Open-Source Supply Chain Security at Google

Russ Cox (he/him)
ACM SCORED
November 2023

go.dev/s/acmscored
Open-Source Supply Chain Security at Google

Russ Cox (he/him)
ACM SCORED
November 2023

go.dev/s/acmscored
“Supply chain security”

What is a software supply chain?
What does it mean to be secure?
“Supply chain security”

What is a software supply chain?
What does it mean to be secure?

Draft: *Supply chain security* is the engineering of defenses against supply chain attacks.
“Supply chain attack”

What is a supply chain attack?
“Supply chain attack”

What is a supply chain attack?

A (software) supply chain attack is the nefarious alteration of trusted software before delivery.

(tweaking a definition by Kim Zetter)
CIA controlled global encryption company for decades, says report

Swiss government orders inquiry after revelations Crypto AG was owned and operated by US and German intelligence
Novel Malware XcodeGhost Modifies Xcode, Infects Apple iOS Apps and Hits App Store

By Claud Xiao
September 17, 2015 at 4:00 PM
Category: Malware, Threat Prevention, Unit 42
Tags: Apple, Baidu, iOS, KeyRaider, OS X, Weibo, Xcode, XcodeGhost

This post is also available in: 日本語 (Japanese)

UPDATE: Since this report's original posting on September 17, three additional XCodeGhost updates have been published, available here, here and here.

On Wednesday, Chinese iOS developers disclosed a new OS X and iOS malware on Sina Weibo. Alibaba researchers then posted an analysis report on the malware, giving it the name XcodeGhost. We have investigated the malware to identify how it spreads, the techniques it uses and its impact.

XcodeGhost is the first compiler malware in OS X. Its malicious code is located in a Mach-O object file that was repackaged into some versions of Xcode installers. These malicious installers were then uploaded to Baidu's cloud.
Where Did I Leave My Keys?: Lessons from the Juniper Dual EC Incident

By Stephen Checkoway, Jacob Maskiewicz, Christina Garman, Joshua Fried, Shaanan Cohney, Matthew Green, Nadia Heninger, Rafael Philipp Weinmann, Eric Rescorla, Hovav Shacham

Communications of the ACM, November 2018, Vol. 61 No. 11, Pages 148-155
10.1145/3266291

In December 2015, Juniper Networks announced multiple security vulnerabilities stemming from unauthorized code in ScreenOS, the operating system for their NetScreen Virtual Private Network (VPN) routers. The more sophisticated of these vulnerabilities was a passive VPN decryption capability, enabled by a change to one of the parameters used by the Dual Elliptic Curve (EC) pseudorandom number generator.

In this paper, we described the results of a full independent analysis of the ScreenOS randomness and VPN key establishment protocol subsystems, which we carried out in response to this incident. While Dual EC is known to be insecure against an attacker who can choose the elliptic curve parameters, Juniper had claimed in 2013 that ScreenOS included countermeasures against this type of attack. We find that, contrary to Juniper’s public statements, the ScreenOS VPN implementation has been vulnerable to passive exploitation by an attacker who selects parameters in a way that is not secure.
The Untold Story of the Boldest Supply-Chain Hack Ever

The attackers were in thousands of corporate and government networks. They might still be there now. Behind the scenes of the SolarWinds investigation.

ILLUSTRATION: TAMEEM SANKARI

STEVEN ADAIR WASN’T too rattled at first.

It was late 2019, and Adair, the president of the security firm Volexity, was investigating a digital security breach at an American think tank. The intrusion
“Open-source software supply chain attack”

An open-source software supply chain attack is the nefarious alteration of a trusted open-source component before delivery used later in a trusted program.
Widely used open source software contained bitcoin-stealing backdoor

Malicious code that crept into event-stream JavaScript library went undetected for weeks.

DAN GOODIN - 11/26/2018, 5:55 PM

A hacker or hackers sneaked a backdoor into a widely used open source code library with the aim of surreptitiously stealing funds stored in bitcoin wallets, software developers said Monday.

The malicious code was inserted in two stages into event-stream, a code library with 2 million downloads that’s used by Fortune 500 companies and small startups alike. In stage one, version 3.3.6, published on September 8, included a benign module known as flatten-stream. Stage two was implemented on October 5 when flatten-stream was updated to include malicious code that attempted to steal bitcoin wallets and transfer their balances to a server located in Kuala Lumpur. The backdoor came to light last Tuesday with a notification from the open source community.
A deep dive into an NSO zero-click iMessage exploit: Remote Code Execution

Posted by Ian Beer & Samuel Groß of Google Project Zero

*We want to thank Citizen Lab for sharing a sample of the FORCEDENTRY exploit with us, and Apple’s Security Engineering and Architecture (SEAR) group for collaborating with us on the technical analysis. The editorial opinions reflected below are solely Project Zero’s and do not necessarily reflect those of the organizations we collaborated with during this research.*

Earlier this year, Citizen Lab managed to capture an NSO iMessage-based zero-click exploit being used to target a Saudi activist. In this two-part blog post series we will describe for the first time how an in-the-wild zero-click iMessage exploit works.

Based on our research and findings, we assess this to be one of the most technically sophisticated exploits we’ve ever seen, further demonstrating that the capabilities NSO provides rival those previously thought to be accessible to only a handful of nation states.

The vulnerability discussed in this blog post was fixed on September 13, 2021 in iOS 14.8 as CVE-2021-30860.
Hello everyone! Earlier today, we identified a vulnerability in the form of an exploit within Log4j – a common Java logging library. This exploit affects many services – including Minecraft Java Edition.

This vulnerability poses a potential risk of your computer being compromised, and while this exploit has been addressed with all versions of the game client patched, you still need to take the following steps to secure your game and your servers.
“Open-source supply chain vulnerability”

An open source supply chain vulnerability is an exploitable weakness in trusted software caused by an open source component.
“Open-source software supply chain vulnerability”

An open source supply chain vulnerability is an exploitable weakness in a trusted software package caused by one of that package’s open source components.

- Trusted software need not be open-source (Minecraft is not).
“Open-source supply chain security”

Earlier draft:
Supply chain security is the engineering of defenses against supply chain attacks.

Open source supply chain security is the engineering of defenses against open source supply chain attacks and open source supply chain vulnerabilities.
Open-Source Supply Chain Security at Google

Disclaimers
- Not exhaustive about efforts at Google.
- No intent to discount work being done elsewhere. Just reporting about Google.
Open-source software supply chain security at Google

Three main approaches:

- **Understanding** the supply chain
- **Strengthening** the supply chain
- **Monitoring** the supply chain
Understanding the software supply chain

(The software supply chain is all the places where a supply chain attack might happen.)
Understanding the software supply chain

Build Graph

go.dev/s/acmscored
Understanding the software supply chain

Server Graph (~2020, outdated)

[Diagram of the software supply chain, showing various components such as Go releases, server graph, Go+BoringCrypto releases, and VSCode Go extension.]
Understanding the software supply chain

Dependency Graph
Understanding the software supply chain

Dependency Graph

Go module
k8s.io/kubernetes v1.28.4

Licenses
- Apache-2.0

Security Advisories
- github.com/s磐/depth-securejoin v0.2.3
  Security: on Windows, paths outside of the roots could be inadvertently produced
  Path outside of the root could be produced on Windows
- go.opentelemetry.io/contrib/instrumentation/github.com/emickle/go-restful/opentelemetry-v0.35.0
  OpenTelemetry-Go Contrib vulnerable to denial of service in solrhttp due to unbounded cardinality metrics
  Memory exhaustion in github.com/opentelemetry/opentelemetry-go-contrib

Go.dev/s/acmscored
Understanding the software supply chain
Strengthening the software supply chain

> Defend against attacks
Find vulnerabilities
Cryptographic signatures make it impossible to nefariously alter code between signing and verifying.

Removes download infrastructure, hosting, network middleboxes as potential attack sites.

Introduces key distribution problems.
Strengthening the software supply chain > Attacks

Go Checksum Database

Map from (module, version) -> SHA256 of file tree
Signed by private key; public key hard-coded in Go distribution

Every download of public module
checks (possibly cached) checksum database entry.

Checksum database assumes first observed copy of code is “correct”.
Makes (module, version) -> code mapping immutable.
Strengthening the software supply chain > Attacks

Sigstore

sign. verify. protect.

Making sure your software is what it claims to be.
Strengthening the software supply chain > Attacks

Sigstore
Strengthening the software supply chain > Attacks

Computer System Security

All these boxes need to be secured too.

Dedicated build systems can provide better security, reproducibility: Google Cloud Build, GitHub Actions.

(But engineering workstations, laptops need security too.)
Perfectly Reproducible, Verified Go Toolchains

Russ Cox
28 August 2023

One of the key benefits of open-source software is that anyone can read the source code and inspect what it does. And yet most software, even open-source software, is downloaded in the form of compiled binaries, which are much more difficult to inspect. If an attacker wanted to run a supply chain attack on an open-source project, the least visible way would be to replace the binaries being served while leaving the source code unmodified.

The best way to address this kind of attack is to make open-source software builds reproducible, meaning that a build that starts with the same sources produces the same outputs every time it runs. That way, anyone can verify that posted binaries are free of hidden changes by building from authentic sources and checking that the rebuilt binaries are bit-for-bit identical to the posted binaries. That approach proves the binaries have no backdoors or other changes not present in the source code, without having to disassemble or look inside them at all. Since anyone can verify the binaries, independent groups can easily detect and report supply chain attacks.
Strengthening the software supply chain > Attacks
Two-Person Approvals
## Supply-chain Levels for Software Artifacts (SLSA)

<table>
<thead>
<tr>
<th>Track/Level</th>
<th>Requirements</th>
<th>Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build L0</td>
<td>(none)</td>
<td>(n/a)</td>
</tr>
<tr>
<td>Build L1</td>
<td>Provenance showing how the package was built</td>
<td>Mistakes, documentation</td>
</tr>
<tr>
<td>Build L2</td>
<td>Signed provenance, generated by a hosted build platform</td>
<td>Tampering after the build</td>
</tr>
<tr>
<td>Build L3</td>
<td>Hardened build platform</td>
<td>Tampering during the build</td>
</tr>
</tbody>
</table>
Strengthening the software supply chain > Attacks

OpenSSF Security Scorecards

What is OpenSSF Scorecard?

Scorecard assesses open source projects for security risks through a series of automated checks

It was created by OSS developers to help improve the health of critical projects that the community depends on.

You can use it to proactively assess and make informed decisions about accepting security risks within your codebase. You can also use the tool to evaluate other projects and dependencies, and work with maintainers to improve codebases you might want to integrate.

Scorecard helps you enforce best practices that can guard against:

- Malicious maintainers
- Build system compromises
- Source code compromises
- Malicious packages
Capslock

% capslock -output=verbose
Capslock is an experimental tool for static analysis of Go packages.

Analyzed packages:
golang.org/x/text v0.9.0

CAPABILITY_READ_SYSTEM_STATE: 2 references (2 direct, 0 transitive)
Example callpath:
  rsc.io/sampler.Hello
sampler.go:32:27:rsc.io/sampler.DefaultUserPrefs

CAPABILITY_UNANALYZED: 5 references (0 direct, 5 transitive)
Example callpath:
  rsc.io/sampler.DefaultUserPrefs
language.go:44:21:(golang.org/x/text/language.CanonType).Make
language.go:50:17:(golang.org/x/text/language.CanonType).Parse
parse.go:58:29:golang.org/x/text/language.canonicalize
lookup.go:50:17:golang.org/x/text/internal/language.normLang
lookup.go:55:18:sort.Search
Capslock Limitations

Code operating within its capabilities can still be nefarious.

- String comparison in a password checker.
- YAML parser in a production system.
- XcodeGhost
Strengthening the software supply chain

Defend against attacks
> Find vulnerabilities

go.dev/s/acmscored
Strengthening the software supply chain > Vulnerabilities

OSS Fuzz

Announcing OSS-Fuzz: Continuous fuzzing for open source software

Thursday, December 1, 2016

We are happy to announce OSS-Fuzz, a new Beta program developed over the past years with the Core Infrastructure Initiative community. This program will provide continuous fuzzing for select core open source software.

Open source software is the backbone of the many apps, sites, services, and networked things that make up “the internet.” It is important that the open source foundation be stable, secure, and reliable, as cracks and weaknesses impact all who build on it.

Recent security stories confirm that errors like buffer overflow and use-after-free can have serious, widespread consequences when they occur in critical open source software. These errors are not only serious, but notoriously difficult to find via routine code audits, even for experienced developers. That’s where fuzz testing comes in. By generating random inputs to a given program, fuzzing triggers and helps uncover errors quickly and thoroughly.
Strengthening the software supply chain > Vulnerabilities

Syzkaller

<table>
<thead>
<tr>
<th>Title</th>
<th>Repro</th>
<th>Cause</th>
<th>Fix</th>
<th>Count</th>
<th>Last</th>
<th>Reported</th>
<th>Discussions</th>
</tr>
</thead>
<tbody>
<tr>
<td>WARNING in _cfg80211 bound to bss_update</td>
<td>wireless</td>
<td></td>
<td></td>
<td>1</td>
<td>4d18h</td>
<td>18h16m</td>
<td>0 [18h16m]</td>
</tr>
<tr>
<td>possible deadlock in stack_depot_put</td>
<td>kernel</td>
<td></td>
<td></td>
<td>7</td>
<td>1d17h</td>
<td>2d17h</td>
<td>3 [1d05h]</td>
</tr>
<tr>
<td>general protection fault in bifs_get_block (2)</td>
<td>bifs</td>
<td>C</td>
<td>error</td>
<td>1</td>
<td>7d00h</td>
<td>2d23h</td>
<td>PATCH [1d14h]</td>
</tr>
<tr>
<td>INFO: task hung in hwmp_fillin</td>
<td>crypto</td>
<td>C</td>
<td>error</td>
<td>8</td>
<td>3d09h</td>
<td>3d08h</td>
<td>0 [2d08h]</td>
</tr>
<tr>
<td>kernel BUG in ext4_mb_release_inode_pr</td>
<td>ext4</td>
<td>syz</td>
<td>unreliable</td>
<td>1</td>
<td>7d03h</td>
<td>3d03h</td>
<td>0 [3d03h]</td>
</tr>
<tr>
<td>memory leak in j1939_netdev_start</td>
<td>can</td>
<td>syz</td>
<td></td>
<td>1</td>
<td>7d11h</td>
<td>3d11h</td>
<td>0 [3d11h]</td>
</tr>
<tr>
<td>memory leak in clear_state_bit</td>
<td>btrfs</td>
<td>C</td>
<td></td>
<td>3</td>
<td>7d09h</td>
<td>4d10h</td>
<td>0 [4d10h]</td>
</tr>
<tr>
<td>WARNING in inode_insert_into_buffer</td>
<td>nfts3</td>
<td>C</td>
<td></td>
<td>2</td>
<td>8d20h</td>
<td>4d21h</td>
<td>0 [3d01h]</td>
</tr>
<tr>
<td>go runtime error</td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td>2d18h</td>
<td>4d22h</td>
<td>0 [4d22h]</td>
</tr>
<tr>
<td>KASAN: slab-use-after-free Read in j1939_netdev_start</td>
<td>bluetooth</td>
<td>C</td>
<td>done</td>
<td>1</td>
<td>5d23h</td>
<td>5d22h</td>
<td>0 [5d13h]</td>
</tr>
<tr>
<td>kernel BUG in entry_points_to_object</td>
<td>reiserfs</td>
<td>C</td>
<td>done</td>
<td>3</td>
<td>2d14h</td>
<td>5d22h</td>
<td>0 [5d22h]</td>
</tr>
<tr>
<td>WARNING in ext4 dio write end io</td>
<td>ext4</td>
<td>C</td>
<td>done</td>
<td>2</td>
<td>6d20h</td>
<td>5d23h</td>
<td>PATCH [5d84h]</td>
</tr>
<tr>
<td>general protection fault in j1939_netdev_connect</td>
<td>kernel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>possible deadlock in nfts_set_size</td>
<td>nfts3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WARNING in format_decode (3)</td>
<td>bpf</td>
<td>C</td>
<td>done</td>
<td>65</td>
<td>4d07h</td>
<td>6d04h</td>
<td>6 [5d01h]</td>
</tr>
<tr>
<td>KASAN: slab-use-after-free Read in kill_orphaned_pgrp</td>
<td>kernel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WARNING in reiserfs_ioctl (2)</td>
<td>reiserfs</td>
<td></td>
<td></td>
<td>1</td>
<td>7d6d</td>
<td>6d23h</td>
<td>1 [6d08h]</td>
</tr>
<tr>
<td>WARNING in reiserfs_ioctl (2)</td>
<td>reiserfs</td>
<td></td>
<td></td>
<td>1</td>
<td>12d</td>
<td>8d01h</td>
<td>0 [8d01h]</td>
</tr>
<tr>
<td>memory leak in btrfs_add_free_space</td>
<td>btrfs</td>
<td>syz</td>
<td></td>
<td>1</td>
<td>12d</td>
<td>8d06h</td>
<td>0 [8d06h]</td>
</tr>
<tr>
<td>memory leak in r8712_init_xmit_priv (2)</td>
<td>usb</td>
<td>C</td>
<td></td>
<td>2</td>
<td>7d05h</td>
<td>8d14h</td>
<td>0 [4d03h]</td>
</tr>
<tr>
<td>BUG: unable to handle kernel paging request in copy_from_kernel_page</td>
<td>C</td>
<td>done</td>
<td></td>
<td>4</td>
<td>12d</td>
<td>8d28h</td>
<td>1 [6d21h]</td>
</tr>
</tbody>
</table>
"It has raised the public awareness to a considerable degree. It is likely to make people more careful and more attentive to vulnerabilities in the future."

– Robert H. Morris,
quoted in the next day’s paper
Strengthening the software supply chain > Safe Languages

Memory-Safe Languages

Software Memory Safety

Executive summary

Modern society relies heavily on software developers to write software that is not only functional but also secure. However, software vulnerabilities are still frequent, and organizations are often compromised by malicious actors. It is critical to prepare the logic in software to mitigate the exploitation of vulnerabilities.

The path forward

Memory issues in software comprise a large portion of the exploitable vulnerabilities in existence. NSA advises organizations to consider making a strategic shift from programming languages that provide little or no inherent memory protection, such as C/C++, to a memory safe language when possible. Some examples of memory safe languages include C#, Go, Java, Ruby™, and Swift®. These languages perform rigorous testing to ensure exploitable software memory issues are eliminated. Examples include

[go.dev/s/acmscored]
Software Memory Safety

Executive summary

Modern society relies heavily on software, which can be compromised for malicious reasons. It is crucial to prepare the logic in software so that vulnerabilities are still frequent. The path forward

Memory issues in software comprise a large portion of the exploitable vulnerabilities in existence. NSA advises organizations to consider making a strategic shift from programming languages that provide little or no inherent memory protection, such as C/C++, to a memory safe language when possible. Some examples of memory safe languages are C#, Go, Java, Ruby™, and Swift®. Memory safe languages provide formal rigorous testing to exploit software and memory issues. Examples include
Strengthening the software supply chain > Safe Languages

Memory-Safe Languages

Build simple, secure, scalable systems with Go

- An open-source programming language supported by Google
- Easy to learn and great for teams
- Built-in concurrency and a robust standard library
- Large ecosystem of partners, communities, and tools

Get Started  Download

Download packages for Windows 64-bit, macOS, Linux, and more

The go command by default downloads and authenticates modules using the Go module mirror and Go checksum database run by Google. Learn more.

Companies using Go

Organizations in every industry use Go to power their software and services  View all stories
Strengthening the software supply chain > Safe Languages

Memory-Safe Languages

Rust

A language empowering everyone to build reliable and efficient software.

Why Rust?

- **Performance**: Rust is blazingly fast and memory-efficient; with no runtime or garbage collection overhead.
- **Reliability**: Rust’s rich type system and ownership model guarantee memory-safety and avoid many classes of runtime errors.
- **Productivity**: Rust has great documentation, a friendly compiler with useful error messages, and a growing ecosystem of tools and libraries.

Version 1.74.0
C/C++: Buffer overflow \Rightarrow Remote code execution
C/C++: Buffer overflow => Remote code execution
Java: Misuse of reflection, code loading => Remote code execution
C/C++: Buffer overflow => Remote code execution
Java: Misuse of reflection, code loading => Remote code execution

Go: Large or malformed inputs => Denial of service
Rust: Large or malformed inputs => Denial of service
Monitoring the software supply chain
Monitoring the software supply chain

What is a Software Bill of Materials (SBOM)?

I don’t know. Do you?
I don’t know. Do you?

Definitely includes list of packages and version.
Monitoring the software supply chain
What is a Software Bill of Materials (SBOM)?

“An SBOM is effectively a nested inventory, a list of ingredients that make up software components. An SBOM identifies and lists software components, information about those components, and supply chain relationships between them. The amount and type of information included in a particular SBOM may vary depending on factors such as the industry or sector and the needs of SBOM consumers.”

Monitoring the software supply chain
What is a Software Bill of Materials (SBOM)?

Definitely includes list of packages and version.
## Monitoring the software supply chain

**Go “SBOM”**

% go version -m $HOME/bin/gomote
/Users/rsc/bin/bin/gomote: go1.21.0

<table>
<thead>
<tr>
<th>path</th>
<th>golang.org/x/build/cmd/gomote</th>
<th>v0.0.0-20230809040836-4f3589752dd4</th>
<th>h1:gK+EqJ6LNNP/...</th>
</tr>
</thead>
<tbody>
<tr>
<td>mod</td>
<td>golang.org/x/build</td>
<td>v0.0.0-20210331224755-41bb18bfe9da</td>
<td>h1:oI5xCqsCo564...</td>
</tr>
<tr>
<td>dep</td>
<td>cloud.google.com/go/compute/metadata</td>
<td>v0.2.3</td>
<td>h1:mg4jlk7mAja6...</td>
</tr>
<tr>
<td>dep</td>
<td>github.com/aws/aws-sdk-go</td>
<td>v1.30.15</td>
<td>h1:Sd8DVzzE8S1...</td>
</tr>
<tr>
<td>dep</td>
<td>github.com/golang/groupcache</td>
<td>v0.1.4</td>
<td>h1:1kZ/sQM3sreP...</td>
</tr>
<tr>
<td>dep</td>
<td>github.com/google/s2a-go</td>
<td>v1.5.3</td>
<td>h1:t6JiXgmwXMjE...</td>
</tr>
<tr>
<td>dep</td>
<td>github.com/google/uuid</td>
<td>v0.4.0</td>
<td>h1:y9/cqRKtT9w...</td>
</tr>
<tr>
<td>dep</td>
<td>googleapis/enterprise-certificate-proxy</td>
<td>v0.2.3</td>
<td>h1:y73uSU6J157Q...</td>
</tr>
<tr>
<td>dep</td>
<td>googleapis/gax-go/v2</td>
<td>v2.10.0</td>
<td>h1:ebSgKfMxnOd...</td>
</tr>
<tr>
<td>dep</td>
<td>googleapis/jmespath/go-jmespath</td>
<td>v0.4.0</td>
<td>h1:BEl5cpj8n8U...</td>
</tr>
<tr>
<td>dep</td>
<td>go.opencensus.io</td>
<td>v0.24.0</td>
<td>h1:y73uSU6J157Q...</td>
</tr>
<tr>
<td>dep</td>
<td>golang.org/x/crypto</td>
<td>v0.9.0</td>
<td>h1:LF6fAI+IutBo...</td>
</tr>
<tr>
<td>dep</td>
<td>golang.org/x/net</td>
<td>v0.10.0</td>
<td>h1:X2//UzNDwYmt...</td>
</tr>
<tr>
<td>dep</td>
<td>golang.org/x/oauth2</td>
<td>v0.8.0</td>
<td>h1:6dkIj13j3LtZ...</td>
</tr>
</tbody>
</table>
Monitoring the software supply chain

Go Vulnerability Database

Data about new vulnerabilities come directly from Go package maintainers or sources such as MITRE and GitHub. Reports are curated by the Go Security team. Learn more at go.dev/security/vuln.

Search

Search by GO ID, alias, or import path

Submit

Recent Reports

GO-2023-2334

Affects: github.com/go-jose/go-jose/v3, github.com/square/go-jose  |  Published: Nov 21, 2023

The go-jose package is subject to a "billion hashes attack" causing denial-of-service when decrypting JWE inputs. This occurs when an attacker can provide a PBES2 encrypted JWE blob with a very large p2c value that, when decrypted, produces a denial-of-service.
Monitoring the software supply chain

Open-Source Vulnerability Database (OSV)

A distributed vulnerability database for Open Source

An open, precise, and distributed approach to producing and consuming vulnerability information for open source.

OSV schema

All advisories in this database use the OpenSSF OSV format, which was developed in collaboration with open source communities.
% govulncheck -mode=binary gomote
Scanning your binary for known vulnerabilities...

Vulnerability #1: GO-2023-2153
  Denial of service from HTTP/2 Rapid Reset in google.golang.org/grpc
  More info: https://pkg.go.dev/vuln/GO-2023-2153
  Module: google.golang.org/grpc
  Found in: google.golang.org/grpc@v1.55.0
  Fixed in: google.golang.org/grpc@v1.58.3
  Example traces found:
    #1: grpc.Server.Serve
    #2: transport.NewServerTransport

Vulnerability #2: GO-2023-2043
  Improper handling of special tags within script contexts in html/template
  More info: https://pkg.go.dev/vuln/GO-2023-2043
  Standard library
  Found in: html/template@go1.21
  Fixed in: html/template@go1.21.1
  Example traces found:
    #1: template.Template.Execute
    #2: template.Template.ExecuteTemplate
% govulncheck ./gomote
Scanning your code and 399 packages across 23 dependent modules for known vulnerabilities...

=== Informational ===

Found 2 vulnerabilities in packages that you import, but there are no call stacks leading to the use of these vulnerabilities. You may not need to take any action. See https://pkg.go.dev/golang.org/x/vuln/cmd/govulncheck for details.

Vulnerability #1: GO-2023-2153
   Denial of service from HTTP/2 Rapid Reset in google.golang.org/grpc
   More info: https://pkg.go.dev/vuln/GO-2023-2153
   Module: google.golang.org/grpc
   Found in: google.golang.org/grpc@v1.58.2
   Fixed in: google.golang.org/grpc@v1.58.3

Vulnerability #2: GO-2023-2102
   ...

No vulnerabilities found.
Monitoring the software supply chain

Vulnerability Scanning in IDE

go.dev/s/acmscored
Monitoring the software supply chain

Vulnerability Scanning in IDE

go.dev/s/acmscored
Monitoring the software supply chain

Vulnerability Scanning in IDE

go.dev/s/acmscored
Monitoring the software supply chain

Vulnerability Scanning in Production
Open-Source Supply Chain Security at Google
Open-Source
Supply Chain Security
Historical Perspective

ESD-TB-74-193, Vol. II

MULTICS SECURITY EVALUATION: VULNERABILITY ANALYSIS

Paul A. Karger, Lt., USAF
Roger R. Schell, Major, USAF

June 1974

Approved for public release; distribution unlimited.

INFORMATION SYSTEMS TECHNOLOGY APPLICATIONS OFFICE
DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
ELECTRONIC SYSTEMS DIVISION (AFSC)
L. G. HANSCOM AFB, MA 01731

Trap doors can be inserted during the distribution phase. If updates are sent via insecure communications - either US Mail or insecure telecommunications, the penetrator can intercept the update and subtly modify it. The penetrator could also generate his own updates and distribute them using forged stationery.
Clearly when a trap door is inserted, it must be well hidden to avoid detection by system maintenance personnel. Trap doors can best be hidden in changes to the binary code of a compiled routine. Such a change is completely invisible on system listings and can be detected only by comparing bit by bit the object code and the compiler listing. However, object code trap doors are vulnerable to recompilations of the module in question.
It was noted above that while object code trap doors are invisible, they are vulnerable to recompilations. The compiler (or assembler) trap door is inserted to permit object code trap doors to survive even a complete recompilation of the entire system. In Multics, most of the ring 0 supervisor is written in PL/I. A penetrator could insert a trap door in the PL/I compiler to note when it is compiling a ring 0 module. Then the compiler would insert an object code trap door in the ring 0 module without listing the code in the listing. Since the PL/I compiler is itself written in PL/I, the trap door can maintain itself, even when the compiler is recompiled.

(38) Compiler trap doors are significantly more complex than the other trap doors described here, because they require a detailed knowledge of the compiler design. However, they are quite practical to implement at a cost of perhaps five times the level shown in Section 3.5. It should be noted that even costs several hundred times larger than those shown here would be considered nominal to a foreign agent.
Reflections on Trusting Trust

To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software.

KEN THOMPSON

INTRODUCTION

I thank the ACM for this award. I can’t help but feel that I am receiving this honor for timing and serendipity rather than for the programs I ever wrote. I will do this in three stages and try to bring it together at the end.
Ken Thompson’s Actual Code

Extract nih.a.

\%
\ar \ xv \ nih.a
x  x.c
x  rc

Let’s read x.c, a C program.

\%
\cat \ x.c

Declare the global variable nihflg, of implied type int.

nihflg;

codenih()
{
char  *p,*s;
int  i;

if(pflag)
   return;

p=line;
while(*p=='\t')
   p++;

s="name = crypt(pwbuf);"
for(i=0;i<21;i++)
   if(s[i]!=p[i])
      goto ll;
Do We Learn From History?

1974 Multics report
1983 Thompson lecture
1988 Internet worm
...

Do We Learn From History?

1974 Multics report
1983 Thompson lecture
1988 Internet worm
...

No.
Do We Learn From History?

1974 Multics report
1983 Thompson lecture
1988 Internet worm
...

No.

But why are things not worse?
Hope For the Future

Industry can fix problems when it wants to.

- HTTP to HTTPS
- 2-Factor Auth and Security Keys
- Passkeys?

Maybe we want to fix supply chain security next. Hopefully we will.
Open-Source Supply Chain Security at Google

Russ Cox (he/him)
ACM SCORED
November 2023

go.dev/s/acmscored