All.Net Analyst Report and Newsletter

Welcome to our Analyst Report and Newsletter

Supply chain and change control – how to get your Trojan to millions without viruses

I have seen a recent increase in reported incidents involving suspicious or malicious alteration of widely used open source "infrastructure" software. This includes, quite recently, an example "Suspicious Packages Found in Python Package Index (PyPI)", not the only recent one, and I haven't verified anything particular about this one, nor am I particularly upset or concerned about it. I don't use Python for, among other reasons, it syntactic requirements for indentation.

What's going on here?

Those if you who have been watching may notice an increase in the reported volume of such things, and then conclude that we are seeing more of this and should focus our attention on it. Hence this article which I hope will help you defocus a bit. A few points:

- We are seeing more does not mean there are actually more. It may just mean that more of us are spending more time looking there, perhaps some are looking more closely, perhaps some are using better tools.
- The strategy for the attacker is to get to the weak underbelly and leverage it for high volume (industrial scale) deployment of desired (malicious is a matter of perspective) code. That weak underbelly is the large number of highly independent programmers who freely contribute to the open source software that is increasingly widely used and more deeply embedded in more systems and places (a tragedy of the commons). They view themselves and each other as benevolent, and the vast majority of them are, at least in my experience. But a few bad apples injected by bad actors makes the whole bunch potentially go bad, in the eyes of the observer. Millions of sites depend on this software, download it, use it without looking further, and get the Trojans through the natural update process of the software. Interdependencies drive up the volume as more and more people use the underlying software utility and library packages in their applications. The lower in the stack you get in, the more things you effect.
- The tactic is to get in where the getting is easy. Open source utilities and libraries are one of those places today. Lots of contributors, no money involved (just time), no real vetting, no real checking. Do something others find useful and you become more influential, depended on, and embedded. Just join right in. You may even get to start to make changes to other peoples' projects as they start to spend their time doing other things (like having a family or earning a living or dying) and of course you can help these things along if you really want to…
- There is also a larger strategy here. The available resources for getting it right in software are inadequate to the volume of software being built and widely deployed. And the available resources and expertise for cyber-security are inadequate to even attend to such issues on a day-to-day basis. If we start to make some class of attacks or attack mechanisms more visible, the defenders see more of them, start to react to

them, and focus their limited available attention on those issues. This means fewer resources for the things we are really interested in, making it easier for us.

It's about the change control

Supply chain changes effect the downstream mechanisms. The culture of find and patch along with major incidents like Equifax's failure to patch a known vulnerability with a known patch in this cause the rate of patching to increase. Faster and more patching with the same resources means fewer checks along the way, which means more chances for things to go wrong. The people driving faster changes legitimately discuss the time between first discovery of the vulnerability and final elimination at endpoints and identify the gap as the exposure period. But as a community, we are missing something important here.

> **It's the gap between the introduction of the vulnerability and the first discovery that creates the "zero-day" phenomena in the first place.**

Accidental or malicious faults in software produce the failures we see in the field the lead to the need to patch. We "see" the failure in the field, then track it back to the "fault" that produced it, and seek to fix the fault to prevent future failures. But there are lots and lots of faults in software we don't yet see appearing as failures. To learn about this, you might try doing code reviews for a while. Or better yet get involved in program verification. As we see more similar faults (each may produce different sorts of failures), the broader community pays more attention to them and a flurry of research, detection, and patching activity takes place to deal with the "buffer overflow", which then gets embedded in products and services and changes to languages, and so forth. But once the "buffer overflow" problem is reduced to the point where we see less of it, the focus turns to the next proximate cause of failures, perhaps "race conditions" this time.

But here's the thing. These are all symptoms, not the disease. The disease is a lack of adequate process for making sure that what gets out there is good (enough). And that means interdicting the vulnerabilities while they are faults and before they can become failures. This is largely done historically by something known as "**sound change control**".

However...

This process of carefully checking things before release takes earlier resources in the production process, which tend to slow the rapid deployment methodology leading to early entrance into markets and higher market share, which means that is is a losing strategy. It's even worse in the open source contributor-led community because the fast and loose approach to getting something out there and maybe fixing it later is what makes you a hero in that community. Arguably, there are rewards for better code, but they are far outweighed by more code that more people use. A vulnerable piece of code that lots of people use is rewarded a lot more highly than the punishment associated with the possible future discovery of an accidental imperfection that may then be fixed. So it's not going to happen that way.

Conclusion

There is always hope. But I don't hold all that much of it. The fail and fix industrial complex is alive and well, and there are far higher rewards for finding a vulnerability than for not introducing one in the first place. We're not smart enough yet to do preventive medicine. Still, if you want to have systems that work well and almost never have these problems, you should engage in a far better regimen of change control and verification, and do so across the supply

chain, or find ways to prevent such supply chain faults from producing failures in your systems. Sorry to distract you from your pounds of fail and fix for my ounce of prevention.

