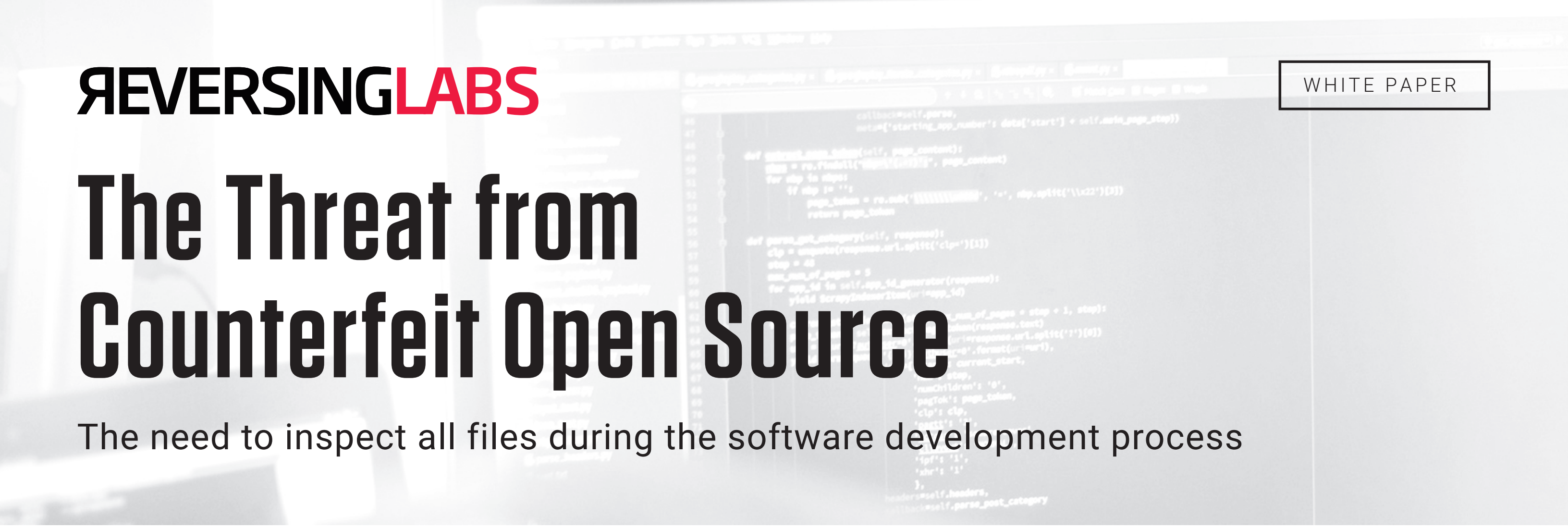


REVERSINGLABS

WHITE PAPER

The Threat from Counterfeit Open Source

The need to inspect all files during the software development process

Open source software is created by talented software developers from around the world who donate their time and intelligence to create software and components, then provide those free of charge to anyone wishing to use them under the terms of their licenses (including the Beerware license). The benefits open source software brings to the world are immeasurable. Open source components accelerate time to market and lower development costs. In a world where software-enabled features are growing and rapid development processes like agile, DevOps, and CI-CD are required, open source is here to stay.

For all its benefits, however, open source can also present risks. Like all software, it can include coding errors that result in vulnerabilities. Most are unintentional, but some can be deliberate.

Attacking the Software Supply Chain

Hackers are rational actors; they want to accomplish their goals using the least amount of effort possible. Attacking proprietary applications can be hard work. Hackers need to conduct research – ideally by obtaining a copy of the software – then attempt to find weaknesses they can exploit.

After years of attacking networks and custom software, enterprising hackers found an easier attack vector and switched to attacking the application development process itself. Even better, attackers need not break into an organization's source repository. Instead, they simply add their malicious code to common open source projects used by organizations and wait for the developers to add the code to proprietary applications themselves.

For years criminals and hackers have used open source to distribute malicious code. Since anyone can create and distribute open source software, criminals can submit updates to well-known packages hoping maintainers will miss the malicious code, or offer to help with continued maintenance of a project. Even more straightforward is to create a project with a plausibly similar project name that is fully under the criminal's control. The package can be entirely bogus or a clone of a well-known project but with select hidden malicious characteristics.

By infecting the Open Source, the responsibility for Breach moves onto Application Developers who are no longer an innocent victim but rather unsuspecting accomplices.

Counterfeit Open Source

If a hacker can successfully add a secret backdoor to a widely used open source project, development organizations will effectively do the hacking for them as they download millions of open source packages from GitHub, PyPI, npm, RubyGems, and other repositories. It is an effective attack vector dating back to at least 2003 when an unknown hacker added a backdoor to the Linux kernel. It is also gaining in popularity. One study found “prototype pollution”, introducing malicious code into otherwise trustworthy components, in over 25% of all open source projects reviewed [1].
If the compromised open source is popular, the attack can affect thousands of applications.

Counterfeit open source is introduced through at least three different attack vectors:

TYPOSQUATTING
Typosquatting is a form of counterfeit open source that occurs when hackers add their malicious code to that of a legitimate file and change the file name slightly to fool developers downloading packages from support sites. On NPM, a hacker added a password recovery tool that would collect and forward credentials to file named bb-builder, hoping users would confuse it with a legitimate file named bb-build. It was downloaded hundreds of times before it was detected by ReversingLabs. Similarly, a hacker contributed a module named “libpeshnx” to popular Python repository PyPI. Its name was similar to a module named "libpeshna" and contained a backdoor intended to allow hackers to take control of devices that used the counterfeit code.

BYPASSING COMMIT CONTROLS
While anyone is free to contribute code to an open source project, only certain individuals can commit code for distribution. “Committers” or “Maintainers” review all contributions to ensure they are appropriate for the project. One way to bypass these controls is to redistribute code through unofficial channels. In one example, hackers added malicious code to Xcode, Apple's official tool for developing iOS and OS X apps, and offered it on a site promising better download speeds in China. The attack compromised over 50 applications offered on the Apple App Store, and affected hundreds of million individual devices when users installed the apps.

ATTACKS ON SOFTWARE DISTRIBUTION NETWORKS
Compromising legitimate software, particularly software with automatic updates, guarantees a widespread breach. In 2017, hackers gained access to CCleaner’s download servers and replaced the legitimate application with one including malicious software, infecting over 20 million users’. Likewise, NotPetya ransomware was originally delivered through the compromised update servers of M.E. Doc software.

[1] Snyk 2020 State of Open Source Security Report

HIDDEN FUNCTIONALITY
Hidden functionality can allow attackers to leverage open source weaknesses. This is typically unintentional, as when a contributor to OpenSSL neglected to validate a variable. This error – which resulted in the Heartbleed vulnerability – went unnoticed for over two years. Hidden functionality can also be from malicious intent; an unknown hacker added a backdoor to Webmin, an open source tool for administering Linux systems, and distributed the compromised software on SourceForge. The backdoor allows a remote attacker to run remote code execution attacks on Webmin installations.

Why Security Defenses Fail

The software development world has changed greatly over the past years. Counterfeit open source leverages changes in how organizations build software to gain a foothold. These include more rapid development and deployment environments and the widespread adoption of open source.

Source Composition Analysis tools only identify components with malicious packages if that component has been added to a vulnerability database. In short, they can only identify malicious code if it is published as a "known vulnerabilities"

The Demand for Rapid Development - As software-enabled features grow in importance, the ability to deliver new features, faster, has made development a competitive differentiator. Agile, DevOps, and Continuous Integration – Continuous Delivery (CI-CD) now dominate new software projects. Rather than delivering a few builds per year, teams may push new builds to production dozens or hundreds of times each day. Traditional security tools that can take hours to perform tests struggle to add value in high velocity environments.

Adoption of Open Source - Organizations in banking, software, government, critical infrastructure, and entertainment have embraced open source software and components. A recent study of over 1,200 commercial applications found open source in 99% of the software, and that on average open source comprised 70% of an application's codebase. In addition, most open source projects are volunteer efforts and therefore cannot afford security scanning tools.

Scarce Security Talent - Demand for security expertise outpaces supply. Research indicates worldwide there are between 3.5 million and 4 million unfilled cybersecurity jobs. Teams struggle to maintain basic items like least privilege controls and security requirements, much less controls for supply chain attacks.

Failing Security Technology - Counterfeit open source is different from normal malware and is not addressed by any traditional security defense. Traditional security testing tools like static and dynamic code defect analysis are designed to identify coding errors that can result in vulnerabilities. They are not capable of conducting deep malware analysis.

Source Composition Analysis (SCA) scanners either parse a build file or scan source and binaries to identify open source components. These tools only identify malicious packages if those have been previously flagged as a "component with known vulnerabilities" and including a Common Vulernabilities and Exposures or CVE designation. Since these components are frequently modified by developers, a missing or misnamed file does not raise any flags. While organizations spend millions looking for suspicious and malicious files in network traffic and email attachments, typosquatting travels through the normal development process in files. Malware scanners do not scan builds.

Stopping Counterfeit Open Source

Identifying evidence of and blocking counterfeit open source requires a different approach from traditional testing tools. Instead of scanning for coding errors or constantly changing hashes of open source components, teams need to look at the contents of the files. This includes:

- **Check for Infected Build and DevOps Systems** - Compilers transform human readable source code into machine readable binaries. Attacks that target compilers, like the W32/Induc-A virus on the Delphi compiler, add malicious code to an application during the build process, subsequently infecting all devices on which the application is installed. Like other viruses, these can be detected by their signature. However, since traditional antivirus software uses hook and inject methods, they will break compile and link processes if installed on a build device.

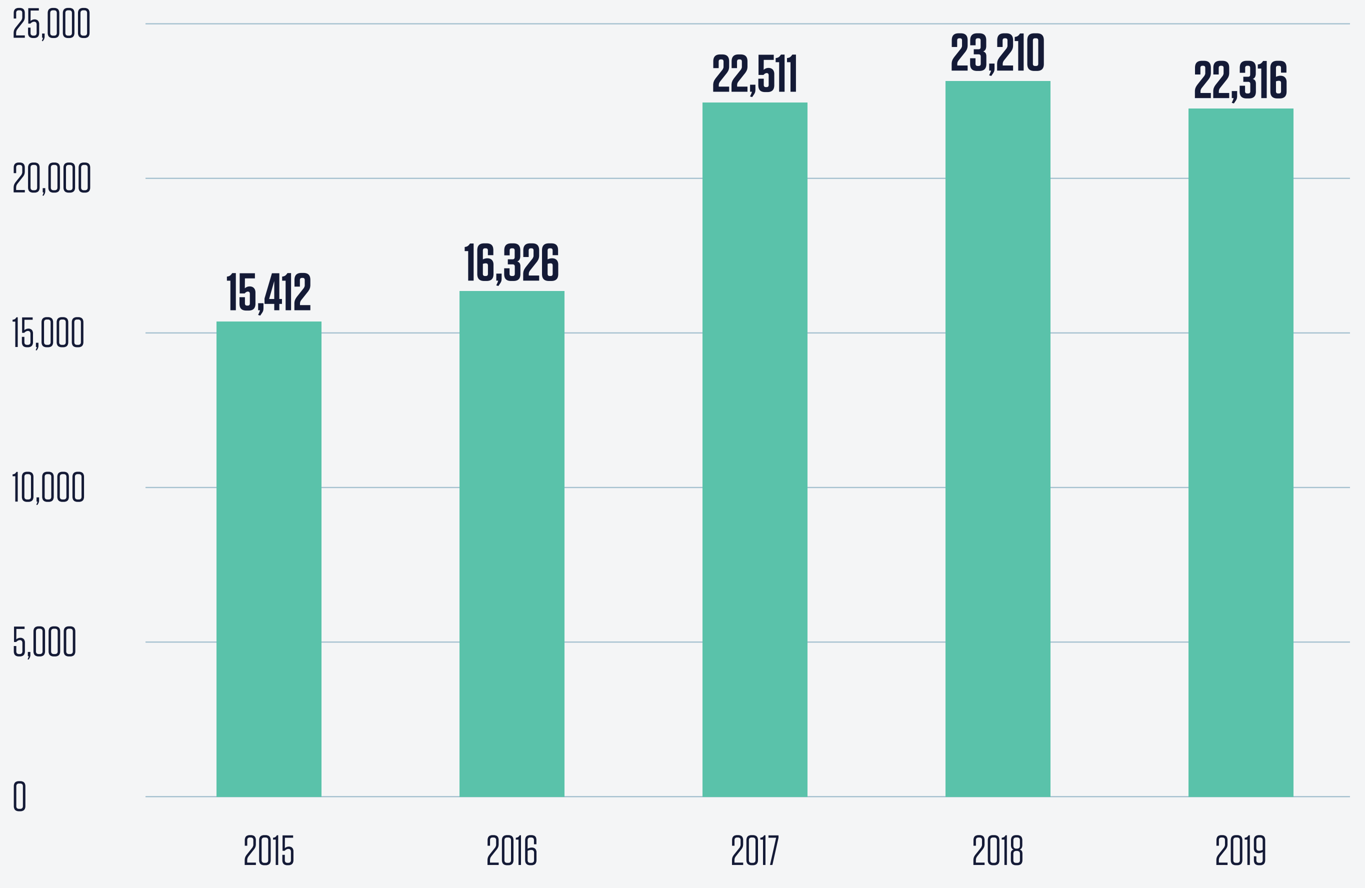
- **Know Where Your Code Originates** - Knowing the trusted provenance of Open Source and its historic maintenance record is critical. One should always attempt to download projects from the main repository, be suspect of pre-built binaries and investigate known forks or similarly sounding projects to identify those with genuine community of users.

- **Examine Components for Malicious Code** - As noted, typosquatting attacks modify legitimate open source libraries with backdoors and malicious code. Recent research by ReversingLabs found over 760 malicious packages in the RubyGems repository. Since open source projects change rapidly, simple hashes for code releases or reliance on package manager parsing to identify "vulnerable" components is insufficient.

  Since attacks and backdoors can be hidden in any type of file, a better approach is to extract, unpack, and examine every file to identify evidence of malicious code and unwanted file types. This includes accidental or deliberate inclusion of web shells, backdoors, coin-mining libraries, or pen testing and IT tools.

- **Block or Patch Legitimate but Vulnerable Open Source** - While not counterfeit software, vulnerable open source components are nonetheless an attractive attack vector; the 2017 Equifax breach resulted from a known vulnerability in Apache Struts. The successful attack was hardly surprising; the OWASP Top 10 list of web application vulnerabilities includes using software with known vulnerabilities. In 2019, over 22,000 new vulnerabilities were disclosed in open source components, and an estimated 30% of the vulnerabilities included a publicly available exploit or proof of concept.

  Organizations can mitigate the risk from vulnerable components by maintaining a complete Software Bill of Materials (SBoM)for each application and mapping those components to public sources of vulnerabilities like the National Vulnerability Database. Creating and examining the SBoM prior to releasing or publishing new code greatly improves an application's security profile. Since new vulnerabilities are disclosed every day, it is critical to continuously monitor sources for the disclosure of new vulnerabilities in components used in internal projects.



VULNERABILITIES DISCLOSED PER YEAR

- **Check Your Certificates** - Digital certificates are generated and authenticated by Certificate Authorities (CA) and used to verify the provenance of software. When a CA is breached, attackers can generate digital certificates themselves and use these to sign malware.

Explainable Threat Intelligence

Counterfeit and vulnerable open source simplify an adversary's job by inserting malicious code or vulnerabilities directly in the target applications. While traditional security testing tools and malware scanners cannot stop this threat, examination of all files during the development process can identify and block backdoors and other malicious code from infecting software. By integrating this into the build and release process, security and development can work together to build more secure software.

To determine the risk profile of any Counterfeit Open Source and build a sound defensive strategy, security teams need field intelligence and an understanding of the tracking and tools used by their adversaries. ReversingLabs' Explainable Threat Intelligence delivers detailed additional information on behavioral characteristics, Indicators of Compromise, and additional decision support data for any potential malware family or malware actor connections.

	ReversingLabs	SCA	SAST	DAST
Identify components with known vulnerabilities	✓	✓	—	—
Examines non-traditional file payloads for malicious code and known malware	✓	—	—	—
Examines all files and identify suspicious IOCs	✓	—	—	—
Examines all build and container files for malware and occlusions	✓	—	—	—
Inspect all crypto certificates for reputation, validity and indicators of compromise	✓	—	—	—

COMPARISON OF REVERSINGLABS, SOURCE COMPOSITION ANALYSIS, STATIC ANALYSIS AND DYNAMIC ANALYSIS

How ReversingLabs Helps

ReversingLabs fills a critical gap in application security testing (AST) tools. Rather than looking for coding errors made by developers (like Static AST or SAST, and Dymanic AST or DAST) or simply the presence of an open source component with a known vulnerability, ReversingLabs quickly and carefully inspects every file in every component to discover malicious code, Indicators of Compromise, and invalid or compromised certificates.

ReversingLabs Titanium Platform works within the Secure Development Lifecycle (SDLC) to identify malicious code and vulnerable components before software is released.

REQUEST A DEMO

© Copyright 2020. ReversingLabs. All rights reserved. ReversingLabs is the registered trademark of ReversingLabs US Inc. All other product and company names mentioned are trademarks or registered trademarks of their respective owners. 2020 August.

REVERSINGLABS
+1.617.250.7518
sales@reversinglabs.com
www.reversinglabs.com