Apple can provide which nobody else can is the signature.)

Is it reasonable to describe these single points of failure as backdoors? I think many people might argue that industry-standard systems for ensuring software update authenticity do not qualify as backdoors, perhaps because their existence is not secret or hidden in any way. But in the present Apple case where they are themselves using the word "backdoor," abusing their cryptographic single point of failure is precisely what the FBI is demanding.

Apple might prevail in their current conflict with the FBI, but the fact that they could also lose means they may have already lost to someone else. Imagine if some other murderous criminal organization wanted to access data on a PIN-encrypted iPhone. What if they, like the FBI has now done, found some people who understand how the technology works and figured out who needs to be coerced to make it possible? Having access to a "secure golden key" could be quite dangerous if sufficiently motivated people decide that they want access to it.

I'm optimistic that the demands the FBI is making to Apple will serve as a wakeup call to many of the people responsible for widely-used software distribution infrastructures. I expect that in the not-too-distant future, for many applications at least, attackers wishing to perform targeted malicious updates will be unable to do so without compromising a multitude of keys held by many people in many different legal jurisdictions. There are a number of promising projects which could help achieve that goal, including the DeDiS Cothority and the Docker project's Notary.

Being free of single points of failure should be a basic requirement for any new software distribution mechanisms deployed today.

105 Comments

1.
2.
3.
4.
5.