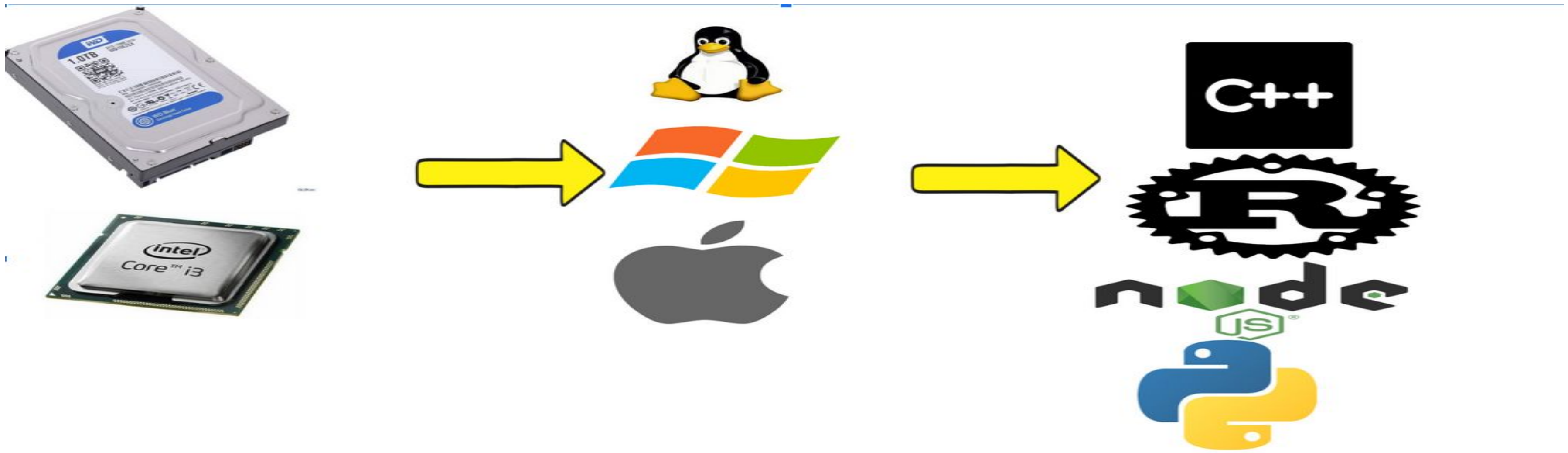
The Software “Supply Chain” and Security

John Allwein - RCOS December 8th, 2021

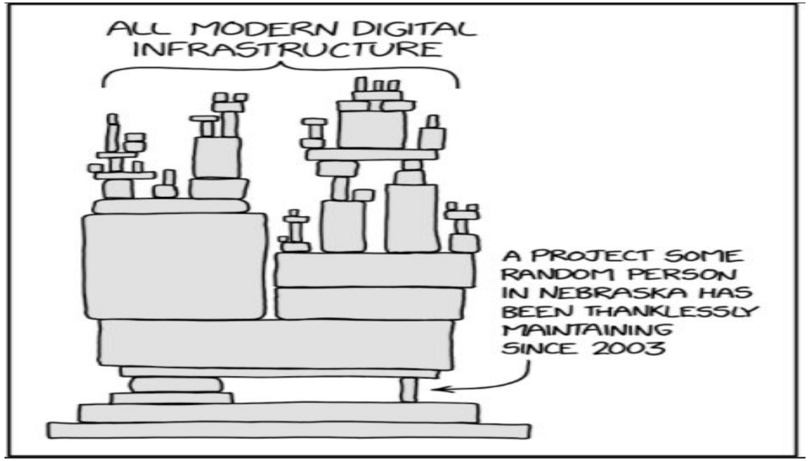
What is the Software Supply Chain?

- At a high level it includes everything from the firmware on the devices you run all the way up to the dependencies of your project
- This talk is largely going to focus on the last category



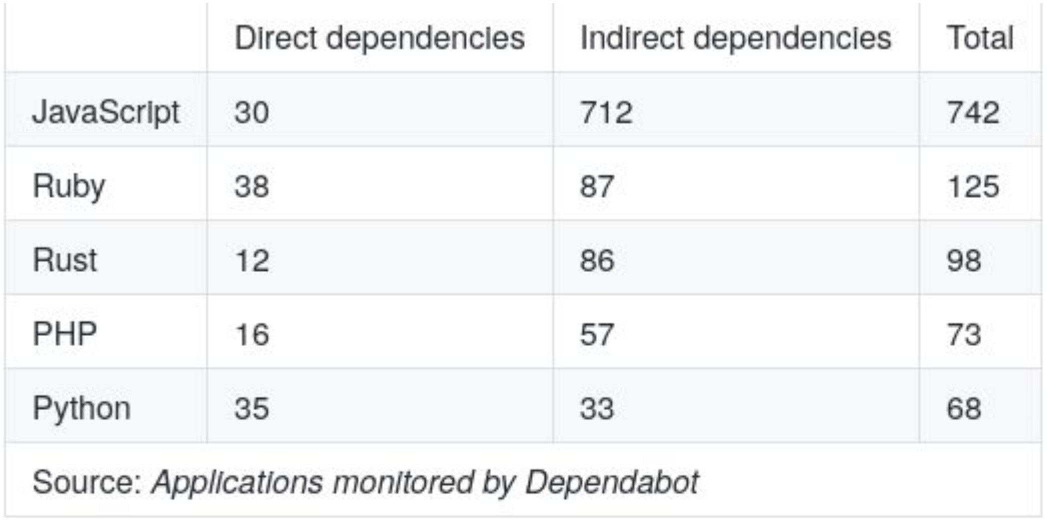
Why should I care about the Supply Chain?

- As always, there's a relevant XKCD.
- https://www.explainxkcd.com/wiki/index.php/2347:_Dependency


ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

That’s a lotta … dependencies

- https://github.blog/2019-01-31-keep-your-dependencies-secure-and-up-to-date-with-github-and-dependabot/
-

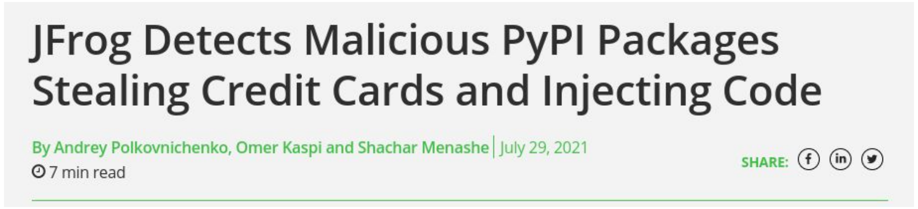


	Direct dependencies	Indirect dependencies	Total
JavaScript	30	712	742
Ruby	38	87	125
Rust	12	86	98
PHP	16	57	73
Python	35	33	68
Source: *Applications monitored by Dependabot*			

Why should I care about the supply chain?

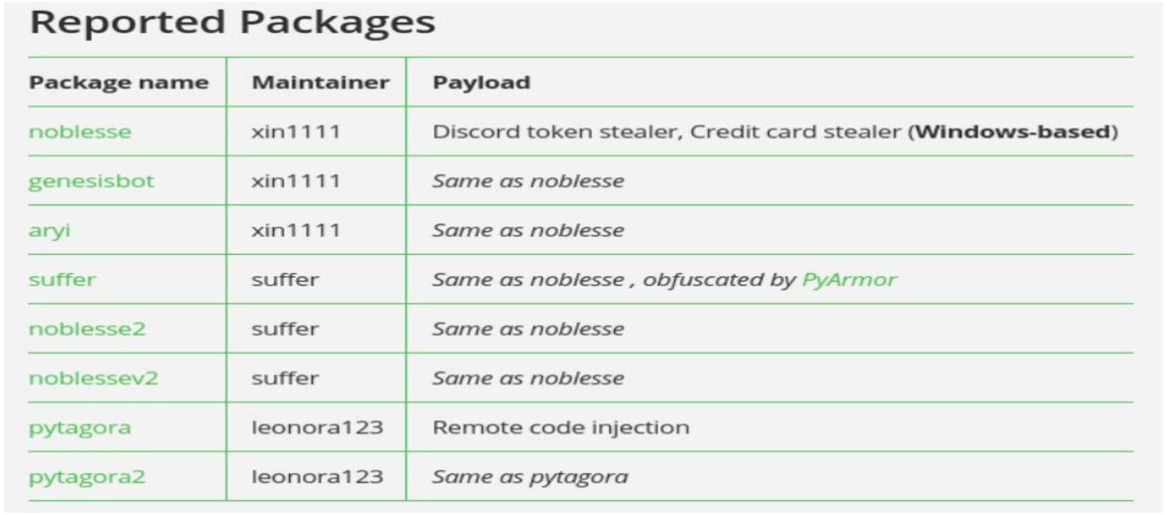
- We all use it
- We all need it
  - Even if everything you write is C with your entire homegrown standard library - who’s auditing your compiler?
- Life’s too short to personally audit all your dependencies
  - We can have hundreds to audit - but it just takes one.
  - Bad actors know this and there’s a rich opportunity to wreak havoc and make a lot of money
- Your users will suffer the consequences of malicious code being injected into your binaries
- We’d (probably) like to be able to build our projects without worrying if some dependency is going to steal credit card data.
- The following slides will contain some recent examples of supply chain or source-based attacks within the past year

Surely these are just a fluke


JFrog Detects Malicious PyPI Packages Stealing Credit Cards and Injecting Code

By Andrey Polkovnichenko, Omer Kaspi and Shachar Menashe | July 29, 2021
7 min read
SHARE:

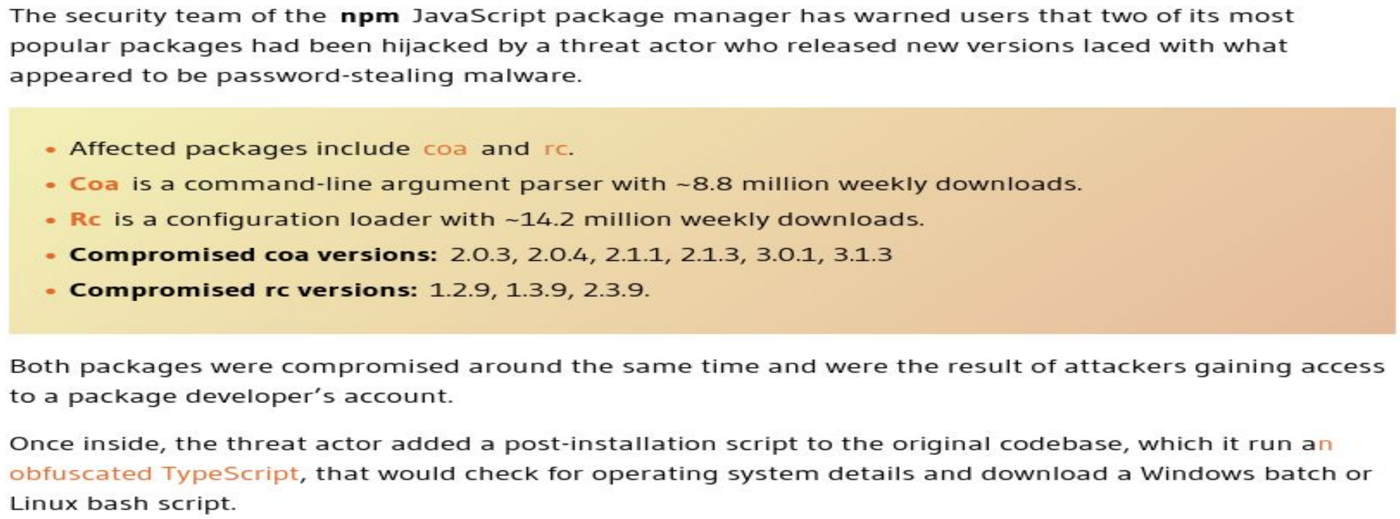
- https://jfrog.com/blog/malicious-pypi-packages-stealing-credit-cards-injecting-code/


Reported Packages

Package name	Maintainer	Payload
noblesse	xin1111	Discord token stealer, Credit card stealer (**Windows-based**)
genesisbot	xin1111	*Same as noblesse*
aryi	xin1111	*Same as noblesse*
suffer	suffer	*Same as noblesse , obfuscated by PyArmor*
noblesse2	suffer	*Same as noblesse*
noblessev2	suffer	*Same as noblesse*
pytagora	leonora123	Remote code injection
pytagora2	leonora123	*Same as pytagora*

Ah, don't run Windows and I'm safe. Got it.

- https://therecord.media/malware-found-in-coa-and-rc-two-npm-packages-with-23m-weekly-downloads/
- November 5, 2021



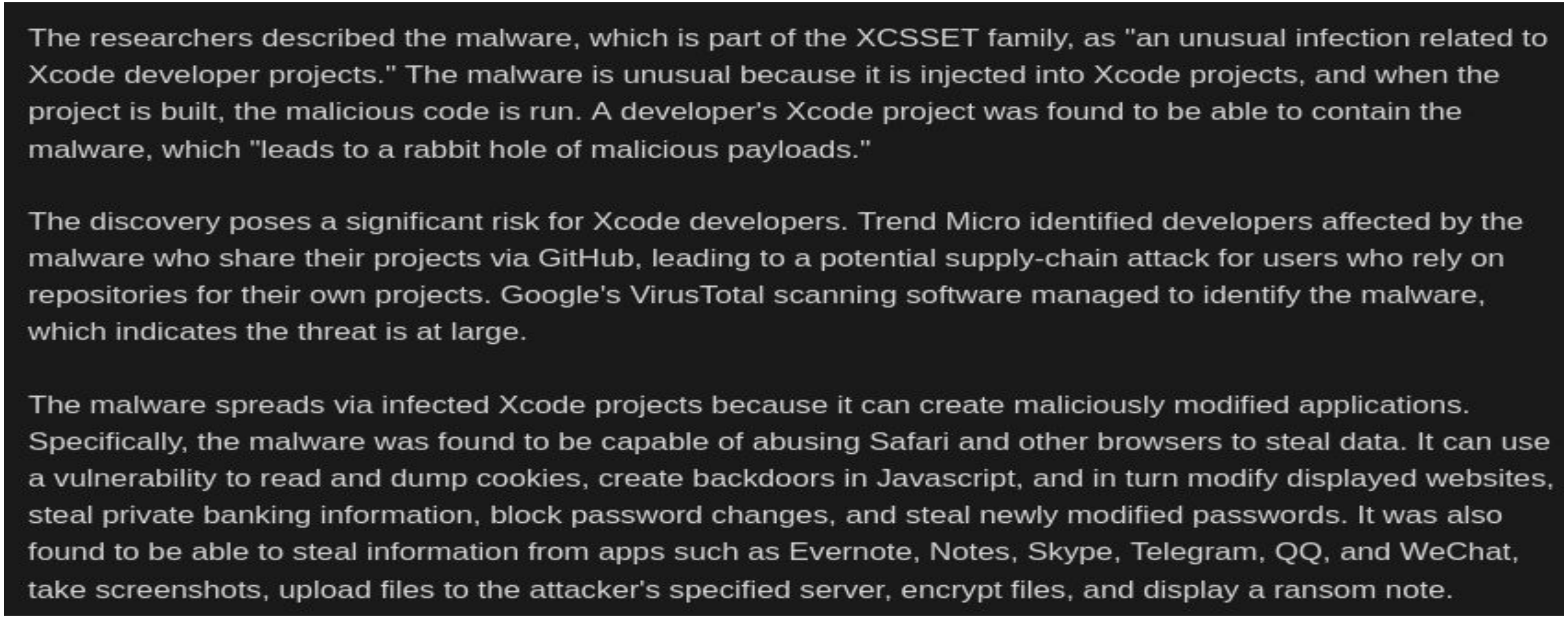
The security team of the **npm** JavaScript package manager has warned users that two of its most popular packages had been hijacked by a threat actor who released new versions laced with what appeared to be password-stealing malware.

- Affected packages include coa and rc.
- Coa is a command-line argument parser with ~8.8 million weekly downloads.
- Rc is a configuration loader with ~14.2 million weekly downloads.
- **Compromised coa versions:** 2.0.3, 2.0.4, 2.1.1, 2.1.3, 3.0.1, 3.1.3
- **Compromised rc versions:** 1.2.9, 1.3.9, 2.3.9.

Both packages were compromised around the same time and were the result of attackers gaining access to a package developer's account.

Once inside, the threat actor added a post-installation script to the original codebase, which it run an obfuscated TypeScript, that would check for operating system details and download a Windows batch or Linux bash script.

I didn’t see Mac OS in there! Where’s my attendance credit?

- https://www.macrumors.com/2020/08/17/new-xcode-malware-found/
- 

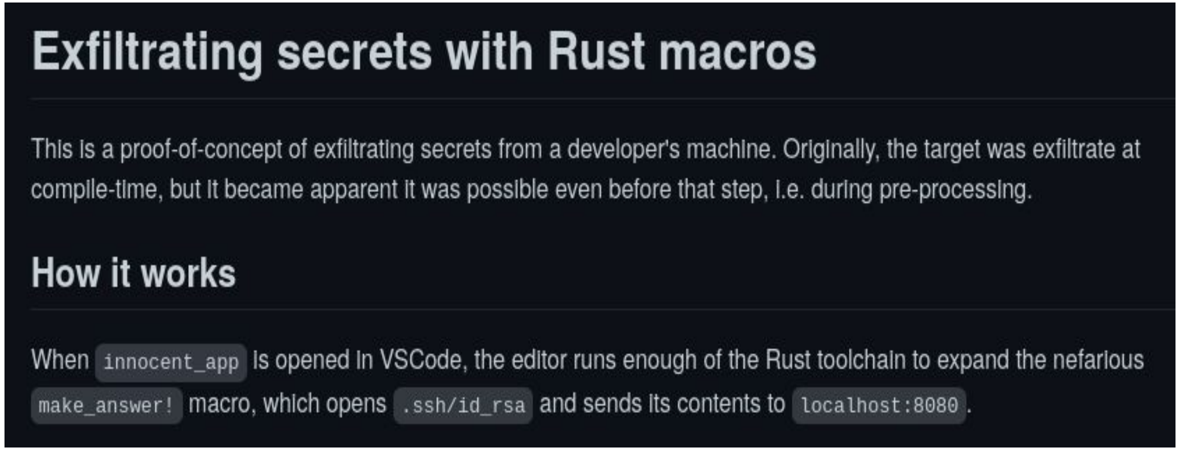
The researchers described the malware, which is part of the XCSSET family, as "an unusual infection related to Xcode developer projects." The malware is unusual because it is injected into Xcode projects, and when the project is built, the malicious code is run. A developer's Xcode project was found to be able to contain the malware, which "leads to a rabbit hole of malicious payloads."

The discovery poses a significant risk for Xcode developers. Trend Micro identified developers affected by the malware who share their projects via GitHub, leading to a potential supply-chain attack for users who rely on repositories for their own projects. Google's VirusTotal scanning software managed to identify the malware, which indicates the threat is at large.

The malware spreads via infected Xcode projects because it can create maliciously modified applications. Specifically, the malware was found to be capable of abusing Safari and other browsers to steal data. It can use a vulnerability to read and dump cookies, create backdoors in Javascript, and in turn modify displayed websites, steal private banking information, block password changes, and steal newly modified passwords. It was also found to be able to steal information from apps such as Evernote, Notes, Skype, Telegram, QQ, and WeChat, take screenshots, upload files to the attacker's specified server, encrypt files, and display a ransom note.

Fine, I'll just run everything in Docker....

- https://github.com/lucky/bad_actor_poc



Exfiltrating secrets with Rust macros

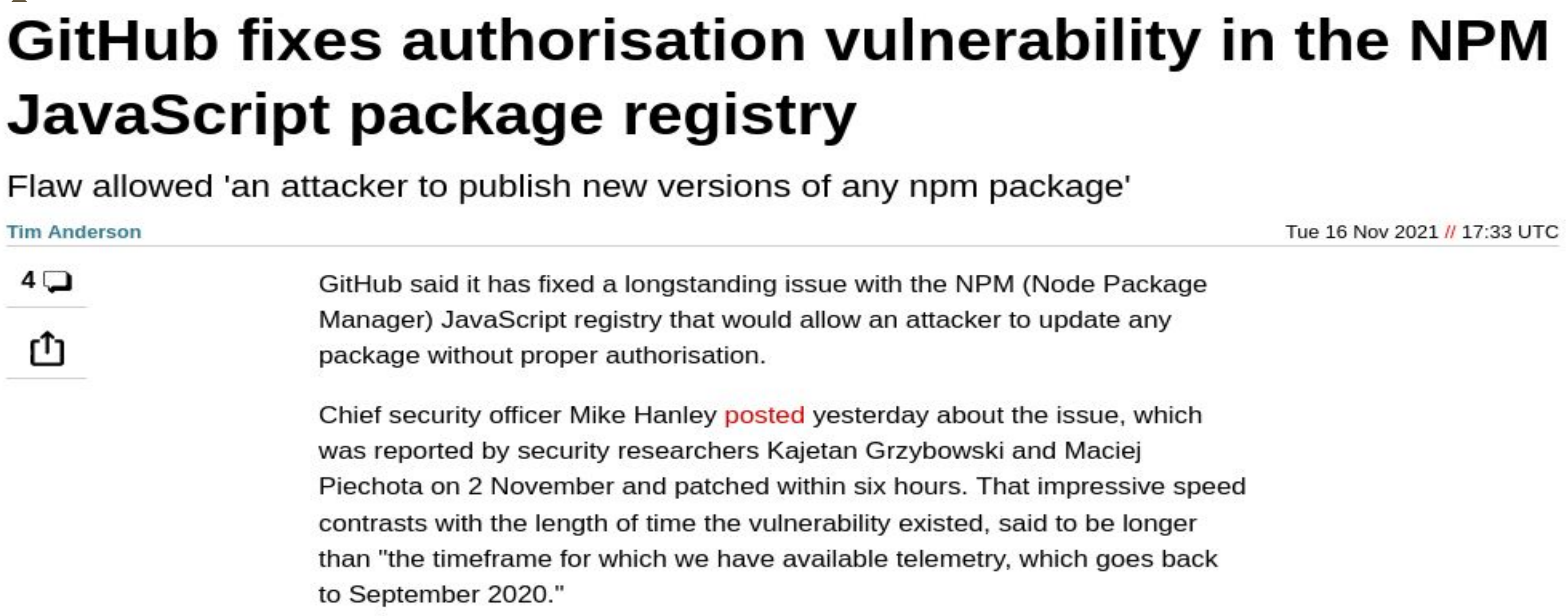
This is a proof-of-concept of exfiltrating secrets from a developer's machine. Originally, the target was exfiltrate at compile-time, but it became apparent it was possible even before that step, i.e. during pre-processing.

How it works

When `innocent_app` is opened in VSCode, the editor runs enough of the Rust toolchain to expand the nefarious `make_answer!` macro, which opens `.ssh/id_rsa` and sends its contents to `localhost:8080`.

I'll only use dependencies by <insert trusted name here>

- https://www.theregister.com/2021/11/16/github_npm_flaw/



GitHub fixes authorisation vulnerability in the NPM JavaScript package registry

Flaw allowed 'an attacker to publish new versions of any npm package'

Tim Anderson — Tue 16 Nov 2021 // 17:33 UTC

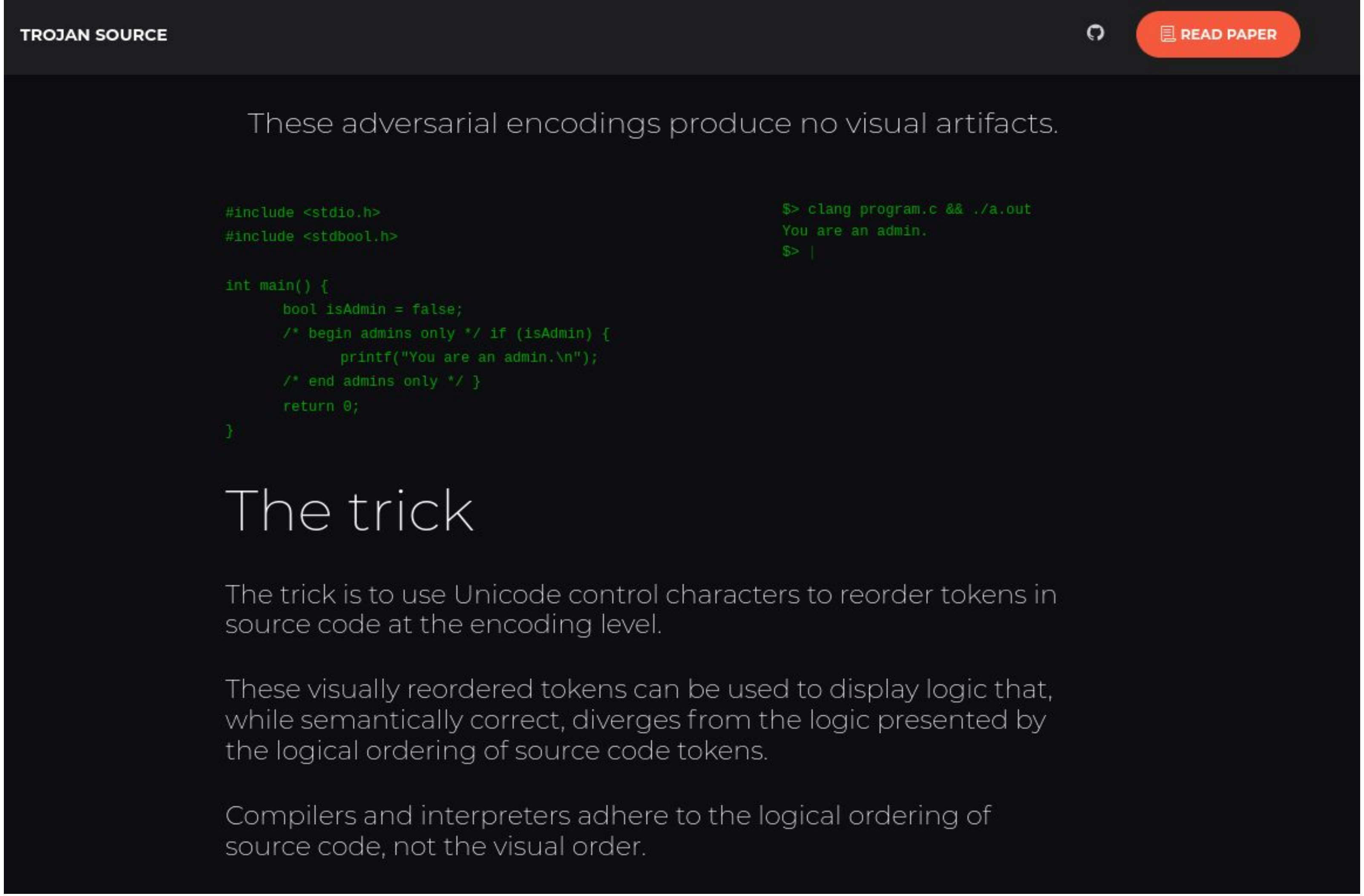
4

GitHub said it has fixed a longstanding issue with the NPM (Node Package Manager) JavaScript registry that would allow an attacker to update any package without proper authorisation.

Chief security officer Mike Hanley posted yesterday about the issue, which was reported by security researchers Kajetan Grzybowski and Maciej Piechota on 2 November and patched within six hours. That impressive speed contrasts with the length of time the vulnerability existed, said to be longer than "the timeframe for which we have available telemetry, which goes back to September 2020."

Fine John, I’ll read every line of every dependency (including transitive dependencies) before building

- I see your move and raise you two CVEs:
  - https://nvd.nist.gov/vuln/detail/CVE-2021-42574
  - https://nvd.nist.gov/vuln/detail/CVE-2021-42694
- Trojan Source Attack (2021)! https://www.trojansource.codes/
- Source code is more than what meets the eye.



These adversarial encodings produce no visual artifacts.

```
#include <stdio.h>
#include <stdbool.h>

int main() {
	bool isAdmin = false;
	/* begin admins only */ if (isAdmin) {
		printf("You are an admin.\n");
	/* end admins only */ }
	return 0;
}
```

```
$> clang program.c && ./a.out
You are an admin.
$> |
```

The trick

The trick is to use Unicode control characters to reorder tokens in source code at the encoding level.

These visually reordered tokens can be used to display logic that, while semantically correct, diverges from the logic presented by the logical ordering of source code tokens.

Compilers and interpreters adhere to the logical ordering of source code, not the visual order.

The variant

A similar attack exists which uses homoglyphs, or characters that appear near identical.

```cpp
#include <iostream>

void sayHello() {
	std::cout << "Hello, World!\n";
}
```

```cpp
void sayΗello() {
	std::cout << "Bye, World!\n";
}
```

The above example defines two distinct functions with near indistinguishable visual differences highlighted for reference.

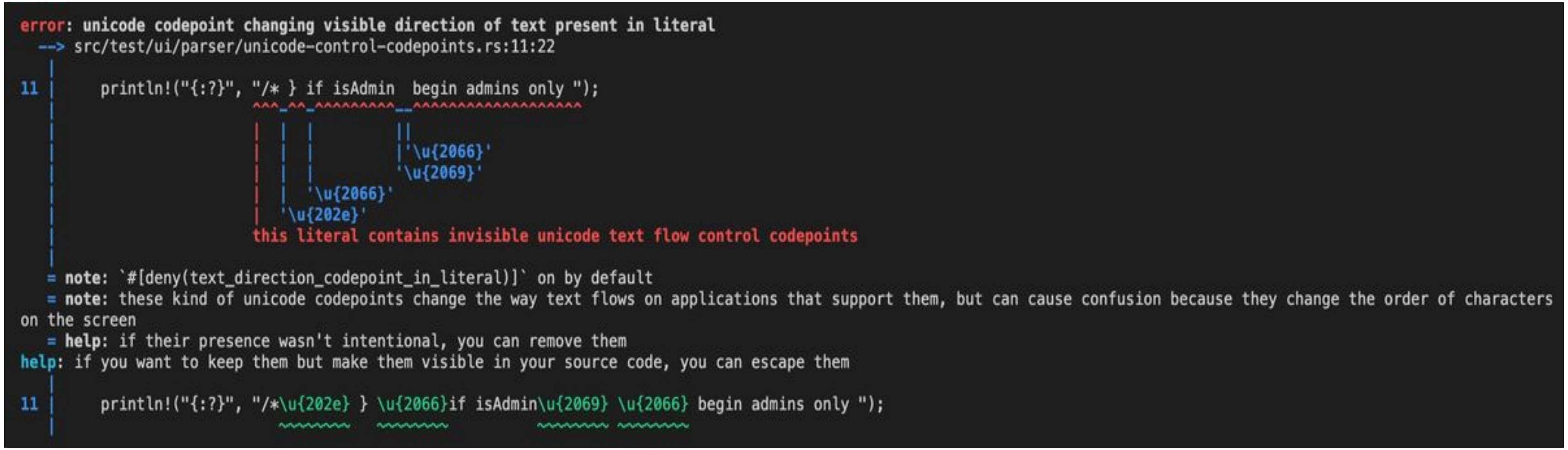
An attacker can define such homoglyph functions in an upstream package imported into the global namespace of the target, which they then call from the victim code.

So what can we do?

- There’s no series of actions you personally can take to avoid being compromised
  - But working together, we can all make the supply chain a safer place :-)
- Starting with the low hanging fruit
  - Multi-factor authentication all the things!
    - Github/Gitlab, npm, crates.io, pypi, etc.
    - Anywhere you’re publishing code and/or binaries should require this
  - Basic password hygiene, don’t re-use passwords, etc.
    - Helps mitigate against services which don’t support MFA
  - Ensure your system remains up-to-date with proper anti-malware software enabled and running

What can language authors do?

- Mitigations to Trojan Source: compiler warnings/errors when suspicious unicode is present

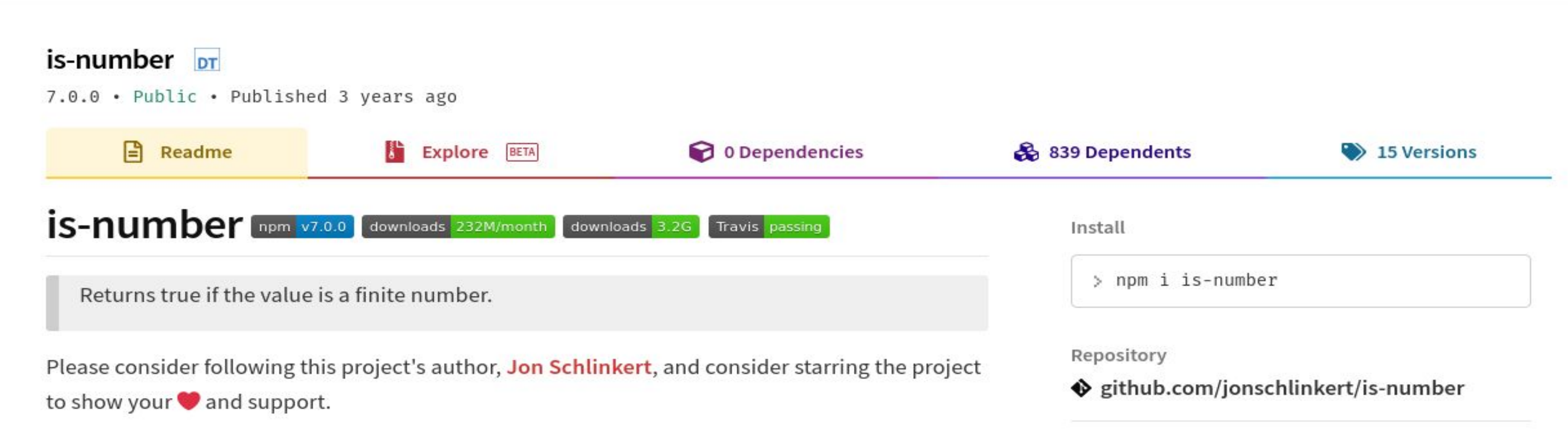


```
error: unicode codepoint changing visible direction of text present in literal
  --> src/test/ui/parser/unicode-control-codepoints.rs:11:22
   |
11 |     println!("{:?}", "/* } if isAdmin  begin admins only ");
   |                       ^^^_^^_^^^^^^^^^^__^^^^^^^^^^^^^^^^^^^
   |                       |  |  |         ||
   |                       |  |  |         |'\u{2066}'
   |                       |  |  |         '\u{2069}'
   |                       |  |  '\u{2066}'
   |                       |  '\u{202e}'
   |                       this literal contains invisible unicode text flow control codepoints
   |
   = note: `#[deny(text_direction_codepoint_in_literal)]` on by default
   = note: these kind of unicode codepoints change the way text flows on applications that support them, but can cause confusion because they change the order of characters
on the screen
   = help: if their presence wasn't intentional, you can remove them
help: if you want to keep them but make them visible in your source code, you can escape them
   |
11 |     println!("{:?}", "/*\u{202e} } \u{2066}if isAdmin\u{2069} \u{2066} begin admins only ");
   |                         ~~~~~~~~   ~~~~~~~~          ~~~~~~~~ ~~~~~~~~
```

- https://github.com/rust-lang/rust/issues/90469

What can language authors do?

- Provide a rich standard library, e.g. the “batteries included” mindset in Python
  - Want to avoid people needing external dependencies for seemingly trivial tasks
  - The screenshot below just shouldn’t be happening



is-number DT

7.0.0 • Public • Published 3 years ago

is-number npm v7.0.0 downloads 232M/month downloads 3.2G Travis passing

> Returns true if the value is a finite number.

Please consider following this project's author, Jon Schlinkert, and consider starring the project to show your ❤️ and support.

Install

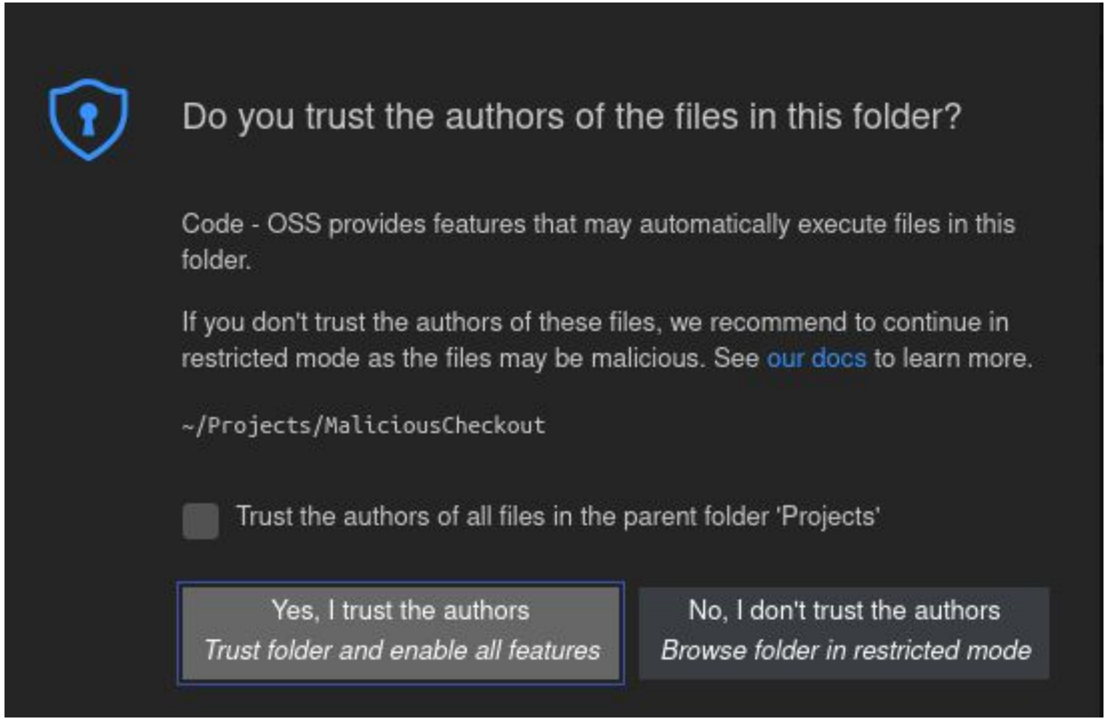
```
> npm i is-number
```

Repository

github.com/jonschlinkert/is-number

Language-adjacent avenues

- This is matter of opinion, but in the case of services like npm, crates.io, pypi, I think it's worth exploring a “verified dependency” system
    - Widely used, critical “infrastructure” packages subject to a manual verification before deployment of new versions
- IDEs/editors should avoid executing any code from simply opening projects
    - Can also integrate checks abuses of Unicode
- VSCode Trusted Workspaces
    - Fancy language server features won’t execute just by opening the workspace
    - Useful for when you just want to browse
    - Combats the Rust macro issue show earlier



Do you trust the authors of the files in this folder?

Code - OSS provides features that may automatically execute files in this folder.

If you don't trust the authors of these files, we recommend to continue in restricted mode as the files may be malicious. See our docs to learn more.

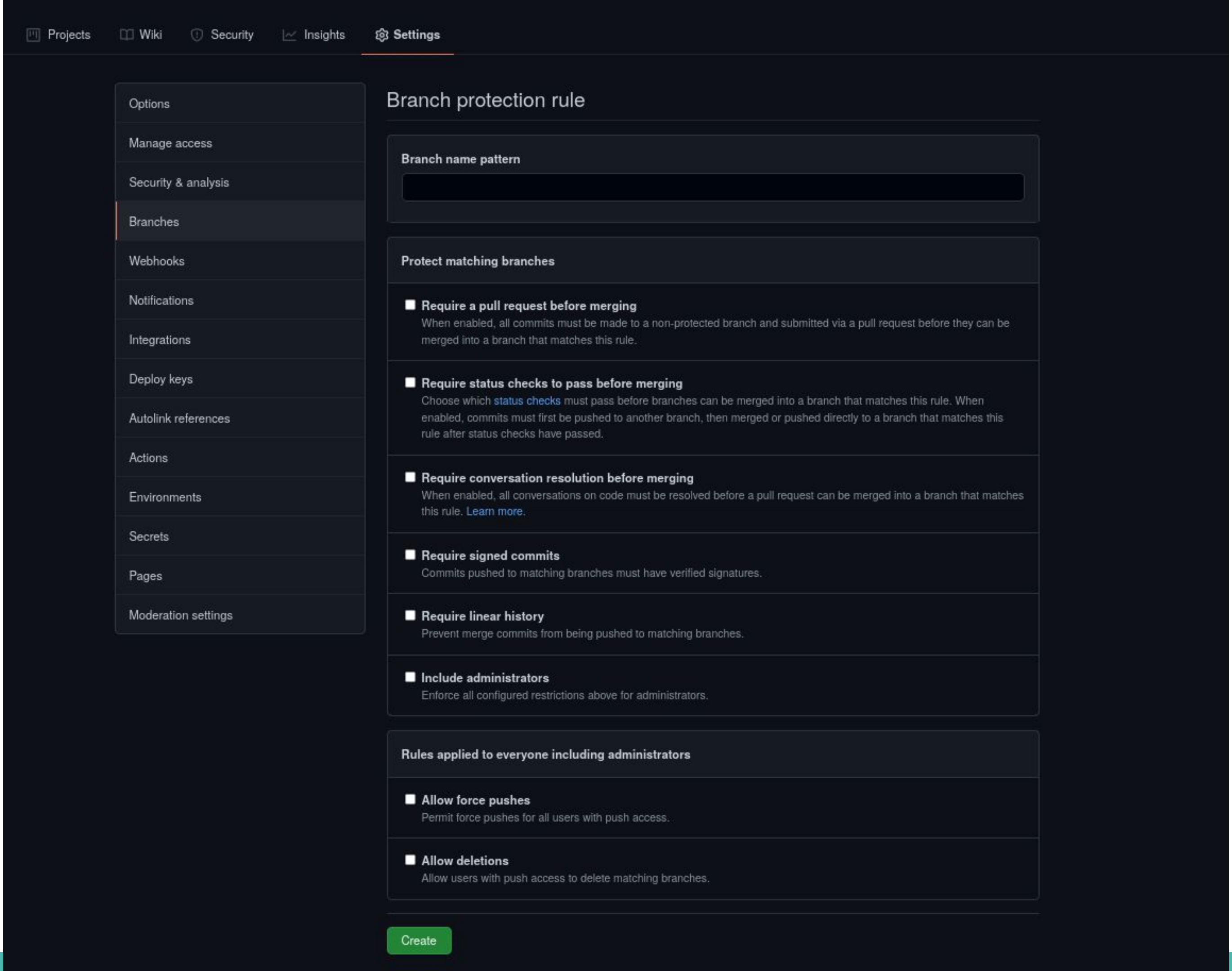
```
~/Projects/MaliciousCheckout
```

☐ Trust the authors of all files in the parent folder 'Projects'

Yes, I trust the authors
Trust folder and enable all features
No, I don't trust the authors
Browse folder in restricted mode

Version Control Safety

- Utilize proper branch protections, notably to avoid force pushes
  - If an attacker compromises someone with sufficiently high permissions to disable this, it won't do a ton of good
  - However, a force push should set off a major red flag that someone may be doing something nefarious by rewriting the history. You don't want to condition your users to ignore this.
- Consider requiring signed commits
  - Git allows people to utilize a PGP signature for cryptographically signing commits. You can then upload you PGP public key to your host (Github/Gitlab) so commits can be verified
  - Commits signed with an unverified key (or a new key) can be flagged as suspicious and potentially compromised
- Don't pin your dependencies against a tag or branch in Git - use commit hashes
  - Tags (and branch state) are mutable. You can can add / delete tags, change which commits they point to, rename them, etc.
  - A compromised account could modify a tag to a malicious commit. Generating a useful hash collision is just not going to happen.
- Consider signing all releases / tags with a verifiable key. This won't stop malicious releases, but your users may notice an unsigned release and refuse to upgrade until authenticity is confirmed
- Don't auto-merge dependency upgrades using Dependabot or similar automated tasks
  - Preferably hold back any non-security related upgrades for several days to a week to enable time for analysis
  - Do a quick inspection of the commit summary just to look for any unusual activity (unverified commits, etc).



Branch protection rule

Branch name pattern

Protect matching branches

- [ ] **Require a pull request before merging**
  When enabled, all commits must be made to a non-protected branch and submitted via a pull request before they can be merged into a branch that matches this rule.
- [ ] **Require status checks to pass before merging**
  Choose which status checks must pass before branches can be merged into a branch that matches this rule. When enabled, commits must first be pushed to another branch, then merged or pushed directly to a branch that matches this rule after status checks have passed.
- [ ] **Require conversation resolution before merging**
  When enabled, all conversations on code must be resolved before a pull request can be merged into a branch that matches this rule. Learn more.
- [ ] **Require signed commits**
  Commits pushed to matching branches must have verified signatures.
- [ ] **Require linear history**
  Prevent merge commits from being pushed to matching branches.
- [ ] **Include administrators**
  Enforce all configured restrictions above for administrators.

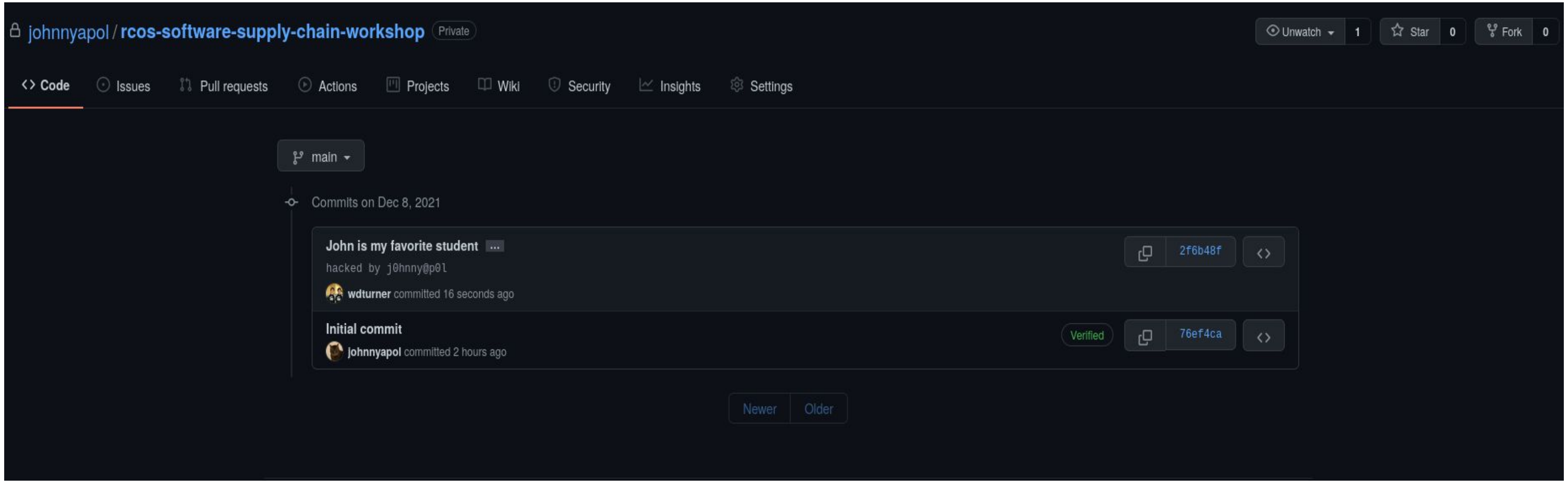
Rules applied to everyone including administrators

- [ ] **Allow force pushes**
  Permit force pushes for all users with push access.
- [ ] **Allow deletions**
  Allow users with push access to delete matching branches.

Otherwise you might have this happen

(forgive me Professor Turner)

https://github.com/johnnyapol/rcos-software-supply-chain-workshop/commit/2f6b48f01656871d07923c722565c767d6edb2b9


johnnyapol / rcos-software-supply-chain-workshop Private
Unwatch 1 Star 0 Fork 0
Code Issues Pull requests Actions Projects Wiki Security Insights Settings
main
Newer Older

Commits on Dec 8, 2021

John is my favorite student ...
hacked by j0hnny@p0l
wdturner committed 16 seconds ago
2f6b48f

Initial commit
johnnyapol committed 2 hours ago
Verified
76ef4ca

Resources for signing + Git + GitHub

- Signing commits- https://docs.github.com/en/authentication/managing-commit-signature-verification/signing-commits
- Signing tags - https://docs.github.com/en/authentication/managing-commit-signature-verification/signing-tags
- Generating a new PGP key (using GPG) - https://docs.github.com/en/authentication/managing-commit-signature-verification/generating-a-new-gpg-key
- Adding the key to GitHub: https://docs.github.com/en/authentication/managing-commit-signature-verification/adding-a-new-gpg-key-to-your-github-account

Reproducible Builds

- A build is reproducible iff. given the same execution environment (compiler versions, dependency versions, source code), the end product (compiled binary) can be independently recreated byte-for-byte
    - C++,C, Rust binaries
    - Python .pyc
    - Java .jars
- Why is this nice?
    - Enables independent verification that the build of your software came from the source code
        - Helps give confidence that binary was not tampered with. This is particularly nice for very large projects (like web browsers), where it is not trivial to build and the build time takes hours. Pretty much everyone uses the compiled binaries for these and we want to make sure they're not a target of malicious interference.

Reproducible Builds

- Nice resource https://reproducible-builds.org/
- Linux distributions are coming a long way in making sure their packages are reproducible. https://reproducible.archlinux.org/ is a nice example.
- Main obstacles to fully reproducible builds
  - Proprietary software can’t be independently reproduced
  - Cryptographic signatures for things like kernels (where modules are built+signed by the distribution) can act as impediments to an independently reproducible build
  - Artifacts in the build, such as timestamps or information about the building machine can cause problems
    - These can usually be worked around by setting up fake environments to reproduce the build conditions. But it is helpful if you provide easy ways to override these in your build system

Alternative avenues with development

- Containers, such as Docker, can help isolate untrusted code from your host system when doing your builds and executing your programs
  - Can never depend on this 100%
  - Probably will stop your credit card from being stolen by a malicious PyPI package, though.
- Virtual Machines can be used for similar reasons as above
- VSCode has a “remote development” feature where your project can live inside a container/VM but you can edit it using VSCode on your regular machine
  - Helps to avoid your regular machine being compromised but affords the convenience of having a native interface
- GitHub codespaces
  - https://github.com/features/codespaces
  - Code compilation, editing, and execution done in the cloud using a vscode-in-the-browser instance

Fin

Thank you everyone for listening - hope I could pass some interesting knowledge onto you.

As this is my last semester at RPI (and thus RCOS), I would like to extend the following thanks to:

- Professor Turner, Professor Kuzmin, and Professor Goldschmidt. Thank you for all your work with RCOS and making it happen.
- Thank you coordinators and mentors!
- Special thank you to Frank for squeezing this workshop in. The semester kinda got away from me.

Happy to take any questions!

Slides will be available on https://github.com/johnnyapol/rcos-software-supply-chain-workshop/