The GitHub Supply Chain Threat: What You Need to Know Today

RESEARCH

Aug. 16, 2022

The GitHub Supply Chain Threat: What You Need to Know Today

If you're a GitHub developer that relies on open source repositories in your code (that would be everyone), Tuesday night's Tweet storm started by Stephen Lacy no doubt caught your attention.

While Lacy quickly walked back his original claim that 35,000 GitHub repositories were infected with malware, the actual issue he brought to the collective attention of software development teams everywhere – the impact a typosquatting attack can have - is something we should all care about. It also points to the larger issue of the vulnerability of software supply chains.

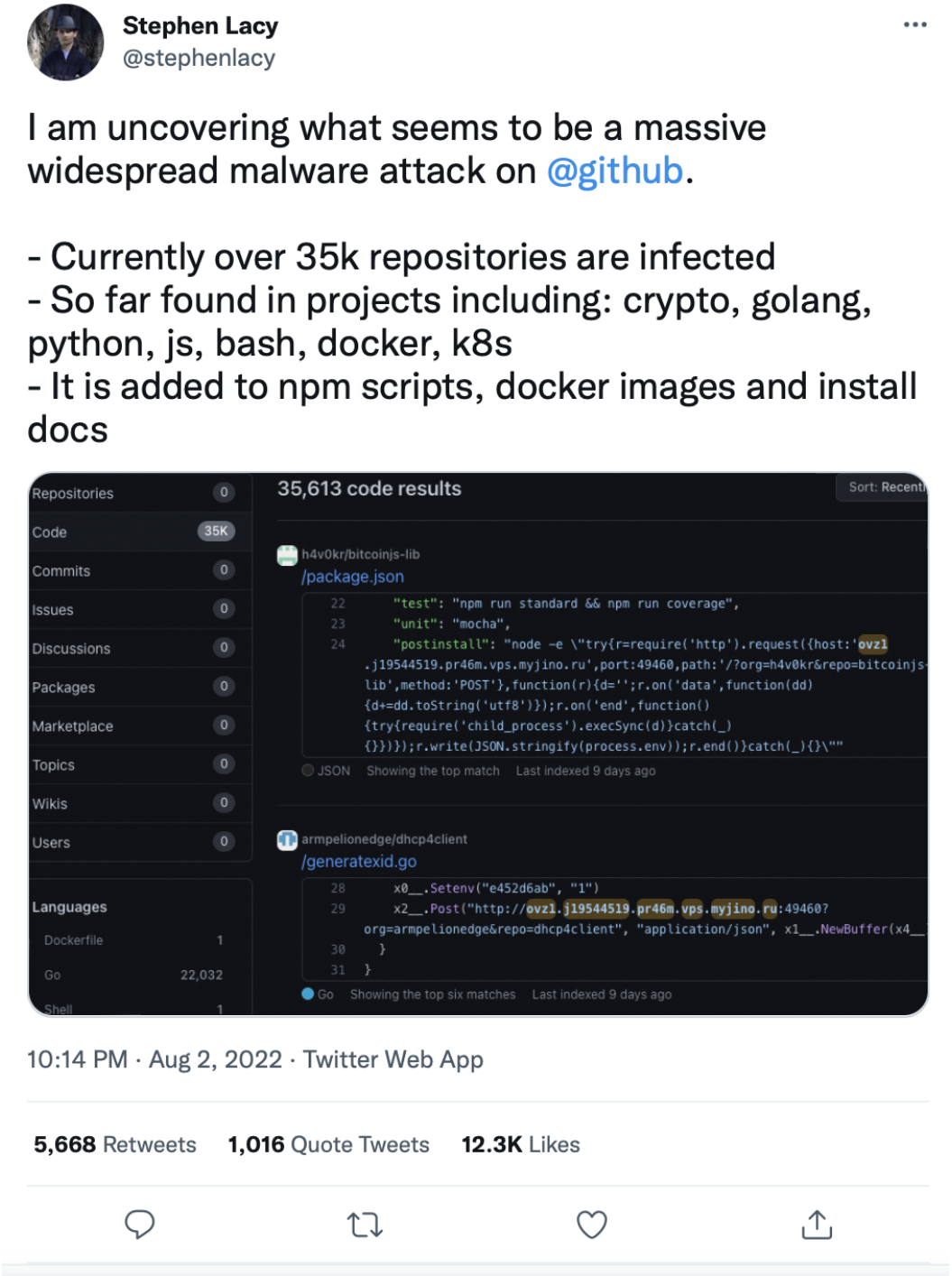
The Typosquatting Threat

Typosquatting is successful because human beings are not infallible creatures - not even developers. The reality is that in the day-to-day world of coding, it is inevitable that a developer will mis-type something in the countless lines of code they write.

Malicious actors know this and prey on the likelihood of such mistakes by creating forks of popular code with nearly identical names. They then infect these misnamed packages with malicious code that can exfiltrate data or conduct remote execution.

An unsuspecting developer can insert these malicious forks into their code with a simple typo in a file name. They are trying to use the original packages, but the typo grabs the malicious code instead.

Typosquatting is, unfortunately, a very popular - and very effective - way for malicious actors to infiltrate apps. It is a point of weakness in the software supply chain, one that relies heavily on human error.



How You Can Protect Your Software Supply Chain

Reducing the likelihood of a typosquatting event is the first step in protecting your apps from software supply chain vulnerabilities. Here are some tips on how to do that.

Vet and curate all packages

To avoid a typosquatting event, it is critical that you examine all the packages you plan to use before they're incorporated into your code. Asking five questions helps determine whether the package is worth the potential risk.

1. **Is the package necessary?** Ask yourself whether this package provides actual value, or if you are just including it because everyone else does.
2. **Is the package documented?** Check GitHub to find out more about the package via a ‘readme’ file. Packages without documentation should raise a red flag.
3. **Is the package tested?** You can find out by checking the Repo for the package.
4. **Is the package maintained?** Any package that hasn’t been updated in more than a year should be cause for further scrutiny.
5. **Is the package respected?** Stick to using packages that other developers currently use and trust.

Asking these questions helps weed out old or maliciously cloned packages before they have a chance to infiltrate your code. Sure you can ask the questions after the fact (when you’re trying to undo damage from malicious code), but you achieve more security savings when you ask the questions beforehand.

Use a proxy server for dependencies

Reduce the risk of developers grabbing malicious code by creating policies around which web packages they can download. Proxy servers give companies the power to create safe lists for developers to use.

These safe lists can be external packages pre-approved by the company or packages written internally by the company. Either way, proxy servers greatly reduce the possibility that developers will inadvertently grab infected packages from external sources.

Implement a scoped namespace

While proxy servers are a great way to reduce external package vulnerability for code, they are not foolproof since they still allow for searching in the public domain. To further reduce the chance that malicious, spoofed code will make its way into your apps, you can mitigate dependency confusion with a scoped namespace.

When you incorporate package scope, you lock the namespace and map it to specific users and organizations. Developers cannot grab spoofed packages, because they are permitted to only grab code from the named domains that you choose.

Leverage the Power of SBOMs

Although not directly related to typosquatting, this strategy strengthens your software supply chain security as a whole. A Software Bill of Materials (SBOM) is a key part of risk management and security for your applications.

When you use an SBOM, you enhance visibility into the software supply chain and can quickly gauge and verify code provenance and the relationships between components. In turn, this helps your teams quickly identify malicious attacks at all stages of the development lifecycle.

The security of the software supply chain is something we take very seriously at Security Journey. We cover it extensively, from our lessons on typosquatting and other common threats to our advanced module on software supply chain security.

The vulnerability Lacy put in the spotlight with his late-night Twitter post is yet another example of how important it is for organizations to understand and take steps to mitigate typosquatting and similar vulnerabilities, while also maintaining focus on the strength and security of their software supply chains overall.

PREVIOUS POST

NEXT POST