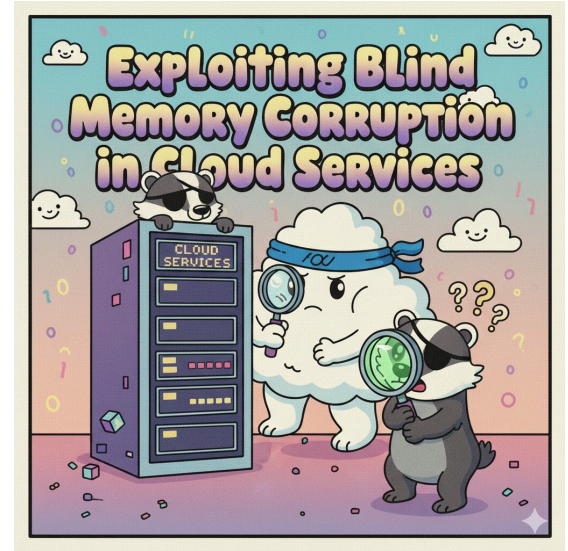




Exploiting Blind Memory Corruption in Cloud Services

Anthony Weems, Simon Scannell, Stefan Schiller



Problem Statement

- Memory Corruption is not commonly associated with Cloud Security
- Considered a theoretical risk. We rarely see reports of successful Memory Corruption chains in Cloud / SaaS backends

Problem Statement

Our goals for this talk are to:

- Discuss why Memory Corruption is under-reported in Cloud
- Outline the unique challenges for exploiting Memory corruption in Cloud
- Demonstrate with an end-to-end chain in GCP that exploitation is feasible and viable

Prerequisites to get the most out of this talk

- We provide as much context, background and detail as possible
- However, you should be familiar with
 - The basics of Heap Buffer Overflow exploitation
 - Address Space Layout Randomization (ASLR)
 - Infoleak vulnerabilities
 - ROP chains
- We go into enough detail that you should be able to follow the talk even if you are a little rusty on these subjects
- Still, if you are missing some background knowledge, feel free to revisit this talk with the recording

Who We Are

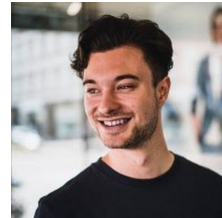
- Members of the Google Cloud Vulnerability Research team
- We hunt for vulnerabilities in Cloud providers and exploit them to gain an understanding of limitations, challenges and possibilities



Anthony Weems



Stefan Schiller



Simon Scannell



Our team's logo. Hunting bugs in the Cloud :)

Agenda

1. Defining Cloud vs “Classical” Targets for Memory Corruption Exploits
2. Solving Unique challenges in Cloud / SaaS environments
3. Introduction to the Target of our Case Study - GCP Artifact Analysis
4. Vulnerability Walkthrough
5. Exploit Details
6. Conclusion



Defining Cloud vs “Classical” Targets for Memory Corruption Exploits

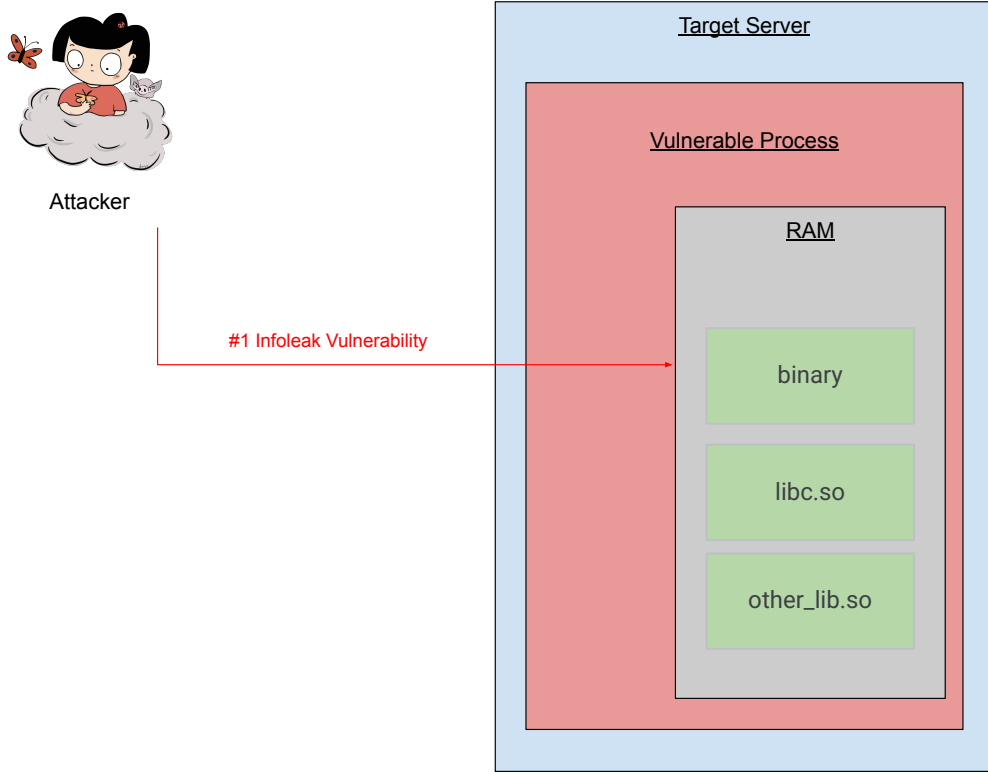


“Classic” Targets for Server-Side Memory Corruption

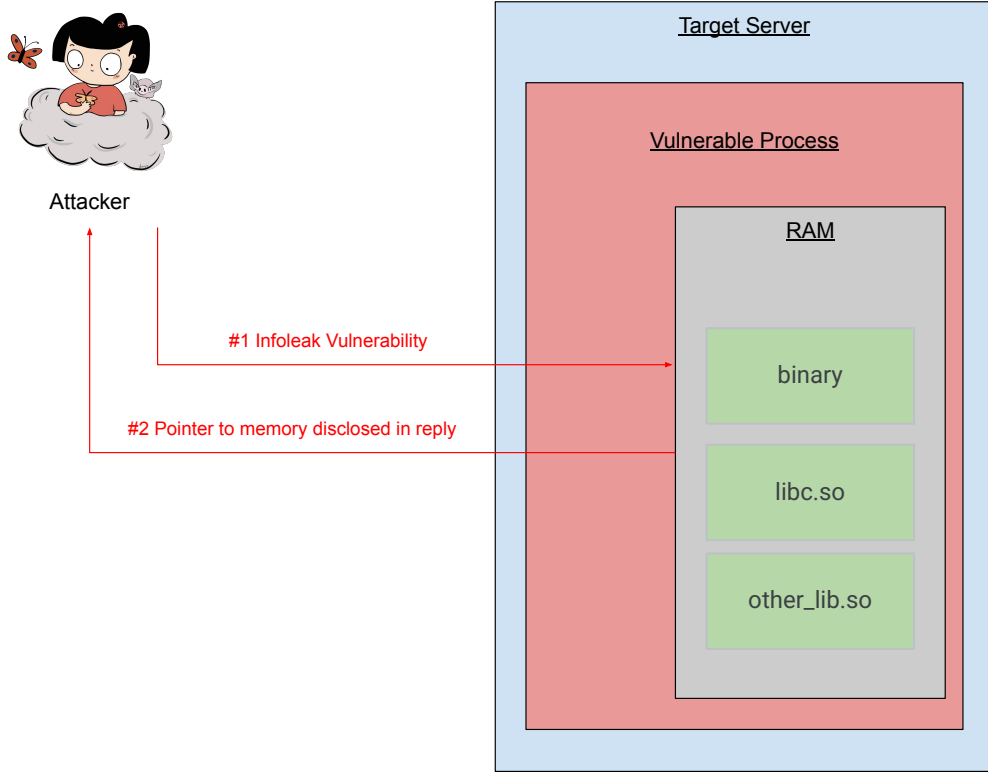
We define “classical” targets for memory corruption as:

- Distributed software where one or more **official binary releases exist**
- Customers install the binaries on **their server**
- Generally speaking, the entire software **runs on a single server**
- Attackers can **obtain binary releases and find vulnerabilities** by reading code, reverse engineering, fuzzing etc
- When an exploit works on one server, it generally works on all servers running the same version and configuration

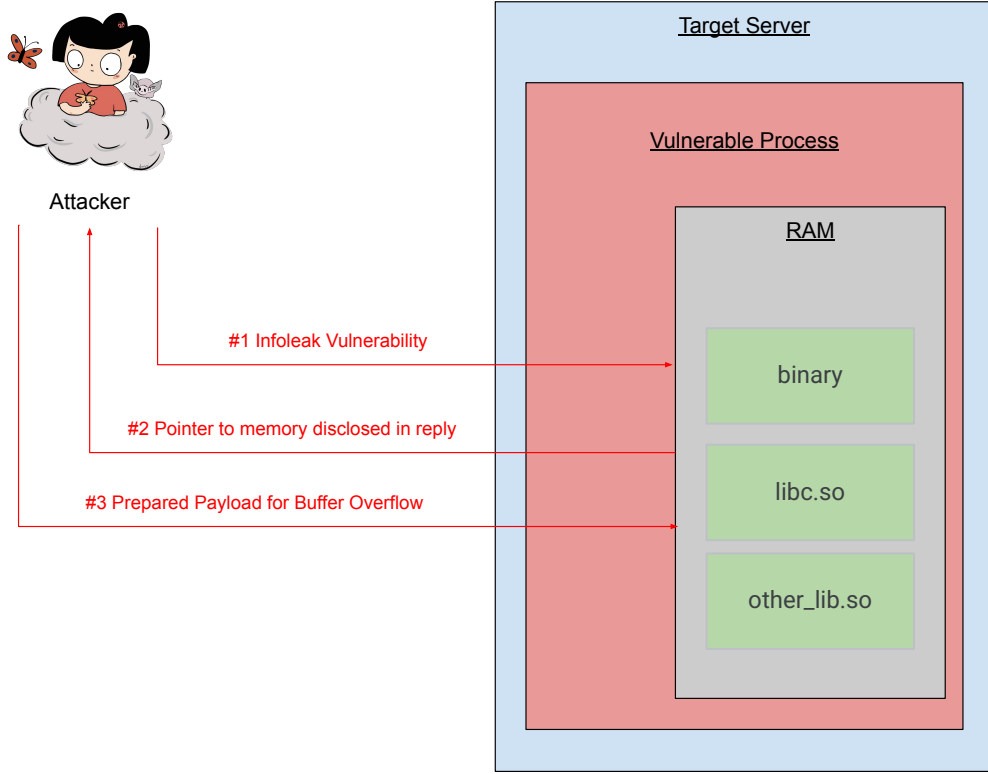
“Classic” Targets for Server-Side Memory Corruption



“Classic” Targets for Server-Side Memory Corruption



“Classic” Targets for Server-Side Memory Corruption



Cloud Targets for Server-Side Memory Corruption

We define Cloud targets for memory corruption as:

- A Service running with an **unknown binary** and **unknown libraries** in an **unknown environment** (OS, distribution, etc.)
- There is **no way to obtain a replication** of the environment for testing
- **Load Balancing** is typically utilized
- Although some Open-Source libraries may be used, their exact usage is unknown to an attacker
- In the context of Cloud, Services are typically **implemented in Memory-Safe languages** (JavaScript, Python, Go, Rust, Java etc.)

Cloud Targets for Server-Side Memory Corruption

This can seem very daunting when staring down the UI of a service and having no insight into how it works. However:

- Once you know what a service does, you can make **educated guesses on how it works**
 - For example: If a service converts images, you know it will have to use some sort of image conversion library
- The good news: The industry still heavily relies on **libraries written in C**. A lot of time memory safe languages just use FFI wrappers to call into them

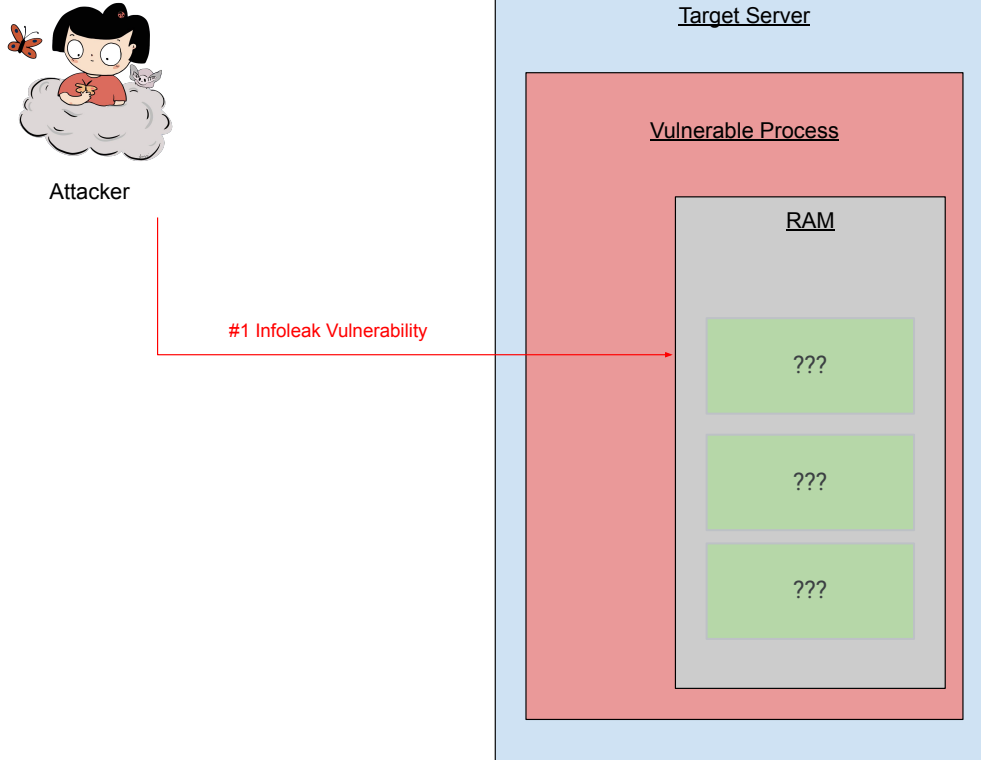
Cloud Targets for Server-Side Memory Corruption

Let's continue with the example of a service that converts images for you.

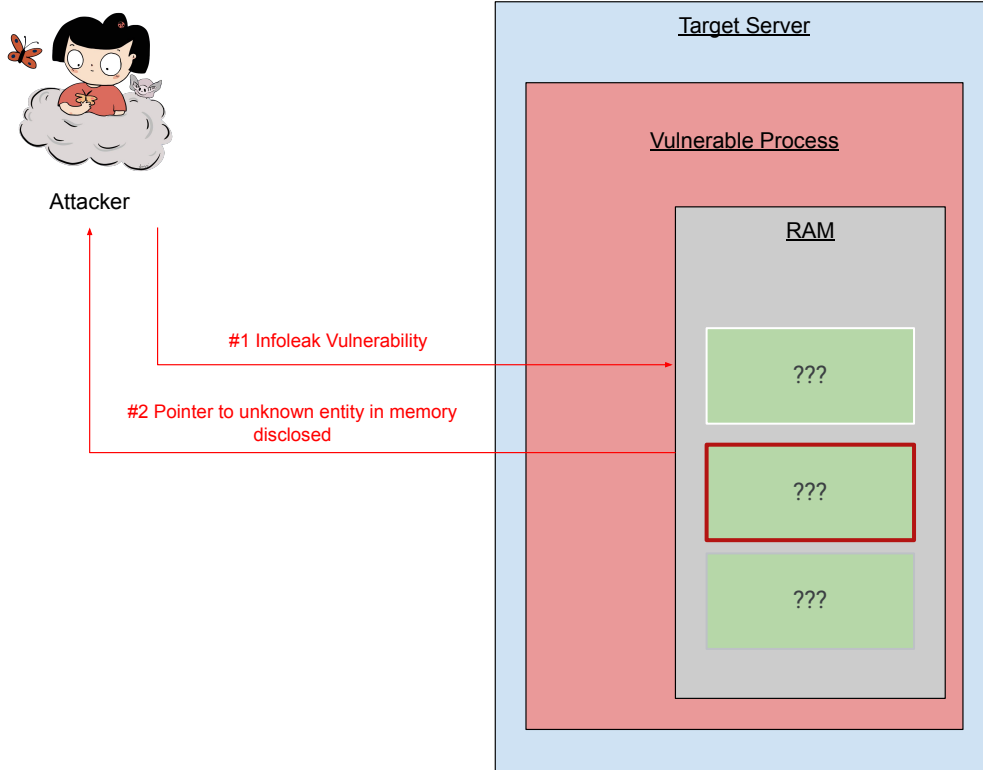
- You identified the exact library that is used by the service
- It is written in C and you identified a known but unpatched vulnerability
- Triggering the Buffer Overflow causes a server error, you are certain it works on the remote server
- You also identified an Out of Bounds Read that lets you read memory

Let's look at some of the problems you would then run into

The Unknown Binary Problem



The Unknown Binary Problem



The Unknown Binary Problem

Let's assume you know from the application logic what the pointer "should" point to, for example a function or global variable.

Since the binaries are likely all **custom-compiled releases**, the pointer does not give us any more information about the **location of ROP chain gadgets** etc

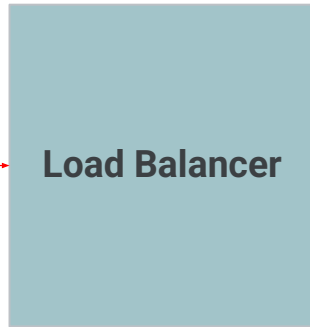
In addition...

The Load Balancing Problem



Attacker

#1 Infolack Vulnerability



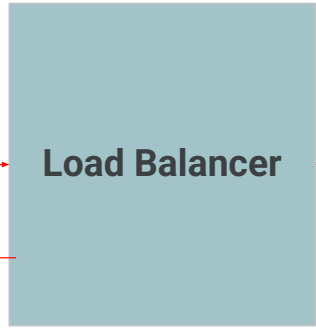
The Load Balancing Problem



Attacker

#1 Infoleak Vulnerability

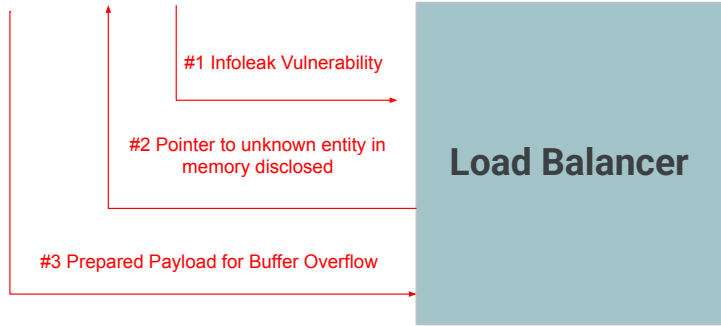
#2 Pointer to unknown entity in memory disclosed



The Load Balancing Problem



Attacker



The Load Balancing Problem

Even if we can use the pointer we leaked, there is no guarantee follow-up stages will trigger on the same worker

If there are only a few workers, brute-force can be viable but what if there are hundreds?

- You will have no idea if your exploit failed or just didn't hit the right worker
- Outages are one of the strongest detection signals



Solving Unique challenges in Cloud / SaaS environments



Solving Unique challenges in Cloud / SaaS environments

The existing Exploit workflow doesn't apply to Cloud services for the reasons we discussed in the previous section.

The solution is to either find **more specialized vulnerabilities** or **add steps** to the exploit workflow

Solving the Unknown Binary Problem

#1 - Arbitrary File Read vulnerabilities let us leak the binaries on a server

CVE-2023-6562 is a vulnerability in the JPEG 2000 standard itself that we discovered while exploiting an image conversion service at Google.[1]

By first reading the environment, we could prepare an exploit for a memory corruption issue.

[1] <https://github.com/google/security-research/security/advisories/GHSA-g6qc-fhcq-vhf9>,
<https://bughunters.google.com/blog/6220757425586176/cvr-the-mines-of-kakad-m>

Solving the Unknown Binary Problem

#2 - Arbitrary Memory Read primitives let us leak the binaries from memory

CVE-2016-5771 is a memory corruption that allowed the researchers to read the custom compiled PHP binary from the server and prepare a second stage exploit. [1]

[1] <https://www.evonide.com/how-we-broke-php-hacked-pornhub-and-earned-20000-dollar/>

Solving the Unknown Binary Problem

#3 - Data Only Exploits let us bypass the need for needing to know anything about the binaries in memory

An example would be

[1] <https://www.evonide.com/how-we-broke-php-hacked-pornhub-and-earned-20000-dollar/>

Solving the Unknown Binary Problem

#3 - Fingerprinting common Docker base images lets us make educated guesses about other binaries running on the server, e.g. libc.so

Google VRP hunters[1, 2] have already successfully done this work when they were able to execute arbitrary code in Gemini's Python execution sandbox and leak, among other things, the [libc.so](https://www.libc.so) file used by Google services

We were able to build on this work and immediately have **some knowledge of the target environment**

[1] <https://www.landh.tech/blog/20250327-we-hacked-gemini-source-code/>

[2] <https://embracethered.com/blog/posts/2024/exploring-google-bard-vm/>

Solving the Load Balancing Problem

This problem is potentially more tricky and requires another, specialized bug.

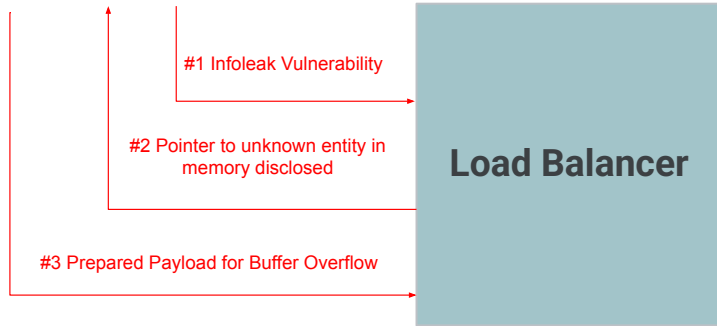
Load Balancing becomes an issue if you have to make **multiple, confounding requests to the same worker**

Load Balancing is not an issue if you can perform your **entire chain in one shot.**

Solving the Load Balancing Problem



Attacker



Solving the Load Balancing Problem

Even with ASLR and other mitigations, these bugs still exist and are exploited in the wild. [1]

- Vulnerabilities in interpreters fall into this category, like Python or JavaScript runtimes. These payloads can **react to their environment**, like using leaked pointers on the fly
- Highly sophisticated bug chains like FORCEDENTRY can establish some sort of scriptability

However, these bugs are **highly specialized** and **powerful**.

[1] <https://citizenlab.ca/2021/09/forcedentry-nso-group-imessage-zero-click-exploit-captured-in-the-wild/>

Conditional Corruption

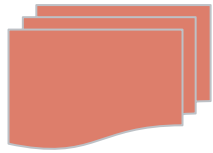
We want to expand the amount of possible bugs we can use against Cloud

In practice, most libraries will not support file formats with dynamic execution

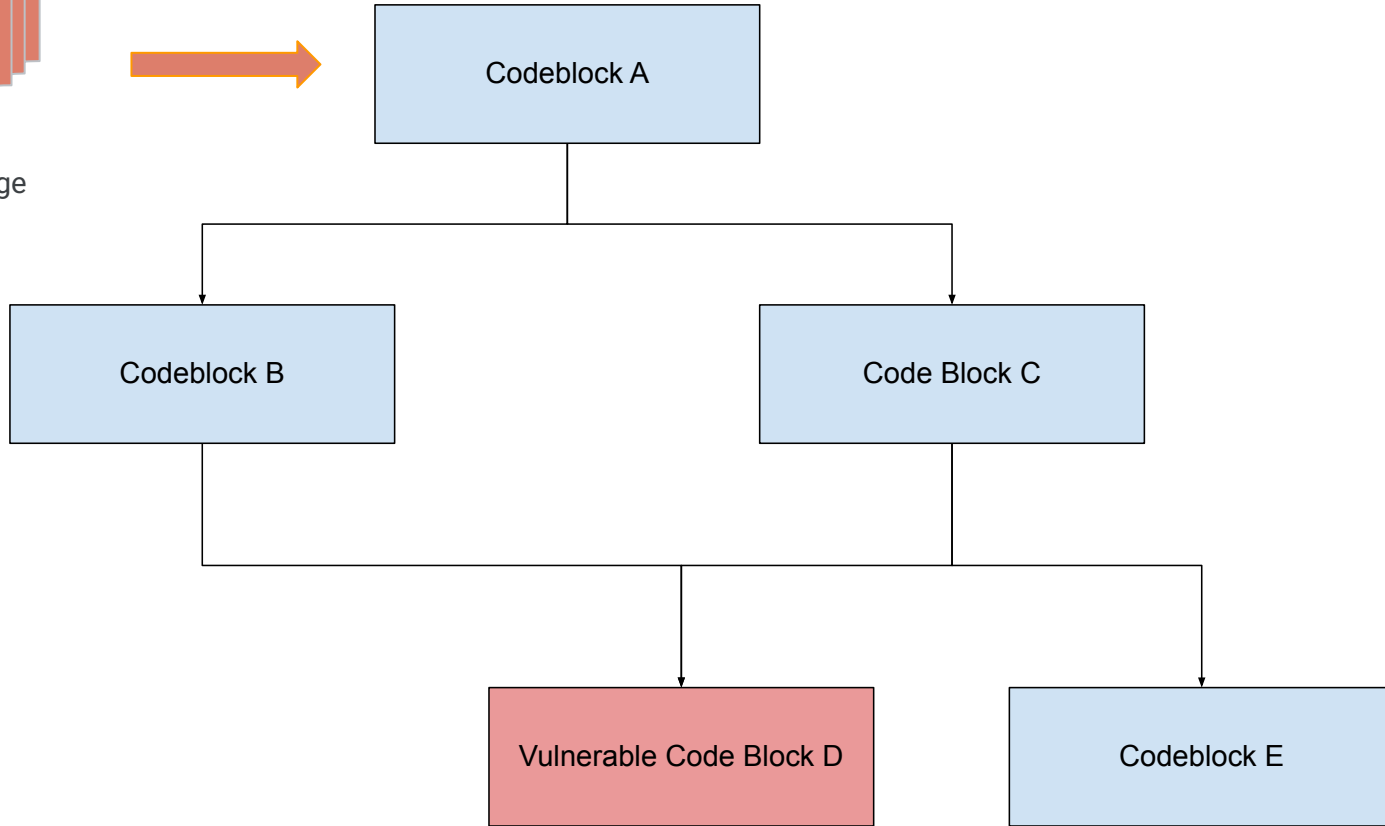
Image formats for example are static. They require the final pointers for the ROP chain to be encoded in their payload

Maybe we can make a compromise?

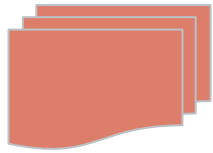
Conditional Corruption



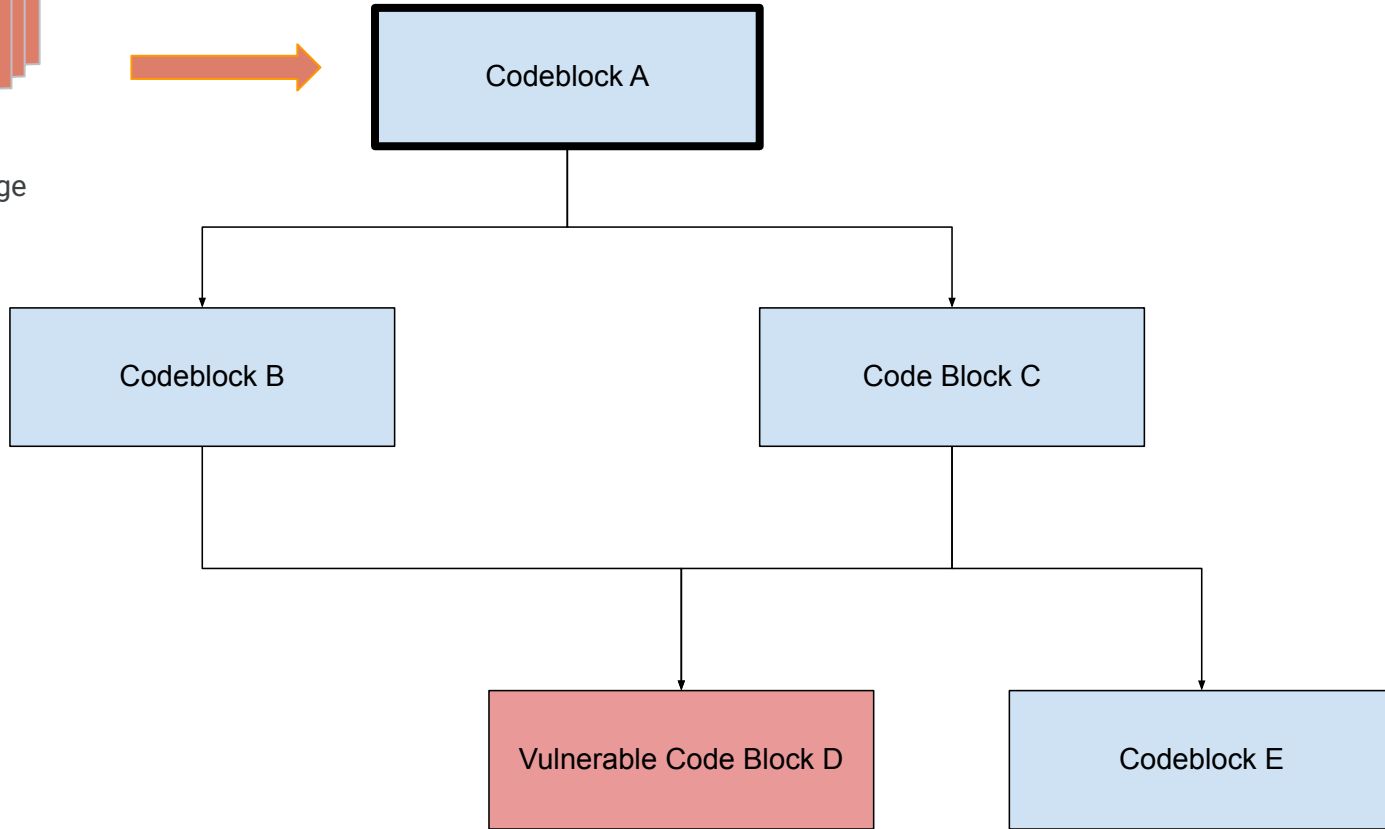
Exploit image



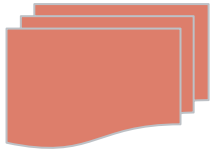
Conditional Corruption



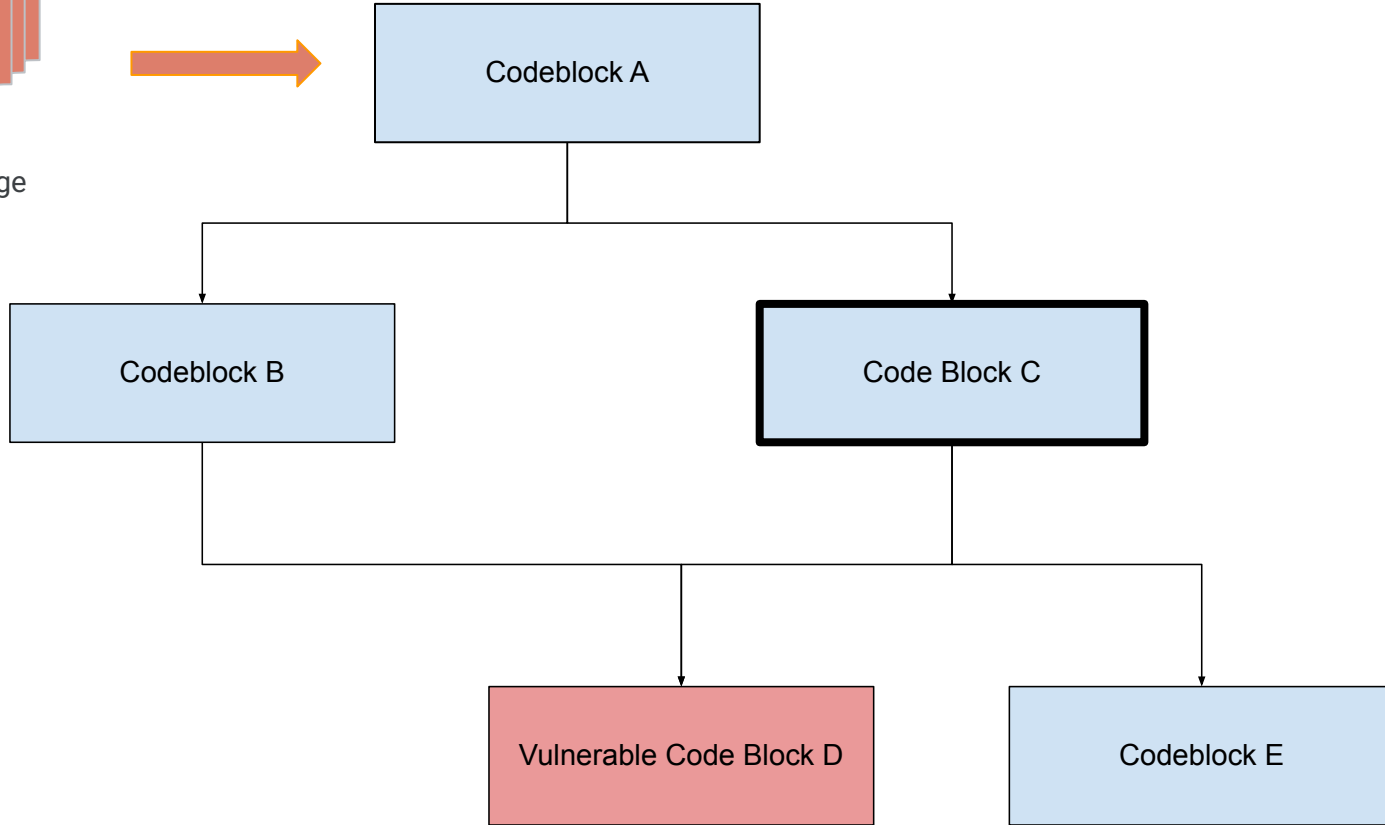
Exploit image



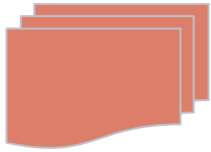
Conditional Corruption



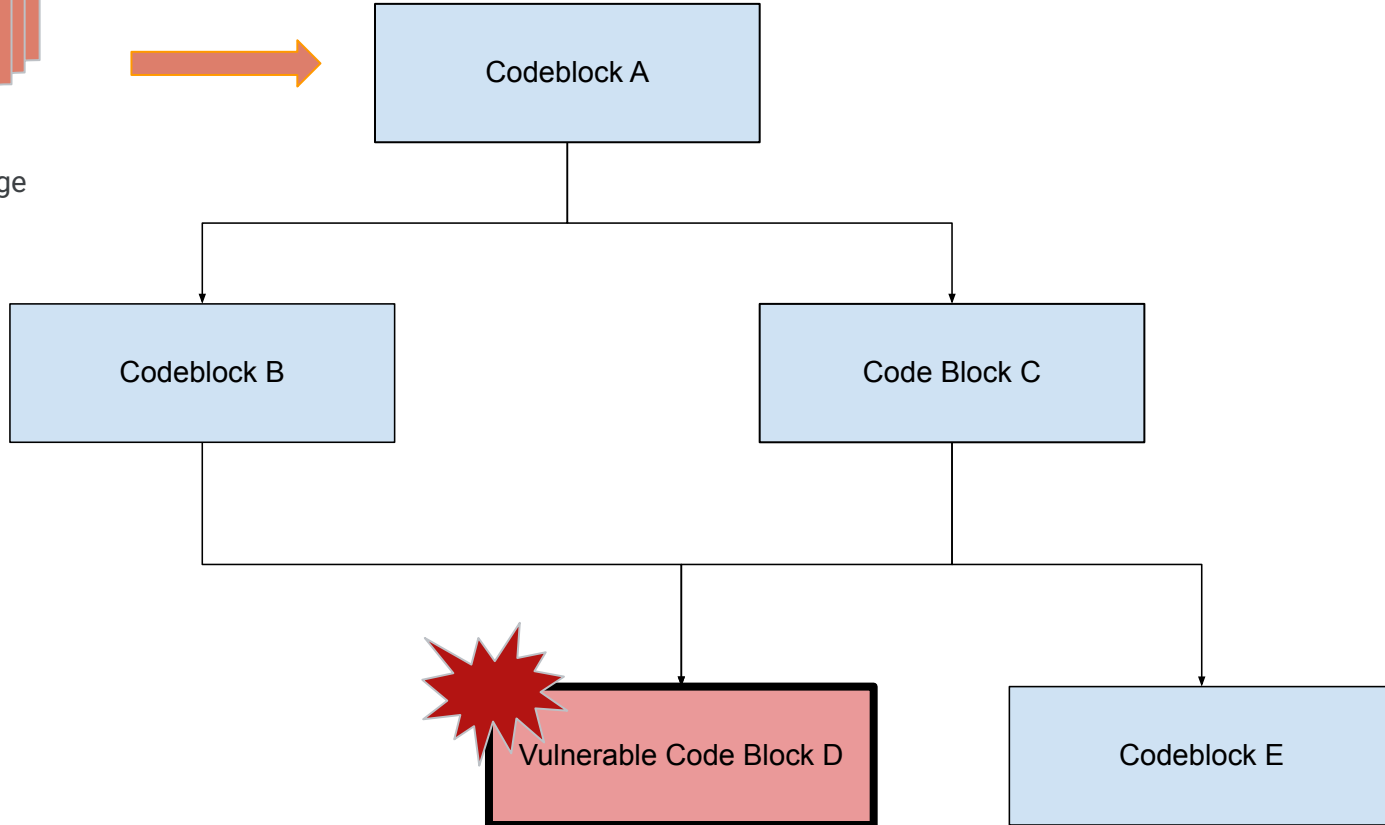
Exploit image



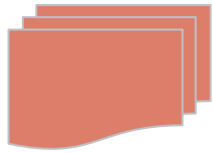
Conditional Corruption



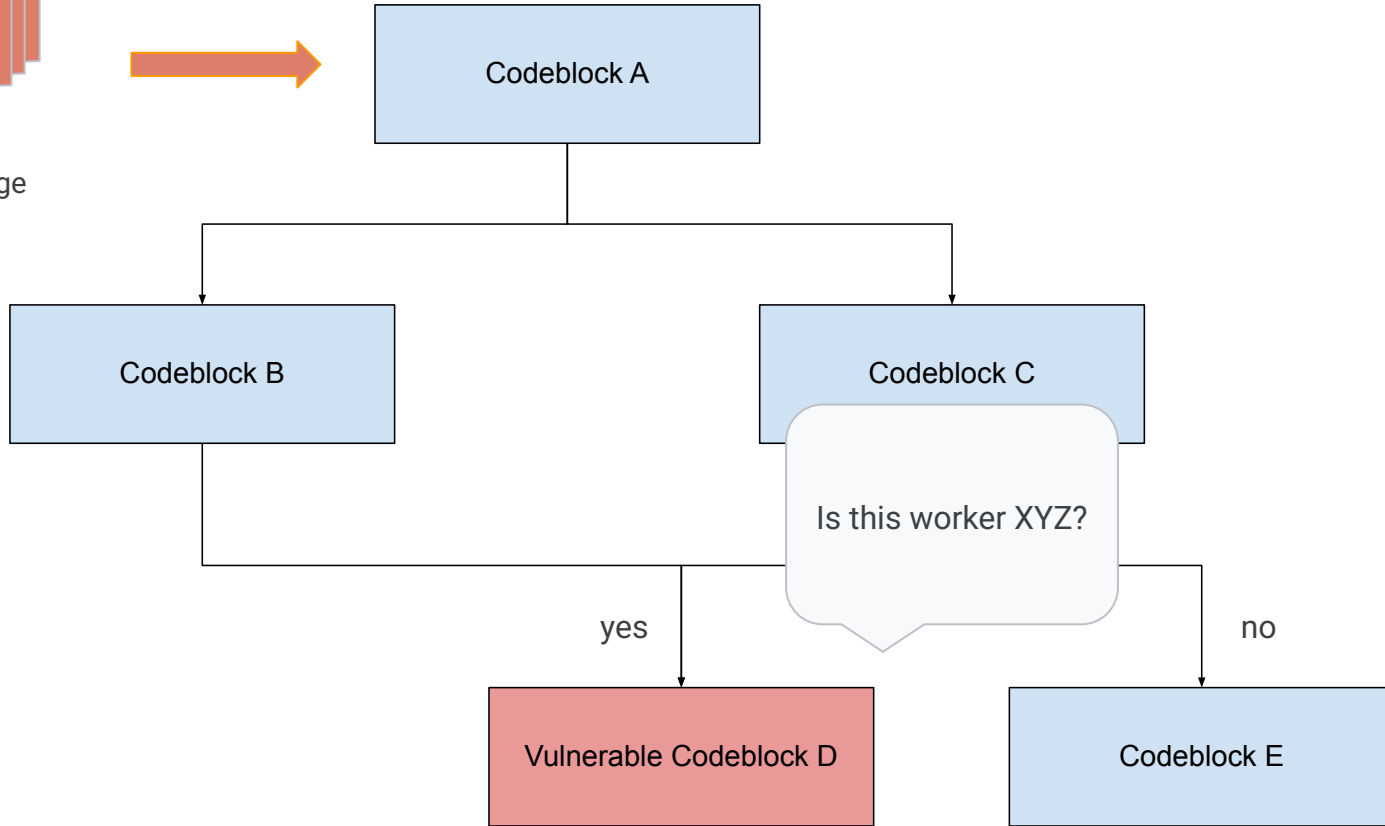
Exploit image



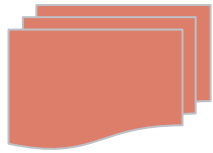
Conditional Corruption



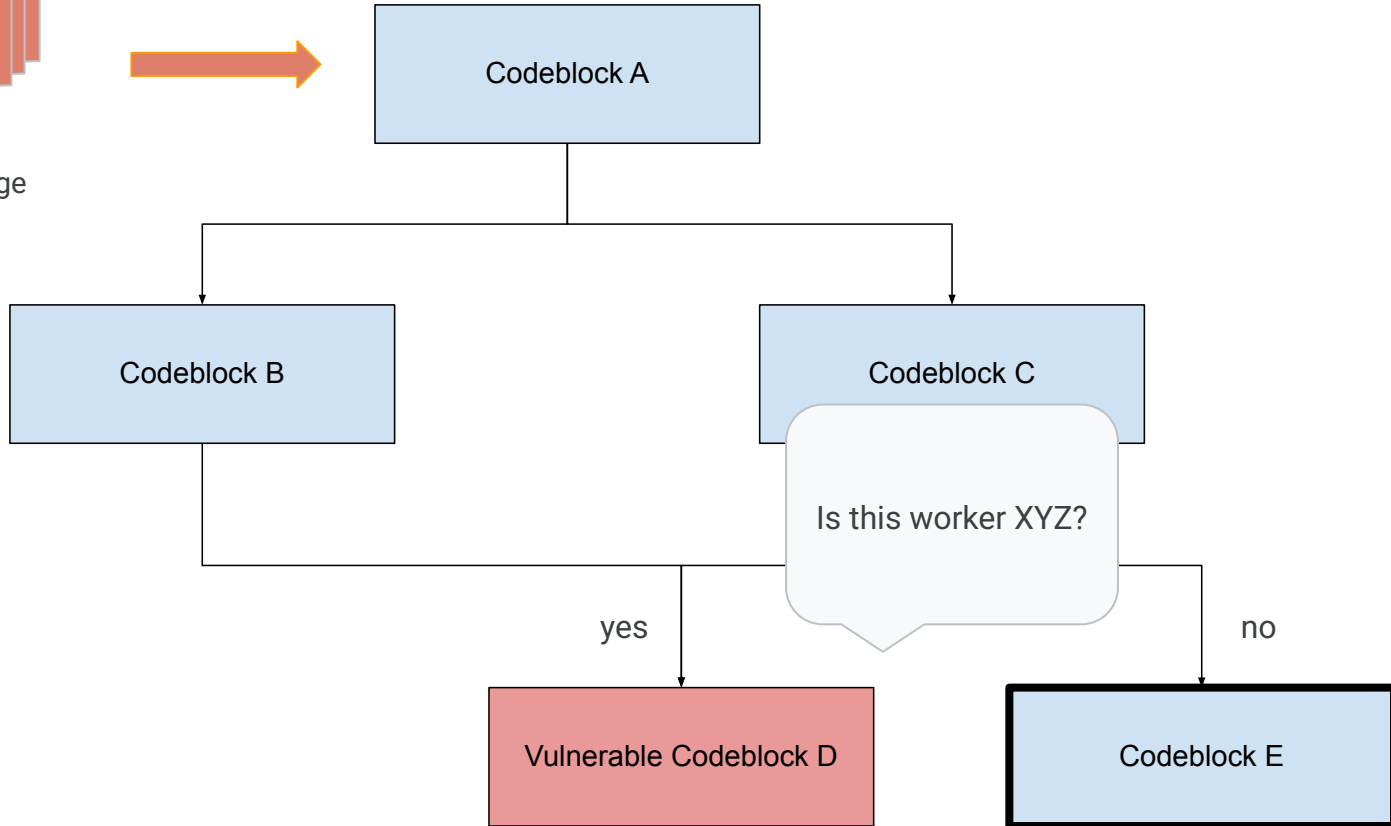
Exploit image



Conditional Corruption



Exploit image



Conditional Corruption

If we can't modify the payload once it is consumed by the application, if we hit the wrong worker, we have no way to stop the worker from crashing.

We do not need to find vulnerabilities that let us implement dynamic execution, just **vulnerabilities that let us error out** if we are **not on the correct worker**.

This increases the amount of possible vulnerabilities and techniques we can use.

Conditional Corruption

Previous case studies:

- CVE-2023-6562 was used when targeting an image service at Google that allowed us to only make the vulnerable library process our image if a certain byte was located at an expected location in memory. This condition is only true for the targeted worker [1]
- CVE-2023-29077 is an OOB Read in ClamAV that let us compare the result of the OOB value with an attacker controlled static value. This let us make sure that we are on the correct worker before triggering a Buffer Overflow. [2]

[1] <https://bughunters.google.com/blog/6220757425586176/cvr-the-mines-of-kakad-m>

[2] <https://www.control.rip/posts/2024/06/16/exploiting-clam-antivirus>

Summary

Exploiting Memory Corruption in Cloud is **difficult** and **requires specialized bugs**.

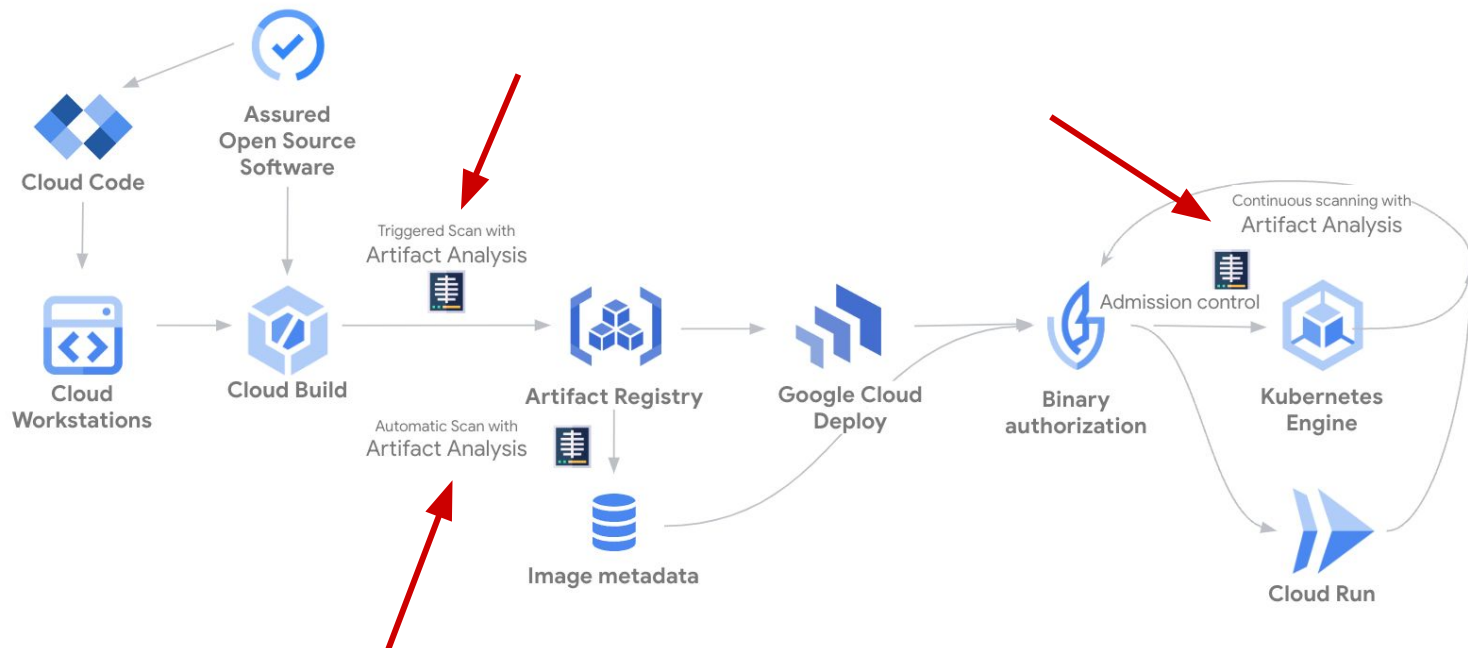
However, we believe it is viable and sometimes even the only vulnerability you can go for. We believe it should be **considered a practical risk** that can be realized with patience and sophistication.



Artifact Analysis



Intro to Artifact Analysis



Intro to Artifact Analysis

- Artifact Analysis scans containers images across Google Cloud products
- As part of this analysis, Artifact Analysis runs our open-source SCALIBR
- SCALIBR is a **Software Composition Analysis LIBRARY** that can:
 - scan file systems or container images and extract package metadata
 - report vulnerabilities in these packages
 - e.g. scan a node package . json for packages with CVEs
- Supports 20+ languages and package managers



Scan results

Effective severity based on factors such as exploitability, scope, impact, and maturity of the vulnerability.

Scan details

Scanning enablement	✔ Scanning active
Package types scanned	Go, Maven, Npm, Composer, Python, Nuget, OS, Rubygems, Rust
Total vulnerabilities	83 ?

✔ Fixes Available

Total [81](#)

✘ No fix available

🔴 Critical	–
🔴 High	–
🟡 Medium	1
🟠 Low	1

📘 VEX Status [Preview](#)

Unspecified	83
🔴 Critical	6
🔴 High	21
🟡 Medium	31
🟠 Low	3

Name	Effective severity ↓	CVSS	Fix available	VEX status	Package	Package type	
CVE-2024-45491	🔴 Critical	9.8	Yes	Unspecified	expat	OS	View fix
CVE-2024-45337	🔴 Critical	9.1	Yes	Unspecified	golang.org/x/crypto	Go	View fix
CVE-2024-45492	🔴 Critical	9.8	Yes	Unspecified	expat	OS	View fix
CVE-2024-5535	🔴 Critical	9.1	Yes	Unspecified	openssl	OS	View fix
CVE-2024-24790	🔴 Critical	9.8	Yes	Unspecified	go	Go stdlib	View fix
CVE-2024-7592	🔴 High	7.5	Yes	Unspecified	python3	OS	View fix
CVE-2025-22868	🔴 High	7.5	Yes	Unspecified	golang.org/x/oauth2	Go	View fix
CVE-2024-12254	🔴 High	7.5	Yes	Unspecified	python3	OS	View fix

Artifact Analysis as a Target

- Artifact Analysis has the following interesting properties
 - scans **user-submitted** container images
 - **extracts** these images to disk
 - **parses** the content of package metadata files
 - **no build-in sandbox** for all the above
 - written in Go, **memory safe language**

Artifact Analysis as a Target

- Artifact Analysis has the following interesting properties
 - scans **user-submitted** container images
 - **extracts** these images to disk
 - **parses** the content of package metadata files
 - **no build-in sandbox** for all the above
 - written in Go, **memory safe language**... or is it?

Artifact Analysis as a Target

- Artifact Analysis has the following interesting properties
 - scans **user-submitted** container images
 - **extracts** these images to disk
 - **parses** the content of package metadata files
 - **no build-in sandbox** for all the above
 - written in Go, **memory safe language...** or is it?
 - RPM parser uses sqlite, which uses **CGO**

Artifact Analysis as a Target

- Artifact Analysis has the following interesting properties
 - scans **user-submitted** container images
 - **extracts** these images to disk
 - **parses** the content of package metadata files
 - **no build-in sandbox** for all the above
 - written in Go, **memory safe language**... or is it?
 - RPM parser uses sqlite, which uses **CGO**

← Files master go-sqlite3 / sqlite3-binding.c 8.85 MB

Code Blame

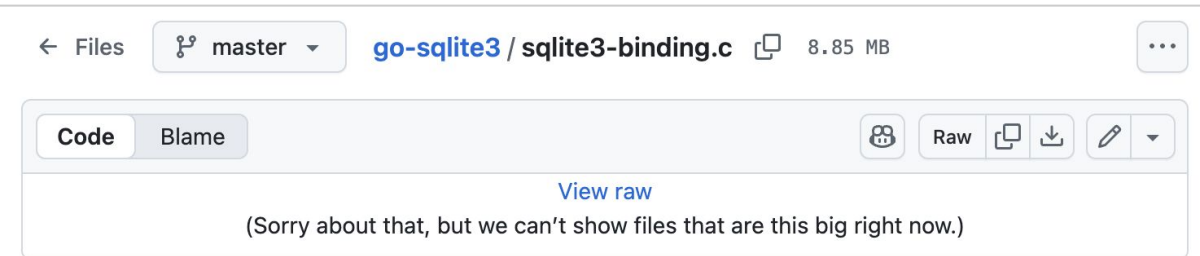


[View raw](#)

(Sorry about that, but we can't show files that are this big right now.)

Artifact Analysis as a Target

- Artifact Analysis has the following interesting properties
 - scans **user-submitted** container images
 - **extracts** these images to disk
 - **parses** the content of package metadata files
 - **no build-in sandbox** for all the above
 - written in Go, **memory safe language...** or is it?
 - RPM parser uses sqlite, which uses **CGO**

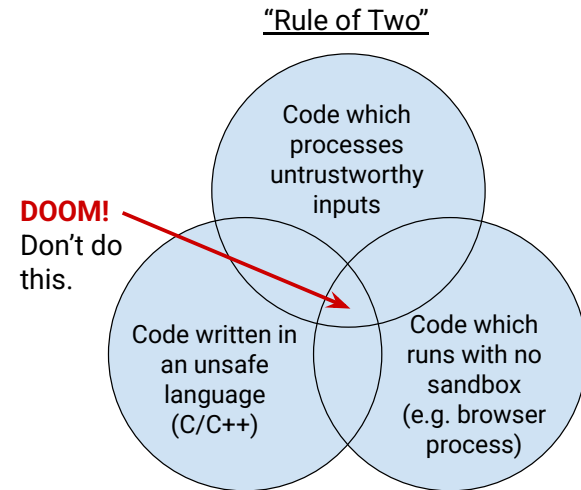


← Files master go-sqlite3 / sqlite3-binding.c 8.85 MB

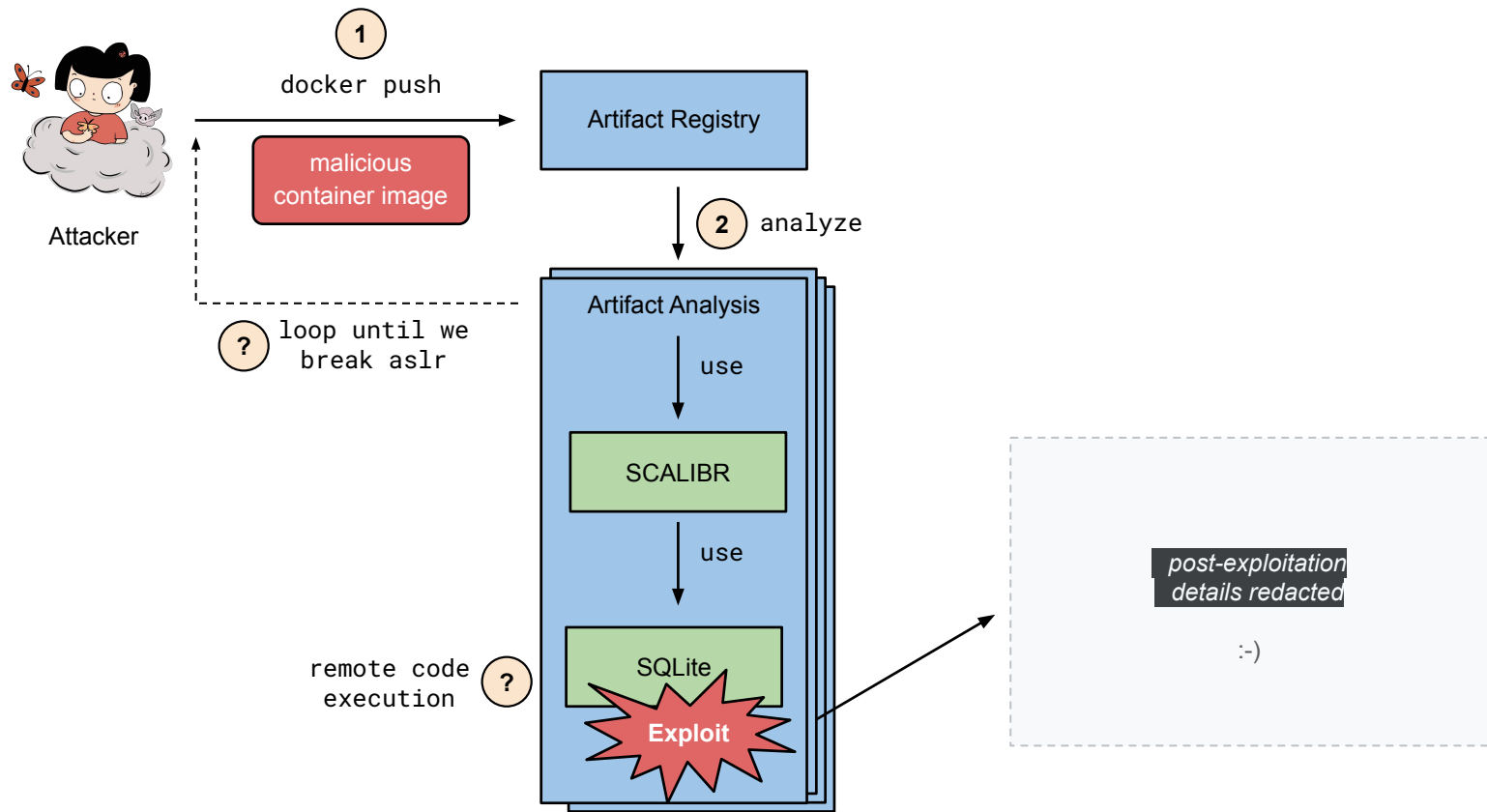
Code Blame Raw [Icons]

[View raw](#)

(Sorry about that, but we can't show files that are this big right now.)



high level plan



Looking for logic vulnerabilities

- SCALIBR extracts the container image to a local directory
- Container images are a series of TAR archive "layers"
- TAR archives permit relative paths and symlinks
- Potential for path traversal vulnerabilities

```
$ crane export python:3 | tar -tvf - | head
drwxr-xr-x 0/0 0 2025-09-29 00:00 usr
drwxr-xr-x 0/0 0 2025-09-29 00:00 usr/local
drwxr-xr-x 0/0 0 2025-10-08 21:27 usr/local/bin
lrwxrwxrwx 0/0 0 2025-10-08 21:27 usr/local/bin/idle -> idle3
lrwxrwxrwx 0/0 0 2025-10-08 21:27 usr/local/bin/pip -> pip3
lrwxrwxrwx 0/0 0 2025-10-08 21:27 usr/local/bin/pydoc -> pydoc3
lrwxrwxrwx 0/0 0 2025-10-08 21:27 usr/local/bin/python -> python3
```

```
for _, layer := range layers {
    dir := layer.Digest().String()
    tarReader := tar.NewReader(layer.Uncompressed())
    for {
        header, err := tarReader.Next()
        if err == io.EOF { break }
    }
}
```

```
for _, layer := range layers {
    dir := layer.Digest().String()
    tarReader := tar.NewReader(layer.Uncompressed())
    for {
        header, err := tarReader.Next()
        if err == io.EOF { break }
        // clean path, a/b/.. → a    a/../b → b
        cleanPath := path.Clean(header.Name)
        fullPath := path.Join(dir, cleanPath)
    }
}
```

```
for _, layer := range layers {
    dir := layer.Digest().String()
    tarReader := tar.NewReader(layer.Uncompressed())
    for {
        header, err := tarReader.Next()
        if err == io.EOF { break }
        // clean path, a/b/.. → a    a/./b → b
        cleanPath := path.Clean(header.Name)
        fullPath := path.Join(dir, cleanPath)
        // skip file if it already exists
        if _, err = os.Lstat(fullPath); err == nil {
            continue
        }
    }
}
```

```
for _, layer := range layers {
    dir := layer.Digest().String()
    tarReader := tar.NewReader(layer.Uncompressed())
    for {
        header, err := tarReader.Next()
        if err == io.EOF { break }
        // clean path, a/b/.. → a  a/./b → b
        cleanPath := path.Clean(header.Name)
        fullPath := path.Join(dir, cleanPath)
        // skip file if it already exists
        if _, err = os.Lstat(fullPath); err == nil {
            continue
        }
        switch header.Typeflag {
        case tar.TypeReg:
            // write files directly to disk
            content, _ := io.ReadAll(tarReader)
            err = os.WriteFile(fullPath, content, header.FileInfo().Mode())
            if err != nil {
                return nil, fmt.Errorf("failed to write file: %w", err)
            }
        }
    }
}
```

...

```
cleanPath := path.Clean(header.Name)
fullPath := path.Join(dir, cleanPath)
```

```
// Clean returns the shortest path name equivalent to path
// by purely lexical processing. It applies the following rules
// iteratively until no further processing can be done:
// 1. Replace multiple / elements with a single one.
// 2. Eliminate each . path name element
// 3. Eliminate each inner .. path name element
//    along with the non-.. element that precedes it.
// 4. Eliminate .. elements that begin a rooted path:
//    that is, replace "/.." by "/" at the beginning.

// Join joins any number of path elements into a single path,
// separating them with /. The result is Cleaned.
```

```
cleanPath := path.Clean(header.Name)
fullPath := path.Join(dir, cleanPath)
```

```
// Clean returns the shortest path name equivalent to path
// by purely lexical processing. It applies the following rules
// iteratively until no further processing can be done:
// 1. Replace multiple / elements with a single one.
// 2. Eliminate each . path name element
// 3. Eliminate each inner .. path name element
//    along with the non-.. element that precedes it.
// 4. Eliminate .. elements that begin a rooted path:
//    that is, replace "/.." by "/" at the beginning.

// Join joins any number of path elements into a single path,
// separating them with /. The result is Cleaned.
```

```
dir := "/tmp/layer"
header.Name := "../../x"
```

```
cleanPath := path.Clean(header.Name)
fullPath := path.Join(dir, cleanPath)
```

```
// Clean returns the shortest path name equivalent to path
// by purely lexical processing. It applies the following rules
// iteratively until no further processing can be done:
// 1. Replace multiple / elements with a single one.
// 2. Eliminate each . path name element
// 3. Eliminate each inner .. path name element
//    along with the non-.. element that precedes it.
// 4. Eliminate .. elements that begin a rooted path:
//    that is, replace "/.." by "/" at the beginning.

// Join joins any number of path elements into a single path,
// separating them with /. The result is Cleaned.
```

```
dir := "/tmp/layer"
header.Name := "../../x"
```

1. no multiple / elements

```
cleanPath := path.Clean(header.Name)
fullPath := path.Join(dir, cleanPath)
```

```
// Clean returns the shortest path name equivalent to path
// by purely lexical processing. It applies the following rules
// iteratively until no further processing can be done:
// 1. Replace multiple / elements with a single one.
// 2. Eliminate each . path name element
// 3. Eliminate each inner .. path name element
//    along with the non-.. element that precedes it.
// 4. Eliminate .. elements that begin a rooted path:
//    that is, replace "/.." by "/" at the beginning.

// Join joins any number of path elements into a single path,
// separating them with /. The result is Cleaned.
```

```
dir := "/tmp/layer"
header.Name := "../../x"
```

1. no multiple / elements
2. no . path elements

```
cleanPath := path.Clean(header.Name)
fullPath := path.Join(dir, cleanPath)
```

```
// Clean returns the shortest path name equivalent to path
// by purely lexical processing. It applies the following rules
// iteratively until no further processing can be done:
// 1. Replace multiple / elements with a single one.
// 2. Eliminate each . path name element
// 3. Eliminate each inner .. path name element
//    along with the non-.. element that precedes it.
// 4. Eliminate .. elements that begin a rooted path:
//    that is, replace "/.." by "/" at the beginning.

// Join joins any number of path elements into a single path,
// separating them with /. The result is Cleaned.
```

```
dir := "/tmp/layer"
header.Name := "../../x"
```

1. no multiple / elements
2. no . path elements
3. no *inner* .. elements

```
cleanPath := path.Clean(header.Name)
fullPath := path.Join(dir, cleanPath)
```

```
// Clean returns the shortest path name equivalent to path
// by purely lexical processing. It applies the following rules
// iteratively until no further processing can be done:
// 1. Replace multiple / elements with a single one.
// 2. Eliminate each . path name element
// 3. Eliminate each inner .. path name element
//    along with the non-.. element that precedes it.
// 4. Eliminate .. elements that begin a rooted path:
//    that is, replace "/.." by "/" at the beginning.

// Join joins any number of path elements into a single path,
// separating them with /. The result is Cleaned.
```

```
dir := "/tmp/layer"
header.Name := "../../x"
```

1. no multiple / elements
2. no . path elements
3. no *inner* .. elements
4. no rooted path

```
cleanPath := path.Clean(header.Name)
fullPath := path.Join(dir, cleanPath)
```

```
// Clean returns the shortest path name equivalent to path
// by purely lexical processing. It applies the following rules
// iteratively until no further processing can be done:
// 1. Replace multiple / elements with a single one.
// 2. Eliminate each . path name element
// 3. Eliminate each inner .. path name element
//    along with the non-.. element that precedes it.
// 4. Eliminate .. elements that begin a rooted path:
//    that is, replace "/.." by "/" at the beginning.

// Join joins any number of path elements into a single path,
// separating them with /. The result is Cleaned.
```

```
dir := "/tmp/layer"
header.Name := "../../x"
```

1. no multiple / elements
2. no . path elements
3. no *inner* .. elements
4. no rooted path

```
path.Clean("../../x") is
equivalent to "../../x"
```

```
cleanPath := path.Clean(header.Name)
fullPath := path.Join(dir, cleanPath)

// Clean returns the shortest path name equivalent to path
// by purely lexical processing. It applies the following rules
// iteratively until no further processing can be done:
// 1. Replace multiple / elements with a single one.
// 2. Eliminate each . path name element
// 3. Eliminate each inner .. path name element
//    along with the non-.. element that precedes it.
// 4. Eliminate .. elements that begin a rooted path:
//    that is, replace "/.." by "/" at the beginning.

// Join joins any number of path elements into a single path,
// separating them with /. The result is Cleaned.
```

```
dir := "/tmp/layer"
header.Name := "../../x"
```

1. no multiple / elements
2. no . path elements
3. no *inner* .. elements
4. no rooted path

```
path.Clean("../../x") is
equivalent to "../../x"
```

```
path.Join(dir, cleanPath)
will concat the elements
```

```
"/tmp/layer/../../x"
```

```
cleanPath := path.Clean(header.Name)
fullPath := path.Join(dir, cleanPath)
```

```
// Clean returns the shortest path name equivalent to path
// by purely lexical processing. It applies the following rules
// iteratively until no further processing can be done:
// 1. Replace multiple / elements with a single one.
// 2. Eliminate each . path name element
// 3. Eliminate each inner .. path name element
//    along with the non-.. element that precedes it.
// 4. Eliminate .. elements that begin a rooted path:
//    that is, replace "/.." by "/" at the beginning.

// Join joins any number of path elements into a single path,
// separating them with /. The result is Cleaned.
```

```
dir := "/tmp/layer"
header.Name := "../../x"
```

1. no multiple / elements
2. no . path elements
3. no *inner* .. elements
4. no rooted path

```
path.Clean("../../x") is
equivalent to "../../x"
```

```
path.Join(dir, cleanPath)
will concat the elements
```

```
"/tmp/layer/../../x"
```

and then run `path.Clean` on
the result, returning...



Using path traversal for RCE?

- We can construct a container image with relative `../.. /` paths and write to an arbitrary path in the Artifact Analysis service
- However, in Google, services often run with **minimal, read-only filesystems**
- Our service is a Go binary, so there aren't any executables to overwrite

Using path traversal for RCE?

- We can construct a container image with relative `../.. / .. /` paths and write to an arbitrary path in the Artifact Analysis service
- However, in Google, services often run with **minimal, read-only filesystems**
- Our service is a Go binary, so there aren't any executables to overwrite
- **No path to remote code execution** exclusively using the path traversal

Using path traversal for exploit reliability

- Assume we find memory corruption in SQLite...
 - We need some way to defeat ASLR without crashing the service

Using path traversal for exploit reliability

- Assume we find memory corruption in SQLite...
 - We need some way to defeat ASLR without crashing the service
 - Recall the following code from the unpack function:

```
// skip file if it already exists
if _, err = os.Lstat(fullPath); err == nil {
    continue
}
// write file, return early if write fails
err = os.WriteFile(fullPath, content, header.FileInfo().Mode())
if err != nil {
    return nil, fmt.Errorf("failed to write file: %w", err)
}
```

Using path traversal for exploit reliability

- Assume we find memory corruption in SQLite...
 - We need some way to defeat ASLR without crashing the service
 - Recall the following code from the unpack function:

```
// skip file if it already exists
if _, err = os.Lstat(fullPath); err == nil {
    continue
}
// write file, return early if write fails
err = os.WriteFile(fullPath, content, header.FileInfo().Mode())
if err != nil {
    return nil, fmt.Errorf("failed to write file: %w", err)
}
```

- What if we could build an oracle such that
 - our exploit runs if we are **correct** about ASLR
 - nothing happens if we are **incorrect** about ASLR

Conditional corruption

```
$ ls -la /proc/`pidof scalibr`/map_files
total 0
dr-x----- 2 .
dr-xr-xr-x 9 ..
lr----- 1 16d6000 -> scalibr
lr----- 1 24fe000 -> scalibr
lr----- 1 2501000 -> scalibr
lr----- 1 400000 -> scalibr
lr----- 1 7f633b6b5000 -> libc.so.6
lr----- 1 7f633b6dd000 -> libc.so.6
lr----- 1 7f633b842000 -> libc.so.6
lr----- 1 7f633b898000 -> libc.so.6
lr----- 1 7f633b89c000 -> libc.so.6
lr----- 1 7f633b8c9000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ca000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8f2000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8fd000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ff000 -> ld-linux-x86-64.so.2
lr----- 1 9dd000 -> scalibr
```

Conditional corruption

```
$ ls -la /proc/`pidof scalibr`/map_files
total 0
dr-x----- 2 .
dr-xr-xr-x 9 ..
lr----- 1 16d6000 -> scalibr
lr----- 1 24fe000 -> scalibr
lr----- 1 2501000 -> scalibr
lr----- 1 4000000 -> scalibr
lr----- 1 7f633b6b5000 -> libc.so.6
lr----- 1 7f633b6dd000 -> libc.so.6
lr----- 1 7f633b842000 -> libc.so.6
lr----- 1 7f633b898000 -> libc.so.6
lr----- 1 7f633b89c000 -> libc.so.6
lr----- 1 7f633b8c9000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ca000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8f2000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8fd000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ff000 -> ld-linux-x86-64.so.2
lr----- 1 9dd000 -> scalibr
```

```
$ tar -tf exploit-wrong-aslr.tar
.../proc/self/map_files/7f6330000000
var/lib/rpm/rpmdb.sqlite
```

```
// skip file if it already exists
if _, err = os.Lstat(fullPath); err == nil {
    continue
}
// write file, return early if write fails
err = os.WriteFile(fullPath, content, header.FileInfo())
if err != nil {
    return nil, fmt.Errorf("failed to write file: %w",
}
```

Conditional corruption

```
$ ls -la /proc/`pidof scalibr`/map_files
total 0
dr-x----- 2 .
dr-xr-xr-x 9 ..
lr----- 1 16d6000 -> scalibr
lr----- 1 24fe000 -> scalibr
lr----- 1 2501000 -> scalibr
lr----- 1 4000000 -> scalibr
lr----- 1 7f633b6b5000 -> libc.so.6
lr----- 1 7f633b6dd000 -> libc.so.6
lr----- 1 7f633b842000 -> libc.so.6
lr----- 1 7f633b898000 -> libc.so.6
lr----- 1 7f633b89c000 -> libc.so.6
lr----- 1 7f633b8c9000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ca000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8f2000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8fd000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ff000 -> ld-linux-x86-64.so.2
lr----- 1 9dd000 -> scalibr
```

```
$ tar -tf exploit-wrong-aslr.tar
.../proc/self/map_files/7f6330000000
var/lib/rpm/rpmdb.sqlite
```

```
// skip file if it already exists
if _, err = os.Lstat(fullPath); err == nil {
    continue
}
// write file, return early if write fails
err = os.WriteFile(fullPath, content, header.FileInfo())
if err != nil {
    return nil, fmt.Errorf("failed to write file: %w",
}
```

Conditional corruption

```
$ ls -la /proc/`pidof scalibr`/map_files
total 0
dr-x----- 2 .
dr-xr-xr-x 9 ..
lr----- 1 16d6000 -> scalibr
lr----- 1 24fe000 -> scalibr
lr----- 1 2501000 -> scalibr
lr----- 1 4000000 -> scalibr
lr----- 1 7f633b6b5000 -> libc.so.6
lr----- 1 7f633b6dd000 -> libc.so.6
lr----- 1 7f633b842000 -> libc.so.6
lr----- 1 7f633b898000 -> libc.so.6
lr----- 1 7f633b89c000 -> libc.so.6
lr----- 1 7f633b8c9000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ca000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8f2000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8fd000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ff000 -> ld-linux-x86-64.so.2
lr----- 1 9dd000 -> scalibr
```

```
$ tar -tf exploit-wrong-aslr.tar
.../proc/self/map_files/7f6330000000
var/lib/rpm/rpmdb.sqlite
```

```
// skip file if it already exists
if _, err = os.Lstat(fullPath); err == nil {
    continue
}
// write file, return early if write fails
err = os.WriteFile(fullPath, content, header.FileInfo())
if err != nil {
    return nil, fmt.Errorf("failed to write file: %w",
}
```

Conditional corruption

```
$ ls -la /proc/`pidof scalibr`/map_files
total 0
dr-x----- 2 .
dr-xr-xr-x 9 ..
lr----- 1 16d6000 -> scalibr
lr----- 1 24fe000 -> scalibr
lr----- 1 2501000 -> scalibr
lr----- 1 4000000 -> scalibr
lr----- 1 7f633b6b5000 -> libc.so.6
lr----- 1 7f633b6dd000 -> libc.so.6
lr----- 1 7f633b842000 -> libc.so.6
lr----- 1 7f633b898000 -> libc.so.6
lr----- 1 7f633b89c000 -> libc.so.6
lr----- 1 7f633b8c9000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ca000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8f2000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8fd000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ff000 -> ld-linux-x86-64.so.2
lr----- 1 9dd000 -> scalibr
```

```
$ tar -tf exploit-wrong-aslr.tar
.../proc/self/map_files/7f6330000000
var/lib/rpm/rpmdb.sqlite
```

```
// skip file if it already exists
if _, err = os.Lstat(fullPath); err == nil {
    continue
}
// write file, return early if write fails
err = os.WriteFile(fullPath, content, header.FileInfo())
if err != nil {
    return nil, fmt.Errorf("failed to write file: %w",
}
```

Conditional corruption

```
$ ls -la /proc/`pidof scalibr`/map_files
total 0
dr-x----- 2 .
dr-xr-xr-x 9 ..
lr----- 1 16d6000 -> scalibr
lr----- 1 24fe000 -> scalibr
lr----- 1 2501000 -> scalibr
lr----- 1 4000000 -> scalibr
lr----- 1 7f633b6b5000 -> libc.so.6
lr----- 1 7f633b6dd000 -> libc.so.6
lr----- 1 7f633b842000 -> libc.so.6
lr----- 1 7f633b898000 -> libc.so.6
lr----- 1 7f633b89c000 -> libc.so.6
lr----- 1 7f633b8c9000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ca000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8f2000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8fd000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ff000 -> ld-linux-x86-64.so.2
lr----- 1 9dd000 -> scalibr
```

```
$ tar -tf exploit-wrong-aslr.tar
.../proc/self/map_files/7f6330000000
var/lib/rpm/rpmdb.sqlite
```

```
// skip file if it already exists
if _, err = os.Lstat(fullPath); err == nil {
    continue
}
// write file, return early if write fails
err = os.WriteFile(fullPath, content, header.FileInfo())
if err != nil {
    return nil, fmt.Errorf("failed to write file: %w",
}
```

Conditional corruption

```
$ ls -la /proc/`pidof scalibr`/map_files
total 0
dr-x----- 2 .
dr-xr-xr-x 9 ..
lr----- 1 16d6000 -> scalibr
lr----- 1 24fe000 -> scalibr
lr----- 1 2501000 -> scalibr
lr----- 1 4000000 -> scalibr
lr----- 1 7f633b6b5000 -> libc.so.6
lr----- 1 7f633b6dd000 -> libc.so.6
lr----- 1 7f633b842000 -> libc.so.6
lr----- 1 7f633b898000 -> libc.so.6
lr----- 1 7f633b89c000 -> libc.so.6
lr----- 1 7f633b8c9000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ca000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8f2000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8fd000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ff000 -> ld-linux-x86-64.so.2
lr----- 1 9dd000 -> scalibr
```

```
$ tar -tf exploit-wrong-aslr.tar
.../proc/self/map_files/7f6330000000
var/lib/rpm/rpmdb.sqlite
```

```
// skip file if it already exists
if _, err = os.Lstat(fullPath); err == nil {
    continue
}
// write file, return early if write fails
err = os.WriteFile(fullPath, content, header.FileInfo())
if err != nil {
    return nil, fmt.Errorf("failed to write file: %w",
}
```

↑ exits early if we are wrong about ASLR

Conditional corruption

```
$ ls -la /proc/`pidof scalibr`/map_files
total 0
dr-x----- 2 .
dr-xr-xr-x 9 ..
lr----- 1 16d6000 -> scalibr
lr----- 1 24fe000 -> scalibr
lr----- 1 2501000 -> scalibr
lr----- 1 4000000 -> scalibr
lr----- 1 7f633b6b5000 -> libc.so.6
lr----- 1 7f633b6dd000 -> libc.so.6
lr----- 1 7f633b842000 -> libc.so.6
lr----- 1 7f633b898000 -> libc.so.6
lr----- 1 7f633b89c000 -> libc.so.6
lr----- 1 7f633b8c9000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ca000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8f2000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8fd000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ff000 -> ld-linux-x86-64.so.2
lr----- 1 9dd000 -> scalibr
```

```
$ tar -tf exploit-correct-aslr.tar
../../proc/self/map_files/7f633b6b5000
var/lib/rpm/rpmdb.sqlite
```

```
// skip file if it already exists
if _, err = os.Lstat(fullPath); err == nil {
    continue
}
// write file, return early if write fails
err = os.WriteFile(fullPath, content, header.FileInfo())
if err != nil {
    return nil, fmt.Errorf("failed to write file: %w",
}
```

Conditional corruption

```
$ ls -la /proc/`pidof scalibr`/map_files
total 0
dr-x----- 2 .
dr-xr-xr-x 9 ..
lr----- 1 16d6000 -> scalibr
lr----- 1 24fe000 -> scalibr
lr----- 1 2501000 -> scalibr
lr----- 1 4000000 -> scalibr
lr----- 1 7f633b6b5000 -> libc.so.6
lr----- 1 7f633b6dd000 -> libc.so.6
lr----- 1 7f633b842000 -> libc.so.6
lr----- 1 7f633b898000 -> libc.so.6
lr----- 1 7f633b89c000 -> libc.so.6
lr----- 1 7f633b8c9000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ca000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8f2000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8fd000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ff000 -> ld-linux-x86-64.so.2
lr----- 1 9dd000 -> scalibr
```

```
$ tar -tf exploit-correct-aslr.tar
../../proc/self/map_files/7f633b6b5000
var/lib/rpm/rpmdb.sqlite
```

```
// skip file if it already exists
if _, err = os.Lstat(fullPath); err == nil {
    continue
}
// write file, return early if write fails
err = os.WriteFile(fullPath, content, header.FileInfo())
if err != nil {
    return nil, fmt.Errorf("failed to write file: %w",
}
```

Conditional corruption

```
$ ls -la /proc/`pidof scalibr`/map_files
total 0
dr-x----- 2 .
dr-xr-xr-x 9 ..
lr----- 1 16d6000 -> scalibr
lr----- 1 24fe000 -> scalibr
lr----- 1 2501000 -> scalibr
lr----- 1 4000000 -> scalibr
lr----- 1 7f633b6b5000 -> libc.so.6
lr----- 1 7f633b6dd000 -> libc.so.6
lr----- 1 7f633b842000 -> libc.so.6
lr----- 1 7f633b898000 -> libc.so.6
lr----- 1 7f633b89c000 -> libc.so.6
lr----- 1 7f633b8c9000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ca000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8f2000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8fd000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ff000 -> ld-linux-x86-64.so.2
lr----- 1 9dd000 -> scalibr
```

```
$ tar -tf exploit-correct-aslr.tar
../../proc/self/map_files/7f633b6b5000
var/lib/rpm/rpmdb.sqlite
```

```
// skip file if it already exists
if _, err = os.Lstat(fullPath); err == nil {
    continue
}
// write file, return early if write fails
err = os.WriteFile(fullPath, content, header.FileInfo())
if err != nil {
    return nil, fmt.Errorf("failed to write file: %w",
}
```

Conditional corruption

```
$ ls -la /proc/`pidof scalibr`/map_files
total 0
dr-x----- 2 .
dr-xr-xr-x 9 ..
lr----- 1 16d6000 -> scalibr
lr----- 1 24fe000 -> scalibr
lr----- 1 2501000 -> scalibr
lr----- 1 4000000 -> scalibr
lr----- 1 7f633b6b5000 -> libc.so.6
lr----- 1 7f633b6dd000 -> libc.so.6
lr----- 1 7f633b842000 -> libc.so.6
lr----- 1 7f633b898000 -> libc.so.6
lr----- 1 7f633b89c000 -> libc.so.6
lr----- 1 7f633b8c9000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ca000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8f2000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8fd000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ff000 -> ld-linux-x86-64.so.2
lr----- 1 9dd000 -> scalibr
```

```
$ tar -tf exploit-correct-aslr.tar
../../proc/self/map_files/7f633b6b5000
var/lib/rpm/rpmdb.sqlite
```

```
// skip file if it already exists
if _, err = os.Lstat(fullPath); err == nil {
    continue
}
// write file, return early if write fails
err = os.WriteFile(fullPath, content, header.FileInfo())
if err != nil {
    return nil, fmt.Errorf("failed to write file: %w",
}
```

← skips path if we are right and continues on to the next entry (our exploit!)

Conditional corruption

```
$ ls -la /proc/`pidof scalibr`/map_files
total 0
dr-x----- 2 .
dr-xr-xr-x 9 ..
lr----- 1 16d6000 -> scalibr
lr----- 1 24fe000 -> scalibr
lr----- 1 2501000 -> scalibr
lr----- 1 4000000 -> scalibr
lr----- 1 7f633b6b5000 -> libc.so.6
lr----- 1 7f633b6dd000 -> libc.so.6
lr----- 1 7f633b842000 -> libc.so.6
lr----- 1 7f633b898000 -> libc.so.6
lr----- 1 7f633b89c000 -> libc.so.6
lr----- 1 7f633b8c9000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ca000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8f2000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8fd000 -> ld-linux-x86-64.so.2
lr----- 1 7f633b8ff000 -> ld-linux-x86-64.so.2
lr----- 1 9dd000 -> scalibr
```

```
$ tar -tf exploit-correct-aslr.tar
../../proc/self/map_files/7f633b6b5000
var/lib/rpm/rpmdb.sqlite
```

```
// skip file if it already exists
if _, err = os.Lstat(fullPath); err == nil {
    continue
}
// write file, return early if write fails
err = os.WriteFile(fullPath, content, header.FileInfo())
if err != nil {
    return nil, fmt.Errorf("failed to write file: %w",
}
```

← skips path if we are right and continues on to the next entry (our exploit!)

With this technique, we have a reliable **method of guessing ASLR without crashing** our target service!



Exploiting SQLite



SQLite version

latest version

3.49.1

>

version used by target

3.47.2

Status Of Recent SQLite CVEs

Though the SQLite developers do not consider CVEs to be a reliable source of information about bugs in SQLite, they recognize that many groups, and especially small teams working at the bottom of tall bureaucracies, sometimes need to track CVEs, whether they are useful or not. To aid in this chore, the following table of recent CVEs affecting SQLite is provided.

If you notice new CVEs associated with SQLite that are not in the table below, please bring them to the attention of the developers on the [SQLite Forum](#) so they can be added.

CVE Number	Fix	Comments
CVE-2024-0232	3.43.2 (2023-10-10)	An attacker that can inject arbitrary SQL statements into an application might be able to provoke a use-after-free bug in SQLite's JSON parser that can (in theory) lead to an application crash and denial of service. See forum thread b25edc1d4662 for the bug report.
CVE-2023-32697	Not a bug in SQLite	This is a bug in the SQLite JDBC library, which is a wrapper library that provides access to SQLite from Java. SQLite JDBC is created and maintained independently from SQLite. Despite the use of "SQLite" in the name, the SQLite JDBC library is not affiliated with the SQLite project in any way. The vulnerability identified by this CVE is in the separate SQLite JDBC wrapper library and does not affect SQLite itself.
CVE-2023-39939	Not a bug in SQLite	This is not a bug in SQLite. This is an SQL injection bug in an application (LuxCal Web Calendar) that links against SQLite. Even though this CVE is not about SQLite, "SQLite" is mentioned in the description and so we list it here.
CVE-2023-39543	Not a bug in SQLite	This is not a bug in SQLite. This is an XSS vulnerability in a separate application (LuxCal Web Calendar) that links against SQLite. The bug is in the application, not in SQLite. However "SQLite" is mentioned in the description and so we list it here.
CVE-2023-7104	3.43.1 (2023-09-11)	This is a bug in the session extension of SQLite, not in the SQLite core. This bug is only reachable by applications that recompile SQLite using the <code>-DSQLITE_ENABLE_SESSION</code> compile-time option and then use the Session C-language APIs to process a changeset that has been subtly corrupted by an adversary. So this bug probably does not apply to you. See forum post f935c4708dd528d9 for additional information.
CVE-2022-46908	Not a bug in the core SQLite library	This is a bug in the <code>--safe</code> command-line option of the command-line shell program that is available for accessing SQLite database files. The bug does not exist in the SQLite library. Nor is it an issue for the CLI as long as the user does not depend on the <code>--safe</code> option. It is not serious. It is debatable whether or not this is a security issue.
CVE-2022-38627	Not a bug in SQLite	This is not a bug in SQLite. This is an SQL injection bug in a specific PHP application. In other words, the bug is in the PHP application code, not in SQLite. Even though this CVE is not about SQLite, "SQLite" is mentioned in the publicity about the bug and so we list it here.
CVE-2022-35737	3.39.2 (2022-07-21)	This bug is an array-bounds overflow. The bug is only accessible when using some of the C-language APIs provided by SQLite. The bug cannot be reached using SQL nor can it be reached by providing SQLite with a corrupt database file. The bug only comes up when very long string inputs (greater than 2 billion bytes in length) are provided as arguments to a few specific C-language interfaces, and even then only under special circumstances.

CVEs

Status Of Recent SQLite CVEs

Though the SQLite developers do not consider CVEs to be a reliable source of information about bugs in SQLite, they recognize that many groups, and especially small teams working at the bottom of tall bureaucracies, sometimes need to track CVEs, whether they are useful or not. To aid in this chore, the following table of recent CVEs affecting SQLite is provided.

If you notice new CVEs associated with SQLite that are not in the table below, please bring them to the attention of the developers on the [SQLite Forum](#) so they can be added.

CVE Number	Fix	Comments
CVE-2024-0232	3.43.2 (2023-10-10)	An attacker that can inject arbitrary SQL statements into an application might be able to provoke a use-after-free bug in SQLite's JSON parser that can (in theory) lead to an application crash and denial of service. See forum thread b25edc1d4662 for the bug report.
CVE-2023-32697	Not a bug in SQLite	This is a bug in the SQLite JDBC library, which is a wrapper library that provides access to SQLite from Java. SQLite JDBC is created and maintained independently and is not affiliated with the SQLite project in any way. The vulnerability identified in this CVE does not affect SQLite itself.
CVE-2023-39939	Not a bug in SQLite	This is not a bug in SQLite, "SQLite" (as used in the title) is a trademark (and/or copyright) that links against SQLite. Even though this CVE is not about SQLite, it is listed here.
CVE-2023-39543	Not a bug in SQLite	This is not a bug in SQLite, it is a bug in an application, not in SQLite. The application is mentioned in the SQLite Calendar that links against SQLite. The bug is in the application, not in SQLite.
CVE-2023-7104	3.43.1 (2023-09-11)	This is a bug in the session extension of SQLite, not in the SQLite core. This bug is only reachable by applications that recompile SQLite using the <code>-DSQLITE_ENABLE_SESSION</code> compile-time option and then use the Session C-language APIs to process a changeset that has been subtly corrupted by an adversary. So this bug probably does not apply to you. See forum post f935c4708dd528d9 for additional information.
CVE-2022-46908	Not a bug in the core SQLite library	This is a bug in the <code>--safe</code> command-line option of the command-line shell program that is available for accessing SQLite database files. The bug does not exist in the SQLite library. Nor is it an issue for the CLI as long as the user does not depend on the <code>--safe</code> option. It is not serious. It is debatable whether or not this is a security issue.
CVE-2022-38627	Not a bug in SQLite	This is not a bug in SQLite. This is an SQL injection bug in a specific PHP application. In other words, the bug is in the PHP application code, not in SQLite. Even though this CVE is not about SQLite, "SQLite" is mentioned in the publicity about the bug and so we list it here.
CVE-2022-35737	3.39.2 (2022-07-21)	This bug is an array-bounds overflow. The bug is only accessible when using some of the C-language APIs provided by SQLite. The bug cannot be reached using SQL nor can it be reached by providing SQLite with a corrupt database file. The bug only comes up when very long string inputs (greater than 2 billion bytes in length) are provided as arguments to a few specific C-language interfaces, and even then only under special circumstances.

CVE-2024-0232
3.43.2 (2023-10-10)

Changelog

2025-01-14 (3.48.0)

- Refactor the "configure" script used to help build SQLite from [canonical sources](#), to fix bugs, improve performance, and make the code more maintainable.
 - This does not affect the "configure" script in the `sqlite3-autoconf-NNNNNNN.tar.gz` "amalgamation tarball", only the [canonical sources](#). The build system for the amalgamation tarball is unchanged. If you are using the amalgamation tarball, nothing about this change log entry applies to you.
 - The key innovation here is that [Autosetup](#) is now used instead of [GNU Autoconf](#). That seems like a big change, but it is really just an implementation detail. The `./configure` script is coded very differently, but should work the same as before.
 - One advantage of the new configure is that you no longer need to install [TCL](#) in order to build most SQLite targets. TCL is still required to run tests or to build accessory programs (like [sqlite3_analyzer](#)) that use TCL, but it is not required for most common targets. Hence, as of this release, the only build dependencies are a C compiler and "make" or "nmake".
- Improved [EXPLAIN QUERY PLAN](#) output for [covering indexes](#).
- Allow a two-argument version of the [iif\(\)](#) [SQL function](#). Also allow [iif\(\)](#) as an alternative spelling for [iif\(\)](#).
- Add the `-.dbtotxt` command to the [CLI](#).
- Add the [SQLITE_IOC_CAP_SUBPAGE_READ](#) property to the `xDeviceCharacteristics` method of the [sqlite3_io_methods](#) object.
- Add the [SQLITE_PREPARE_DONT_LOG](#) option to [sqlite3_prepare_v3\(\)](#) that prevents warning messages being sent to the [error log](#) if the SQL is ill-formed. This allows `sqlite3_prepare_v3()` to be used to do test compiles of SQL to check for validity without polluting the error log with false messages.
- Increase the minimum allowed value of [SQLITE_LIMIT_LENGTH](#) from 1 to 30.
- Added the [SQLITE_FCNTL_NULL_IO](#) file control.
- Extend the FTS5 auxiliary API `xInstToken()` to work with prefix queries via the [insttoken](#) configuration option and the [fts5_insttoken\(\)](#) [SQL function](#).
- Increase the [maximum number of arguments to an SQL function](#) from 127 to 1000.
- Remove vestigial traces of `SQLITE_USER_AUTHENTICATION`.
- Various obscure bug fixes.

Hashes:

- `SQLITE_SOURCE_ID`: 2025-01-14 11:05:00 d2fe6b05f38d9d7cd78c5d252e99ac59f1aea071d669830c1ffe4e8966e84010
- SHA3-256 for `sqlite3.c`: 522bd9492526147d12fc8521028eb960557353d8f08c2035f15ac737d4b91e21

2024-12-07 (3.47.2)

- Fix a problem in text-to-floating-point conversion for SQLite that can cause values between `'1.8446744073709550592eNNN'` and `'1.8446744073709551609eNNN'` for any exponent NNN to be rendered incorrectly. In other words, some numeric text values where the first 16 significant digits are `'1844674407370955'` might be converted into the wrong floating-point value. See [forum thread 569a7209179a7f5e](#). This problem only arises on x64 and i386 hardware. The problem was introduced in 3.47.0.
- Other minor bug fixes.

Hashes:

- `SQLITE_SOURCE_ID`: 2024-12-07 20:39:59 2aabe05e2e8cae4847a802ee2daddc1d7413d8fc560254d93ee3e72c14685b6c
- SHA3-256 for `sqlite3.c`: 586b5789453ee5c51e8d858cd836f20ce60df882323f2070547cbaf23a5898af

Changelog

2025-01-14 (3.48.0)

1. Refactor the "configure" script used to help build SQLite from [canonical sources](#), to fix bugs, improve performance, and make the code more maintainable.
 - This does not affect the "configure" script in the `sqlite3-autoconf-NNNNNNN.tar.gz` "amalgamation tarball", only the [canonical sources](#). The build system for the amalgamation tarball is unchanged. If you are using the amalgamation tarball, nothing about this change log entry applies to you.
 - The key innovation here is that [Autosetup](#) is now used instead of [GNU Autoconf](#). That seems like a big change, but it is really just an implementation detail. The `./configure` script is coded very differently, but should work the same as before.
 - One advantage of the new configure is that you no longer need to install [TCL](#) in order to build most SQLite targets. TCL is still required to run tests or to build accessory programs (like [sqlite3_analyzer](#)) that use TCL, but it is not required for most common targets. Hence, as of this release, the only build dependencies are a C compiler and "make" or "nmake".
2. Improved [EXPLAIN QUERY PLAN](#) output for [covering indexes](#).
3. Allow a two-argument version of the [iif\(\)](#) [SQL function](#). Also allow [iif\(\)](#) as an alternative spelling for [iiff\(\)](#).
4. Add the `-.dbtotxt` command to the [CLI](#).
5. Add the [SQLITE_IOCAP_SUBPAGE_READ](#) property to the `xDeviceCharacteristics` method of the [sqlite3_io_methods](#) object.
6. Add the [SQLITE_PREPARE_DONT_LOG](#) option to [sqlite3_prepare_v3\(\)](#) that prevents warning messages being sent to the [error log](#) if the SQL is ill-formed. This allows `sqlite3_prepare_v3()` to be used to do test compiles of SQL to check for validity without polluting the error log with false messages.
7. Increase the minimum allowed value of [SQLITE_LIMIT_LENGTH](#) from 1 to 30.
8. Added the [SQLITE_FCNTL_NULL_IO](#) file control.
9. Extend the FTS5 auxiliary API `xInstToken()` to work with prefix queries via the [insttoken](#) configuration option and the [fts5_insttoken\(\)](#) [SQL function](#).
10. Increase the [maximum number of arguments to an SQL function](#) from 127 to 1000.
11. Remove vestigial traces of `SQLITE_USER_AUTHENTICATION`.
12. Various obscure bug fixes.

Hashes:

13. `SQLITE_SOURCE_ID`: 2025-01-14 11:05:00 d2fe6b05f38d9d7cd78c5d252e99ac59f1aea071d669830c1ffe4e8966e84010
14. SHA3-256 for `sqlite3.c`: 522bd9492526147d12fc8521028eb960557353d8f08c2035f15ac737d4b91e21

2024-12-07 (3.47.2)

1. Fix a problem in text-to-floating-point conversion for SQLite that can cause values between `'1.8446744073709550592eNNN'` and `'1.8446744073709551609eNNN'` for any exponent NNN to be rendered incorrectly. In other words, some numeric text values where the first 16 significant digits are `'1844674407370955'` might be converted into the wrong floating-point value. See [forum thread 569a7209179a7f5e](#). This problem only arises on x64 and i386 hardware. The problem was introduced in 3.47.0.
2. Other minor bug fixes.

Hashes:

3. `SQLITE_SOURCE_ID`: 2024-12-07 20:39:59 2aabe05e2e8cae4847a802ee2daddc1d7413d8fc560254d93ee3e72c14685b6c
4. SHA3-256 for `sqlite3.c`: 586b5789453ee5c51e8d858cd836f20ce60df882323f2070547cbaf23a5898af

Changelog

2025-02-18 (3.49.1)

1. Improve portability of makefiles and configure scripts.
2. Fix a bug in the [concat_ws\(\)](#) function, introduced in [version 3.44.0](#), that could lead to a memory error, specifically a write past the end of allocated space, if the separator string is larger than two megabytes.
3. Enhanced the [SQLITE_DBCONFIG_LOOKASIDE](#) interface to make it more robust against misuse.

Hashes:

4. `SQLITE_SOURCE_ID`: 2025-02-18 13:38:58 873d4e274b4988d260ba8354a9718324a1c26187a4ab4c1cc0227c03d0f10e70
5. SHA3-256 for `sqlite3.c`: 22eab05c842463a3705c63922f12f68531200185ced510ed6aed2065ca44e816

2025-02-06 (3.49.0)

1. Enhancements to the [query planner](#):
 - a. Improve the [query-time index](#) optimization so that it works on [WITHOUT ROWID](#) tables.
 - b. Better query plans for large [star-query](#) joins. This fixes three different performance regressions that were reported on the SQLite Forum.
 - c. When two or more queries have the same estimated cost, use the one with the fewer bytes per row.
2. Enhance the [if\(\) SQL function](#) so that it can accept any number of arguments greater than or equal to two.
3. Enhance the [session extension](#) so that it works on databases that make use of [generated columns](#).
4. Omit the `SQLITE_USE_STDIO_FOR_CONSOLE` compile-time option which was not implemented correctly and never worked right. In its place add the `SQLITE_USE_W32_FOR_CONSOLE_IO` compile-time option. This option applies to command-line tools like the [CLI](#) only, not to the SQLite core. It causes Win32 APIs to be used for console I/O instead of `stdio`. This option affects Windows builds only.
5. Three new options to [sqlite3_db_config\(\)](#). All default to "on".
 - a. [SQLITE_DBCONFIG_ENABLE_ATTACH_CREATE](#)
 - b. [SQLITE_DBCONFIG_ENABLE_ATTACH_WRITE](#)
 - c. [SQLITE_DBCONFIG_ENABLE_COMMENTS](#)
6. Replace [Autotools](#) with [Autosetup](#) for the configure script used in the [precompiled amalgamation tarball](#). The configure script for the [canonical source code](#) was changed to `Autosetup` in the previous (3.48.0) release. Only the main SQLite configure script in the amalgamation tarball is changed. The (deprecated) configuration script use by [TEA](#) subdirectory of the amalgamation tarball still relies on `Autotools`.
7. Various minor patches and fixes for problems seen in the 3.48.0 release.

Changelog

2025-02-18 (3.49.1)

1. Improve portability of makefiles and configure scripts.
2. Fix a bug in the [concat_ws\(\)](#) function, introduced in [version 3.44.0](#), that could lead to a memory error, specifically a write past the end of allocated space, if the separator string is larger than two megabytes.
3. Enhanced the [SQLITE_DBCONFIG_LOOKASIDE](#) interface to make it more robust against misuse.

Hashes:

4. `SQLITE_SOURCE_ID`: 2025-02-18 13:38:58 873d4e274b4988d260ba8354a9718324a1c26187a4ab4c1cc0227c03d0f10e70
5. SHA3-256 for `sqlite3.c`: 22eab05c842463a3705c63922f12f68531200185ced510ed6aed2065ca44e816

2025-02-06 (3.49.0)

1. Enhancements to the [query planner](#):
 - a. Improve the [query-time index](#) optimization so that it works on [WITHOUT ROWID](#) tables.
 - b. Better query plans for large [star-query](#) joins. This fixes three different performance regressions that were reported on the SQLite Forum.
 - c. When two or more queries have the same estimated cost, use the one with the fewer bytes per row.
2. Enhance the [if\(\) SQL function](#) so that it can accept any number of arguments greater than or equal to two.
3. Enhance the [session extension](#) so that it works on databases that make use of [generated columns](#).
4. Omit the `SQLITE_USE_STDIO_FOR_CONSOLE` compile-time option which was not implemented correctly and never worked right. In its place add the `SQLITE_USE_W32_FOR_CONSOLE_IO` compile-time option. This option applies to command-line tools like the [CLI](#) only, not to the SQLite core. It causes Win32 APIs to be used for console I/O instead of `stdio`. This option affects Windows builds only.
5. Three new options to [sqlite3_db_config\(\)](#). All default to "on".
 - a. [SQLITE_DBCONFIG_ENABLE_ATTACH_CREATE](#)
 - b. [SQLITE_DBCONFIG_ENABLE_ATTACH_WRITE](#)
 - c. [SQLITE_DBCONFIG_ENABLE_COMMENTS](#)
6. Replace [Autotools](#) with [Autosetup](#) for the configure script used in the [precompiled amalgamation tarball](#). The configure script for the [canonical source code](#) was changed to `Autosetup` in the previous (3.48.0) release. Only the main SQLite configure script in the amalgamation tarball is changed. The (deprecated) configuration script use by [TEA](#) subdirectory of the amalgamation tarball still relies on `Autotools`.
7. Various minor patches and fixes for problems seen in the 3.48.0 release.

Changelog

2025-02-18 (3.49.1)

1. Improve portability of makefiles and configure scripts.
2. Fix a bug in the `concat_ws()` function, introduced in [version 3.44.0](#), that could lead to a memory error, specifically a write past the end of allocated space, if the separator string is larger than two megabytes.
3. Enhanced the `SQLITE_DBCONFIG_LOOKASIDE` interface to make it more robust against misuse.

Hashes:

4. SQ
5. SH

... could lead to a memory error, specifically a write past the end of allocated space, ...

2025-

1. Enhance the `if()` [SQL function](#) so that it can accept any number of arguments greater than or equal to two.
2. Enhance the [session extension](#) so that it works on databases that make use of [generated columns](#).
3. Omit the `SQLITE_USE_STDIO_FOR_CONSOLE` compile-time option which was not implemented correctly and new `SQLITE_USE_W32_FOR_CONSOLE_IO` compile-time option. This option applies to command-line tools like the `CLI` and `SQLITE_CLI` in the SQLite core. It causes Win32 APIs to be used for console I/O instead of `stdio`. This option affects Windows builds only.
4. Three new options to `sqlite3_db_config()`. All default to "on".
 - a. `SQLITE_DBCONFIG_ENABLE_ATTACH_CREATE`
 - b. `SQLITE_DBCONFIG_ENABLE_ATTACH_WRITE`
 - c. `SQLITE_DBCONFIG_ENABLE_COMMENTS`
5. Replace `Autotools` with `Autosetup` for the configure script used in the [precompiled amalgamation tarball](#). The configure script for the [canonical source code](#) was changed to `Autosetup` in the previous (3.48.0) release. Only the main SQLite configure script in the amalgamation tarball is changed. The (deprecated) configuration script use by [TEA](#) subdirectory of the amalgamation tarball still relies on `Autotools`.
6. Various minor patches and fixes for problems seen in the 3.48.0 release.



Commit

```
src/func.c @@ -1570,7 +1570,7 @@ static void concatFuncCore(  
    1570     1570     for(i=0; i<argc; i++){  
    1571     1571         n += sqlite3_value_bytes(argv[i]);  
    1572     1572     }  
    1573     - n += (argc-1)*nSep;  
    1573     + n += (argc-1)*(i64)nSep;  
    1574     1574     z = sqlite3_malloc64(n+1);  
    1575     1575     if( z==0 ){  
    1576     1576         sqlite3_result_error_nomem(context);  
    ...  
    ↓
```

1) <https://github.com/sqlite/sqlite/commit/f4fc2ee20311a0a5141726c71d318ab52001c974>

concat_ws

```
static void concatFuncCore(  
    sqlite3_context *context,  
    int argc,  
    sqlite3_value **argv,  
    int nSep,  
    const char *zSep  
)  
{  
    i64 j, k, n = 0;  
    int i;  
    char *z;  
    for(i=0; i<argc; i++){  
        n += sqlite3_value_bytes(argv[i]);  
    }  
    n += (argc-1)*nSep;  
    z = sqlite3_malloc64(n+1);  
    if( z==0 ){  
        sqlite3_result_error_nomem(context);  
        return;  
    }  
    j = 0;  
    for(i=0; i<argc; i++){  
        k = sqlite3_value_bytes(argv[i]);  
        if( k>0 ){  
            const char *v = (const char*)sqlite3_value_text(argv[i]);  
            if( v!=0 ){  
                if( j>0 && nSep>0 ){  
                    memcpy(&z[j], zSep, nSep);  
                    j += nSep;  
                }  
                memcpy(&z[j], v, k);  
                j += k;  
            }  
        }  
    }  
}
```

concatFuncCore

implements
concat_ws

concat_ws

```
static void concatFuncCore(  
    sqlite3_context *context,  
    int argc,  
    sqlite3_value **argv,  
    int nSep,  
    const char *zSep  
)  
{  
    i64 j, k, n = 0;  
    int i;  
    char *z;  
    for(i=0; i<argc; i++){  
        n += sqlite3_value_bytes(argv[i]);  
    }  
    n += (argc-1)*nSep;  
    z = sqlite3_malloc0(n+1);  
    if( z==0 ){  
        sqlite3_result_error_nomem(context);  
        return;  
    }  
    j = 0;  
    for(i=0; i<argc; i++){  
        k = sqlite3_value_bytes(argv[i]);  
        if( k>0 ){  
            const char *v = (const char*)sqlite3_value_text(argv[i]);  
            if( v!=0 ){  
                if( j>0 && nSep>0 ){  
                    memcpy(&z[j], zSep, nSep);  
                    j += nSep;  
                }  
                memcpy(&z[j], v, k);  
                j += k;  
            }  
        }  
    }  
}
```

n += (argc-1)*nSep;

concat_ws

```
concat_ws(separator, str1, str2, ...);
```

concat_ws

```
concat_ws(separator, str1, str2, ...);
```

```
SELECT concat_ws(':', 'aa', 'bb', 'cc');
```

concat_ws

```
concat_ws(separator, str1, str2, ...);
```

```
SELECT concat_ws(':', 'aa', 'bb', 'cc');
```



```
aa:bb:cc
```

concat_ws

```
concat_ws(separator, str1, str2, ...);
```

```
SELECT concat_ws(':', 'aa', 'bb', 'cc');
```



aa:bb:cc



size?

concat_ws

```
concat_ws(separator, str1, str2, ...);
```

```
SELECT concat_ws(':', 'aa', 'bb', 'cc');
```

2 + 2 + 2

aa:bb:cc

concat_ws

```
concat_ws(separator, str1, str2, ...);
```

```
SELECT concat_ws(':', 'aa', 'bb', 'cc');
```

$(3-1) * 1 + 2 + 2 + 2$

aa:bb:cc

concat_ws

```
concat_ws(separator, str1, str2, ...);
```

```
SELECT concat_ws(':', 'aa', 'bb', 'cc');
```

$(3-1) * 1 + 2 + 2 + 2$

aa:bb:cc

8



concat_ws

```
static void concatFuncCore(  
    sqlite3_context *context,  
    int argc,  
    sqlite3_value **argv,  
    int nSep,  
    const char *zSep  
) {  
    i64 j, k, n = 0;  
    int i;  
    char *z;  
    for(i=0; i<argc; i++){  
        n += sqlite3_value_bytes(argv[i]);  
    }  
    n += (argc-1)*nSep;  
    z = sqlite3_malloc64(n+1);  
    if( z==0 ){  
        sqlite3_result_error_nomem(context);  
        return;  
    }  
    j = 0;  
    for(i=0; i<argc; i++){  
        k = sqlite3_value_bytes(argv[i]);  
        if( k>0 ){  
            const char *v = (const char*)sqlite3_value_text(argv[i]);  
            if( v!=0 ){  
                if( j>0 && nSep>0 ){  
                    memcpy(&z[j], zSep, nSep);  
                    j += nSep;  
                }  
                memcpy(&z[j], v, k);  
                j += k;  
            }  
        }  
    }  
}
```

concat_ws

```
static void concatFuncCore(  
    sqlite3_context *context,  
    int argc,  
    sqlite3_value **argv,  
    int nSep,  
    const char *zSep  
) {  
    i64 j, k, n = 0;  
    int i;  
    char *z;  
    for(i=0; i<argc; i++){  
        n += sqlite3_value_bytes(argv[i]);  
    }  
    n += (argc-1)*nSep;  
    z = sqlite3_malloc64(n+1);  
    if( z==0 ){  
        sqlite3_result_error_nomem(context);  
        return;  
    }  
    j = 0;  
    for(i=0; i<argc; i++){  
        k = sqlite3_value_bytes(argv[i]);  
        if( k>0 ){  
            const char *v = (const char*)sqlite3_value_text(argv[i]);  
            if( v!=0 ){  
                if( j>0 && nSep>0 ){  
                    memcpy(&z[j], zSep, nSep);  
                    j += nSep;  
                }  
                memcpy(&z[j], v, k);  
                j += k;  
            }  
        }  
    }  
}
```

i64 j, k, n = 0;

concat_ws

```
static void concatFuncCore(  
    sqlite3_context *context,  
    int argc,  
    sqlite3_value **argv,  
    int nSep,  
    const char *zSep  
) {  
    i64 j, k, n = 0;  
    int i;  
    char *z;  
    for(i=0; i<argc; i++){  
        n += sqlite3_value_bytes(argv[i]);  
    }  
    n += (argc-1)*nSep;  
    z = sqlite3_malloc64(n+1);  
    if( z==0 ){  
        sqlite3_result_error_nomem(context);  
        return;  
    }  
    j = 0;  
    for(i=0; i<argc; i++){  
        k = sqlite3_value_bytes(argv[i]);  
        if( k>0 ){  
            const char *v = (const char*)sqlite3_value_text(argv[i]);  
            if( v!=0 ){  
                if( j>0 && nSep>0 ){  
                    memcpy(&z[j], zSep, nSep);  
                    j += nSep;  
                }  
                memcpy(&z[j], v, k);  
                j += k;  
            }  
        }  
    }  
}
```

```
for(i=0; i<argc; i++){  
    n += sqlite3_value_bytes(argv[i]);  
}
```

concat_ws

```
static void concatFuncCore(  
    sqlite3_context *context,  
    int argc,  
    sqlite3_value **argv,  
    int nSep,  
    const char *zSep  
)  
{  
    i64 j, k, n = 0;  
    int i;  
    char *z;  
    for(i=0; i<argc; i++){  
        n += sqlite3_value_bytes(argv[i]);  
    }  
    n += (argc-1)*nSep;  
    z = sqlite3_malloc0(n+1);  
    if( z==0 ){  
        sqlite3_result_error_nomem(context);  
        return;  
    }  
    j = 0;  
    for(i=0; i<argc; i++){  
        k = sqlite3_value_bytes(argv[i]);  
        if( k>0 ){  
            const char *v = (const char*)sqlite3_value_text(argv[i]);  
            if( v!=0 ){  
                if( j>0 && nSep>0 ){  
                    memcpy(&z[j], zSep, nSep);  
                    j += nSep;  
                }  
                memcpy(&z[j], v, k);  
                j += k;  
            }  
        }  
    }  
}
```

`n += (argc-1)*nSep;`

Integer Overflow

```

static void concatFuncCore(
    sqlite3_context *context,
    int argc,
    sqlite3_value **argv,
    int nSep,
    const char *zSep
){
    i64 j, k, n = 0;
    int i;
    char *z;
    for(i=0; i<argc; i++){
        n += sqlite3_value_bytes(argv[i]);
    }
    n += (argc-1)*nSep;
    z = sqlite3_malloc64(n+1);
    if( z==0 ){
        sqlite3_result_error_nomem(context);
        return;
    }
    j = 0;
    for(i=0; i<argc; i++){
        k = sqlite3_value_bytes(argv[i]);
        if( k>0 ){
            const char *v = (const char*)sqlite3_value_text(argv[i]);
            if( v!=0 ){
                if( j>0 && nSep>0 ){
                    memcpy(&z[j], zSep, nSep);
                    j += nSep;
                }
                memcpy(&z[j], v, k);
                j += k;
            }
        }
    }
}

```

The diagram illustrates the integer overflow calculation $n += (argc-1) * nSep;$. Red dashed arrows from the code point to the variables in the equation: `int argc` points to `argc`, `int nSep` points to `nSep`, and `i64 j, k, n = 0;` points to `n`. Below the equation, the types are labeled: `n` is `i64` (blue), `argc-1` is `int` (yellow), and `nSep` is `int` (yellow).

Integer Overflow

```

static void concatFuncCore(
    sqlite3_context *context,
    int argc,
    sqlite3_value **argv,
    int nSep,
    const char *zSep
){
    i64 j, k, n = 0;
    int i;
    char *z;
    for(i=0; i<argc; i++){
        n += sqlite3_value_bytes(argv[i]);
    }
    n += (argc-1)*nSep;
    z = sqlite3_malloc64(n+1);
    if( z==0 ){
        sqlite3_result_error_nomem(context);
        return;
    }
    j = 0;
    for(i=0; i<argc; i++){
        k = sqlite3_value_bytes(argv[i]);
        if( k>0 ){
            const char *v = (const char*)sqlite3_value_text(argv[i]);
            if( v!=0 ){
                if( j>0 && nSep>0 ){
                    memcpy(&z[j], zSep, nSep);
                    j += nSep;
                }
                memcpy(&z[j], v, k);
                j += k;
            }
        }
    }
}

```



n +=
i64


(argc-1) * nSep;
int int

Integer Overflow

```

static void concatFuncCore(
    sqlite3_context *context,
    int argc,
    sqlite3_value **argv,
    int nSep,
    const char *zSep
){
    i64 j, k, n = 0;
    int i;
    char *z;
    for(i=0; i<argc; i++){
        n += sqlite3_value_bytes(argv[i]);
    }
    n += (argc-1)*nSep;
    z = sqlite3_malloc64(n+1);
    if( z==0 ){
        sqlite3_result_error_nomem(context);
        return;
    }
    j = 0;
    for(i=0; i<argc; i++){
        k = sqlite3_value_bytes(argv[i]);
        if( k>0 ){
            const char *v = (const char*)sqlite3_value_text(argv[i]);
            if( v!=0 ){
                if( j>0 && nSep>0 ){
                    memcpy(&z[j], zSep, nSep);
                    j += nSep;
                }
                memcpy(&z[j], v, k);
                j += k;
            }
        }
    }
}

```



$$\begin{array}{c}
 \text{i64} \\
 n
 \end{array}
 +=
 \begin{array}{c}
 \text{int} \\
 (\text{argc}-1)
 \end{array}
 *
 \begin{array}{c}
 \text{int} \\
 \text{nSep}
 \end{array}
 ;$$

Integer Overflow

```

static void concatFuncCore(
    sqlite3_context *context,
    int argc,
    sqlite3_value **argv,
    int nSep,
    const char *zSep
){
    i64 j, k, n = 0;
    int i;
    char *z;
    for(i=0; i<argc; i++){
        n += sqlite3_value_bytes(argv[i]);
    }
    n += (argc-1)*nSep;
    z = sqlite3_malloc64(n+1);
    if( z==0 ){
        sqlite3_result_error_nomem(context);
        return;
    }
    j = 0;
    for(i=0; i<argc; i++){
        k = sqlite3_value_bytes(argv[i]);
        if( k>0 ){
            const char *v = (const char*)sqlite3_value_text(argv[i]);
            if( v!=0 ){
                if( j>0 && nSep>0 ){
                    memcpy(&z[j], zSep, nSep);
                    j += nSep;
                }
                memcpy(&z[j], v, k);
                j += k;
            }
        }
    }
}

```

$$n \ += \ (argc-1) * nSep;$$

i64
int
int

32-bit

$$(18-1) * 0x100000000$$

Integer Overflow

```

static void concatFuncCore(
    sqlite3_context *context,
    int argc,
    sqlite3_value **argv,
    int nSep,
    const char *zSep
){
    i64 j, k, n = 0;
    int i;
    char *z;
    for(i=0; i<argc; i++){
        n += sqlite3_value_bytes(argv[i]);
    }
    n += (argc-1)*nSep;
    z = sqlite3_malloc64(n+1);
    if( z==0 ){
        sqlite3_result_error_nomem(context);
        return;
    }
    j = 0;
    for(i=0; i<argc; i++){
        k = sqlite3_value_bytes(argv[i]);
        if( k>0 ){
            const char *v = (const char*)sqlite3_value_text(argv[i]);
            if( v!=0 ){
                if( j>0 && nSep>0 ){
                    memcpy(&z[j], zSep, nSep);
                    j += nSep;
                }
                memcpy(&z[j], v, k);
                j += k;
            }
        }
    }
}

```

n += (
argc-1) *
nSep;

i64 int int

32-bit
 (18-1) * 0x10000000
 = 0x1110000000
32-bit

Integer Overflow

```

static void concatFuncCore(
    sqlite3_context *context,
    int argc,
    sqlite3_value **argv,
    int nSep,
    const char *zSep
){
    i64 j, k, n = 0;
    int i;
    char *z;
    for(i=0; i<argc; i++){
        n += sqlite3_value_bytes(argv[i]);
    }
    n += (argc-1)*nSep;
    z = sqlite3_malloc64(n+1);
    if( z==0 ){
        sqlite3_result_error_nomem(context);
        return;
    }
    j = 0;
    for(i=0; i<argc; i++){
        k = sqlite3_value_bytes(argv[i]);
        if( k>0 ){
            const char *v = (const char*)sqlite3_value_text(argv[i]);
            if( v!=0 ){
                if( j>0 && nSep>0 ){
                    memcpy(&z[j], zSep, nSep);
                    j += nSep;
                }
                memcpy(&z[j], v, k);
                j += k;
            }
        }
    }
}

```

$$n \ += \ (argc-1) * nSep;$$

i64
int
int

32-bit

$$(18-1) * 0x10000000$$

$$= 0x110000000$$

32-bit

$$n \ += \ 0x10000000$$

Too small allocation

```
static void concatFuncCore(  
    sqlite3_context *context,  
    int argc,  
    sqlite3_value **argv,  
    int nSep,  
    const char *zSep  
)  
{  
    i64 j, k, n = 0;  
    int i;  
    char *z;  
    for(i=0; i<argc; i++){  
        n += sqlite3_value_bytes(argv[i]);  
    }  
    n += (argc-1)*nSep;  
    z = sqlite3_malloc64(n+1);  
    if( z==0 ){  
        sqlite3_result_error_nomem(context);  
        return;  
    }  
    j = 0;  
    for(i=0; i<argc; i++){  
        k = sqlite3_value_bytes(argv[i]);  
        if( k>0 ){  
            const char *v = (const char*)sqlite3_value_text(argv[i]);  
            if( v!=0 ){  
                if( j>0 && nSep>0 ){  
                    memcpy(&z[j], zSep, nSep);  
                    j += nSep;  
                }  
                memcpy(&z[j], v, k);  
                j += k;  
            }  
        }  
    }  
}
```

`z = sqlite3_malloc64(n+1);`

Too small allocation

```
static void concatFuncCore(  
    sqlite3_context *context,  
    int argc,  
    sqlite3_value **argv,  
    int nSep,  
    const char *zSep  
)  
{  
    i64 j, k, n = 0;  
    int i;  
    char *z;  
    for(i=0; i<argc; i++){  
        n += sqlite3_value_bytes(argv[i]);  
    }  
    n += (argc-1)*nSep;  
    z = sqlite3_malloc64(n+1);  
    if( z==0 ){  
        sqlite3_result_error_nomem(context);  
        return;  
    }  
    j = 0;  
    for(i=0; i<argc; i++){  
        k = sqlite3_value_bytes(argv[i]);  
        if( k>0 ){  
            const char *v = (const char*)sqlite3_value_text(argv[i]);  
            if( v!=0 ){  
                if( j>0 && nSep>0 ){  
                    memcpy(&z[j], zSep, nSep);  
                    j += nSep;  
                }  
                memcpy(&z[j], v, k);  
                j += k;  
            }  
        }  
    }  
}
```



required space

0x1`10000000 (>4GB)

Too small allocation

```
static void concatFuncCore(  
    sqlite3_context *context,  
    int argc,  
    sqlite3_value **argv,  
    int nSep,  
    const char *zSep  
)  
{  
    i64 j, k, n = 0;  
    int i;  
    char *z;  
    for(i=0; i<argc; i++){  
        n += sqlite3_value_bytes(argv[i]);  
    }  
    n += (argc-1)*nSep;  
    z = sqlite3_malloc64(n+1);  
    if( z==0 ){  
        sqlite3_result_error_nomem(context);  
        return;  
    }  
    j = 0;  
    for(i=0; i<argc; i++){  
        k = sqlite3_value_bytes(argv[i]);  
        if( k>0 ){  
            const char *v = (const char*)sqlite3_value_text(argv[i]);  
            if( v!=0 ){  
                if( j>0 && nSep>0 ){  
                    memcpy(&z[j], zSep, nSep);  
                    j += nSep;  
                }  
                memcpy(&z[j], v, k);  
                j += k;  
            }  
        }  
    }  
}
```

actual allocation

~0x10000000 (256MB)

required space

Too small allocation

```
static void concatFuncCore(
    sqlite3_context *context,
```

```
if( j>0 && nSep>0 ){
    memcpy(&z[j], zSep, nSep);
    j += nSep;
}
memcpy(&z[j], v, k);
```

```
return;
}
j = 0;
for(i=0; i<argc; i++){
    k = sqlite3_value_bytes(argv[i]);
    if( k>0 ){
        const char *v = (const char*)sqlite3_value_text(argv[i]);
        if( v!=0 ){
            if( j>0 && nSep>0 ){
                memcpy(&z[j], zSep, nSep);
                j += nSep;
            }
            memcpy(&z[j], v, k);
            j += k;
        }
    }
}
```

actual allocation

} ~0x10000000 (256MB)

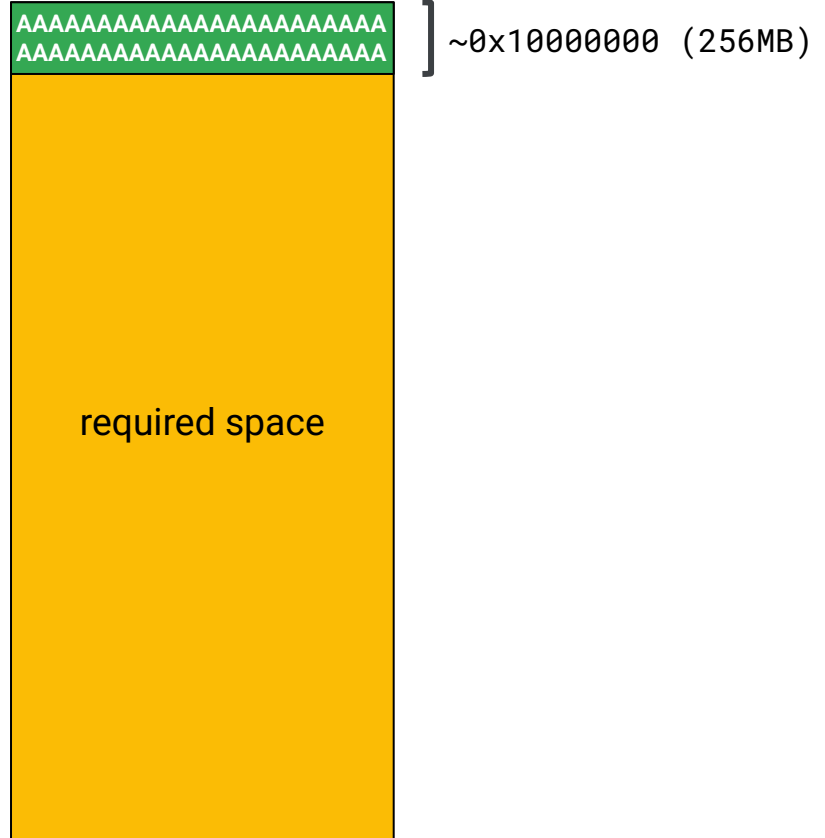
required space

Too small allocation

```

static void concatFuncCore(
    sqlite3_context *context,
    int argc,
    sqlite3_value **argv,
    int nSep,
    const char *zSep
){
    i64 j, k, n = 0;
    int i;
    char *z;
    for(i=0; i<argc; i++){
        n += sqlite3_value_bytes(argv[i]);
    }
    n += (argc-1)*nSep;
    z = sqlite3_malloc64(n+1);
    if( z==0 ){
        sqlite3_result_error_nomem(context);
        return;
    }
    j = 0;
    for(i=0; i<argc; i++){
        k = sqlite3_value_bytes(argv[i]);
        if( k>0 ){
            const char *v = (const char*)sqlite3_value_text(argv[i]);
            if( v!=0 ){
                if( j>0 && nSep>0 ){
                    memcpy(&z[j], zSep, nSep);
                    j += nSep;
                }
                memcpy(&z[j], v, k);
                j += k;
            }
        }
    }
}

```

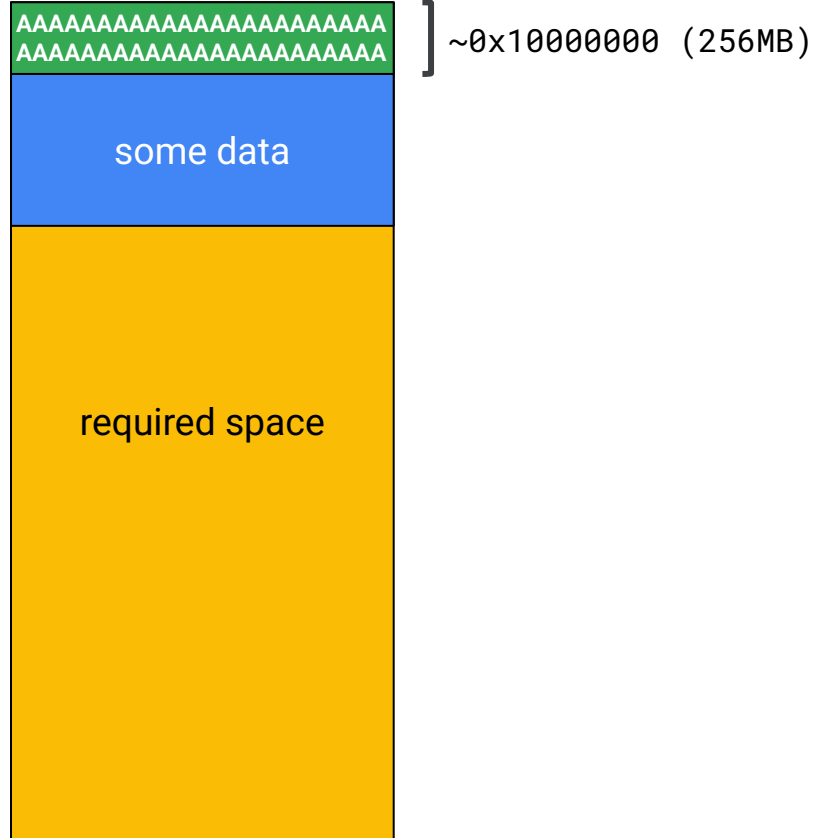


Too small allocation

```

static void concatFuncCore(
    sqlite3_context *context,
    int argc,
    sqlite3_value **argv,
    int nSep,
    const char *zSep
){
    i64 j, k, n = 0;
    int i;
    char *z;
    for(i=0; i<argc; i++){
        n += sqlite3_value_bytes(argv[i]);
    }
    n += (argc-1)*nSep;
    z = sqlite3_malloc64(n+1);
    if( z==0 ){
        sqlite3_result_error_nomem(context);
        return;
    }
    j = 0;
    for(i=0; i<argc; i++){
        k = sqlite3_value_bytes(argv[i]);
        if( k>0 ){
            const char *v = (const char*)sqlite3_value_text(argv[i]);
            if( v!=0 ){
                if( j>0 && nSep>0 ){
                    memcpy(&z[j], zSep, nSep);
                    j += nSep;
                }
                memcpy(&z[j], v, k);
                j += k;
            }
        }
    }
}

```

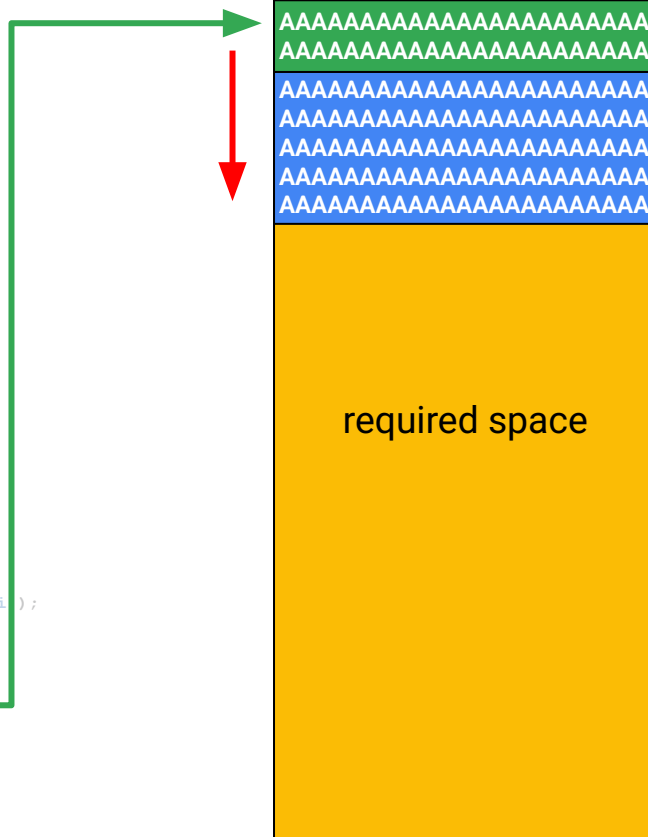


Too small allocation

```

static void concatFuncCore(
    sqlite3_context *context,
    int argc,
    sqlite3_value **argv,
    int nSep,
    const char *zSep
){
    i64 j, k, n = 0;
    int i;
    char *z;
    for(i=0; i<argc; i++){
        n += sqlite3_value_bytes(argv[i]);
    }
    n += (argc-1)*nSep;
    z = sqlite3_malloc64(n+1);
    if( z==0 ){
        sqlite3_result_error_nomem(context);
        return;
    }
    j = 0;
    for(i=0; i<argc; i++){
        k = sqlite3_value_bytes(argv[i]);
        if( k>0 ){
            const char *v = (const char*)sqlite3_value_text(argv[i]);
            if( v!=0 ){
                if( j>0 && nSep>0 ){
                    memcpy(&z[j], zSep, nSep);
                    j += nSep;
                }
                memcpy(&z[j], v, k);
                j += k;
            }
        }
    }
}

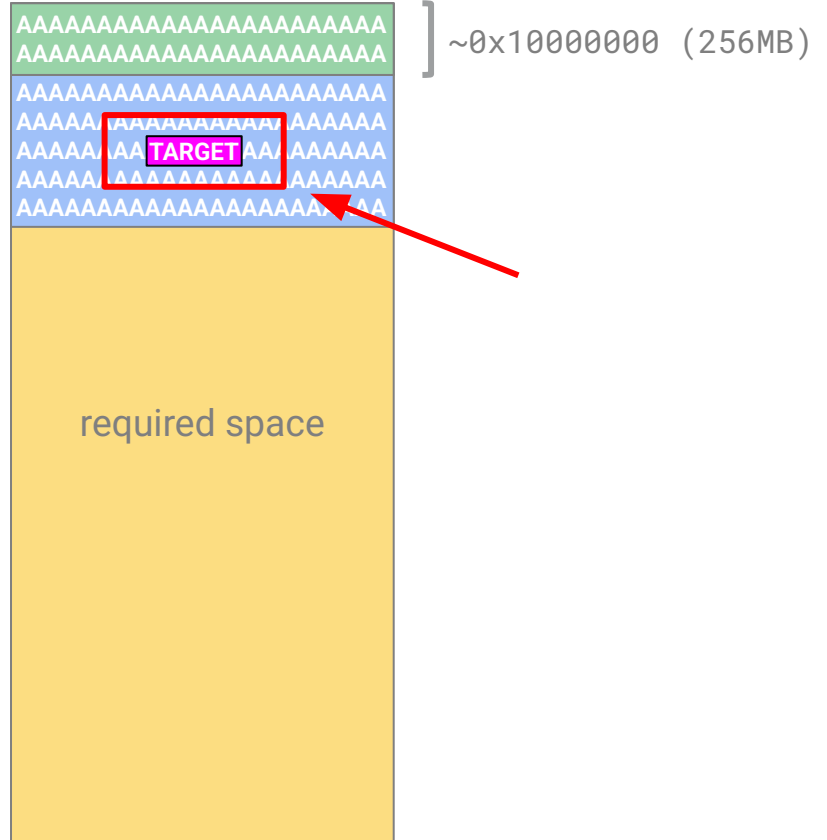
```



~0x10000000 (256MB)

Too small allocation

```
static void concatFuncCore(  
    sqlite3_context *context,  
    int argc,  
    sqlite3_value **argv,  
    int nSep,  
    const char *zSep  
)  
{  
    i64 j, k, n = 0;  
    int i;  
    char *z;  
    for(i=0; i<argc; i++){  
        n += sqlite3_value_bytes(argv[i]);  
    }  
    n += (argc-1)*nSep;  
    z = sqlite3_malloc64(n+1);  
    if( z==0 ){  
        sqlite3_result_error_nomem(context);  
        return;  
    }  
    j = 0;  
    for(i=0; i<argc; i++){  
        k = sqlite3_value_bytes(argv[i]);  
        if( k>0 ){  
            const char *v = (const char*)sqlite3_value_text(argv[i]);  
            if( v!=0 ){  
                if( j>0 && nSep>0 ){  
                    memcpy(&z[j], zSep, nSep);  
                    j += nSep;  
                }  
                memcpy(&z[j], v, k);  
                j += k;  
            }  
        }  
    }  
}
```

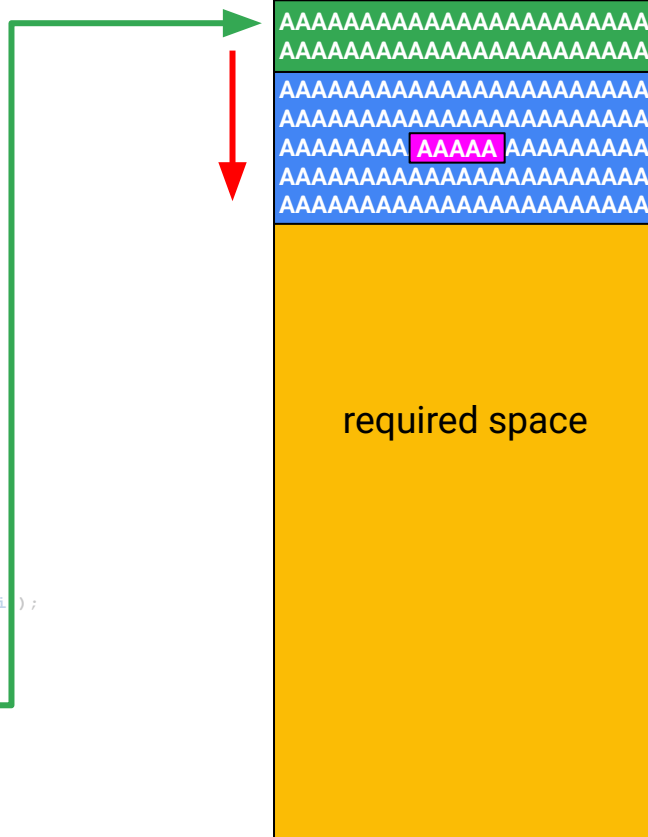


Too small allocation

```

static void concatFuncCore(
    sqlite3_context *context,
    int argc,
    sqlite3_value **argv,
    int nSep,
    const char *zSep
){
    i64 j, k, n = 0;
    int i;
    char *z;
    for(i=0; i<argc; i++){
        n += sqlite3_value_bytes(argv[i]);
    }
    n += (argc-1)*nSep;
    z = sqlite3_malloc64(n+1);
    if( z==0 ){
        sqlite3_result_error_nomem(context);
        return;
    }
    j = 0;
    for(i=0; i<argc; i++){
        k = sqlite3_value_bytes(argv[i]);
        if( k>0 ){
            const char *v = (const char*)sqlite3_value_text(argv[i]);
            if( v!=0 ){
                if( j>0 && nSep>0 ){
                    memcpy(&z[j], zSep, nSep);
                    j += nSep;
                }
                memcpy(&z[j], v, k);
                j += k;
            }
        }
    }
}

```

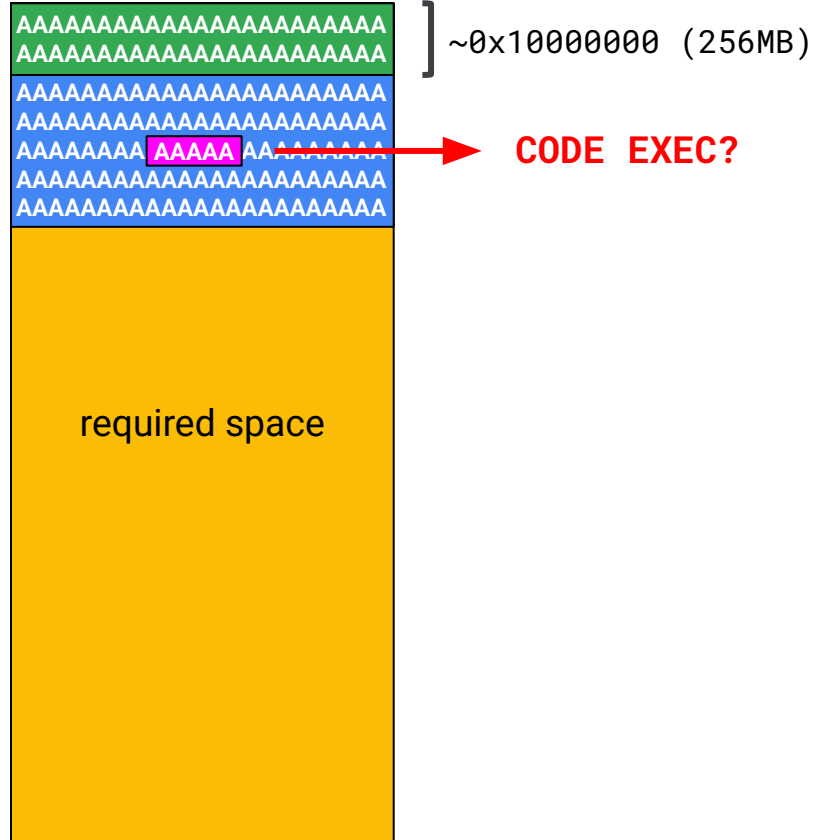


~0x10000000 (256MB)

required space

Too small allocation

```
static void concatFuncCore(  
    sqlite3_context *context,  
    int argc,  
    sqlite3_value **argv,  
    int nSep,  
    const char *zSep  
) {  
    i64 j, k, n = 0;  
    int i;  
    char *z;  
    for(i=0; i<argc; i++){  
        n += sqlite3_value_bytes(argv[i]);  
    }  
    n += (argc-1)*nSep;  
    z = sqlite3_malloc64(n+1);  
    if( z==0 ){  
        sqlite3_result_error_nomem(context);  
        return;  
    }  
    j = 0;  
    for(i=0; i<argc; i++){  
        k = sqlite3_value_bytes(argv[i]);  
        if( k>0 ){  
            const char *v = (const char*)sqlite3_value_text(argv[i]);  
            if( v!=0 ){  
                if( j>0 && nSep>0 ){  
                    memcpy(&z[j], zSep, nSep);  
                    j += nSep;  
                }  
                memcpy(&z[j], v, k);  
                j += k;  
            }  
        }  
    }  
}
```

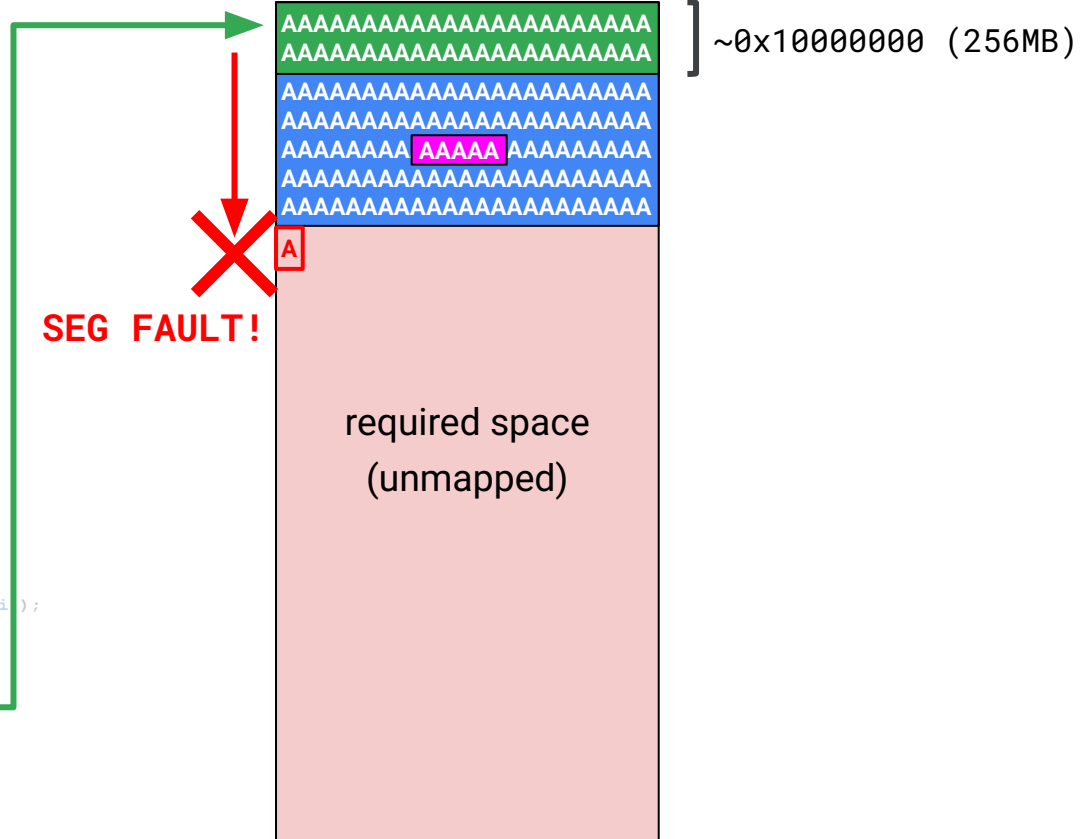


Too small allocation

```

static void concatFuncCore(
    sqlite3_context *context,
    int argc,
    sqlite3_value **argv,
    int nSep,
    const char *zSep
){
    i64 j, k, n = 0;
    int i;
    char *z;
    for(i=0; i<argc; i++){
        n += sqlite3_value_bytes(argv[i]);
    }
    n += (argc-1)*nSep;
    z = sqlite3_malloc64(n+1);
    if( z==0 ){
        sqlite3_result_error_nomem(context);
        return;
    }
    j = 0;
    for(i=0; i<argc; i++){
        k = sqlite3_value_bytes(argv[i]);
        if( k>0 ){
            const char *v = (const char*)sqlite3_value_text(argv[i]);
            if( v!=0 ){
                if( j>0 && nSep>0 ){
                    memcpy(&z[j], zSep, nSep);
                    j += nSep;
                }
                memcpy(&z[j], v, k);
                j += k;
            }
        }
    }
}

```



Changelog

2025-02-18 (3.49.1)

1. Improve portability of makefiles and configure scripts.
2. Fix a bug in the `concat_ws()` function, introduced in [version 3.44.0](#), that **could lead to a memory error, specifically a write past the end of allocated space**, if the separator string is larger than two megabytes.
3. Enhanced the `SQLITE_DBCONFIG_LOOKASIDE` interface to make it more robust against misuse.

Hashes:

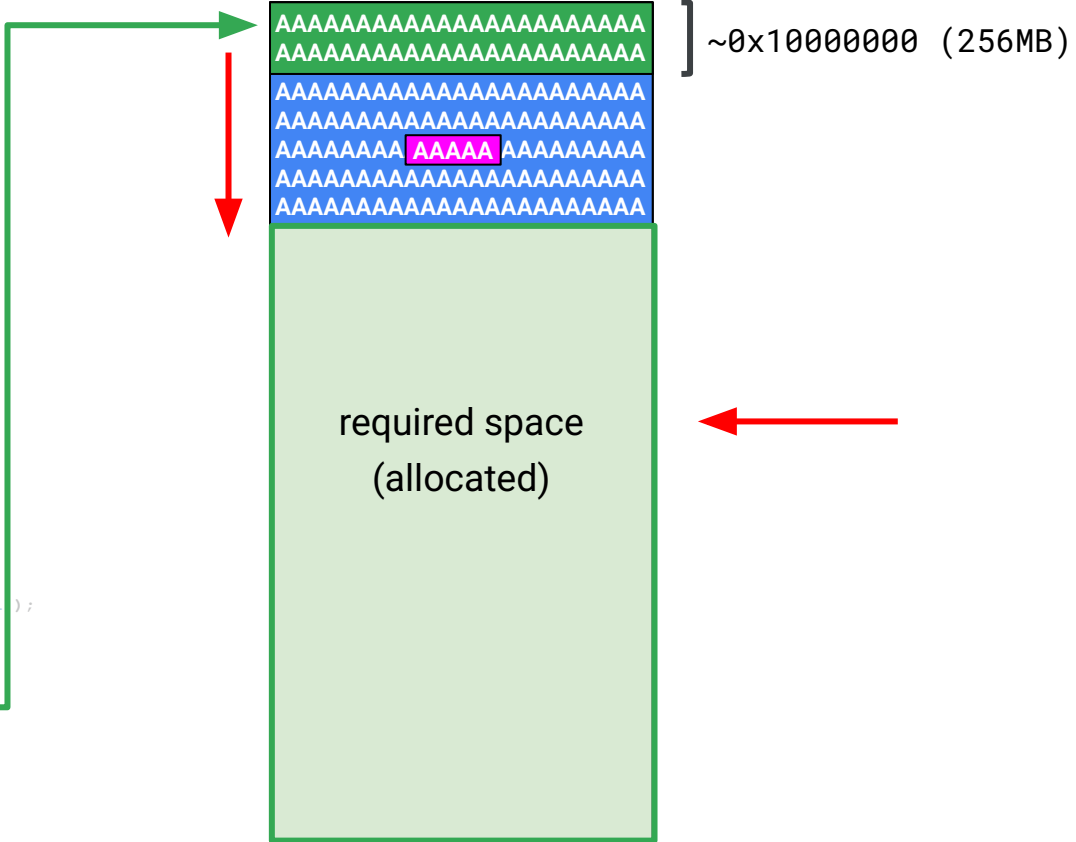
4. SQ
 5. SH
- ... could lead to a memory error, specifically a write past the end of allocated space, ...**

2025-

1. Enh
 - a. Better query plans for large [star-query](#) joins. This fixes three different performance regressions that were reported on the SQLite Forum.
 - b. When two or more queries have the same estimated cost, use the one with the fewer bytes per row.
 - c. Enhance the `if()` [SQL function](#) so that it can accept any number of arguments greater than or equal to two.
2. Enhance the [session extension](#) so that it works on databases that make use of [generated columns](#).
3. Omit the `SQLITE_USE_STDIO_FOR_CONSOLE` compile-time option which was not implemented correctly and never worked right. In its place add the `SQLITE_USE_W32_FOR_CONSOLE_IO` compile-time option. This option applies to command-line tools like the [CLI](#) only, not to the SQLite core. It causes Win32 APIs to be used for console I/O instead of `stdio`. This option affects Windows builds only.
4. Three new options to `sqlite3_db_config()`. All default to "on".
 - a. [SQLITE_DBCONFIG_ENABLE_ATTACH_CREATE](#)
 - b. [SQLITE_DBCONFIG_ENABLE_ATTACH_WRITE](#)
 - c. [SQLITE_DBCONFIG_ENABLE_COMMENTS](#)
5. Replace [Autotools](#) with [Autosetup](#) for the configure script used in the [precompiled amalgamation tarball](#). The configure script for the [canonical source code](#) was changed to `Autosetup` in the previous (3.48.0) release. Only the main SQLite configure script in the amalgamation tarball is changed. The (deprecated) configuration script use by [TEA](#) subdirectory of the amalgamation tarball still relies on `Autotools`.
6. Various minor patches and fixes for problems seen in the 3.48.0 release.

Exploitable?

```
static void concatFuncCore(  
    sqlite3_context *context,  
    int argc,  
    sqlite3_value **argv,  
    int nSep,  
    const char *zSep  
)  
{  
    i64 j, k, n = 0;  
    int i;  
    char *z;  
    for(i=0; i<argc; i++){  
        n += sqlite3_value_bytes(argv[i]);  
    }  
    n += (argc-1)*nSep;  
    z = sqlite3_malloc64(n+1);  
    if( z==0 ){  
        sqlite3_result_error_nomem(context);  
        return;  
    }  
    j = 0;  
    for(i=0; i<argc; i++){  
        k = sqlite3_value_bytes(argv[i]);  
        if( k>0 ){  
            const char *v = (const char*)sqlite3_value_text(argv[i]);  
            if( v!=0 ){  
                if( j>0 && nSep>0 ){  
                    memcpy(&z[j], zSep, nSep);  
                    j += nSep;  
                }  
                memcpy(&z[j], v, k);  
                j += k;  
            }  
        }  
    }  
}
```

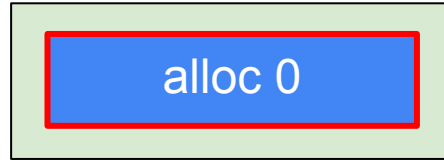


Wild copy challenge

SELECT ... huge string ...

Wild copy challenge

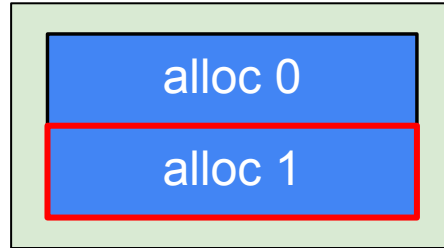
SELECT ... huge string ...



Wild copy challenge

SELECT ... huge string ...

UNION SELECT ... huge string ...

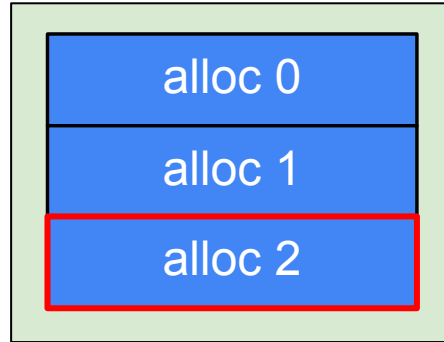


Wild copy challenge

SELECT ... huge string ...

UNION SELECT ... huge string ...

UNION SELECT ... huge string ...



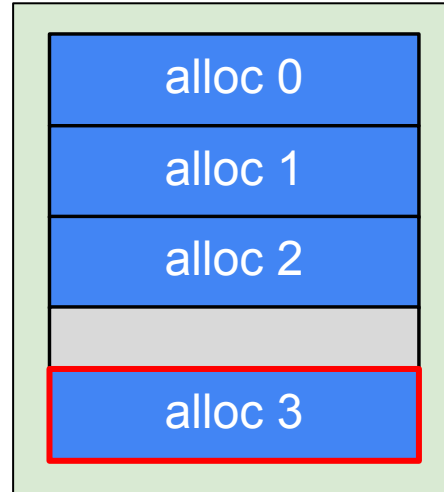
Wild copy challenge

SELECT ... huge string ...

UNION SELECT ... huge string ...

UNION SELECT ... huge string ...

UNION SELECT ... huge string ... →



Wild copy challenge

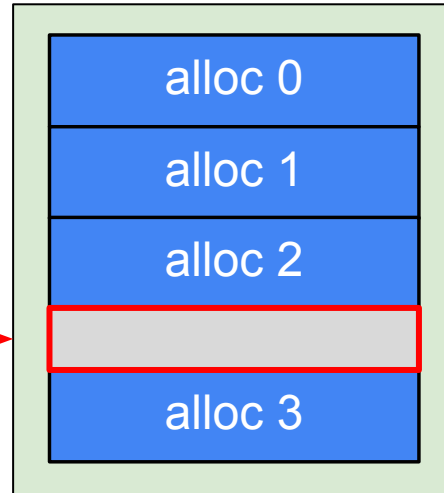
SELECT ... huge string ...

UNION SELECT ... huge string ...

UNION SELECT ... huge string ...

UNION SELECT ... huge string ...

read-only →



Wild copy challenge

SELECT ... huge string ...

UNION SELECT ... huge string ...

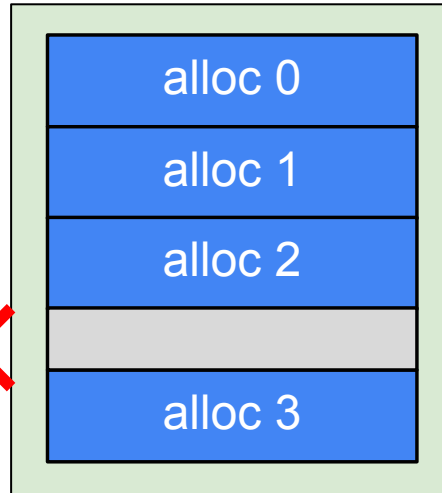
UNION SELECT ... huge string ...

UNION SELECT ... huge string ...

overflow



SEG FAULT!



Heap groom

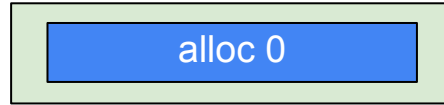
adjust size

SELECT

... big string ...
.. ~~huge string~~ ...

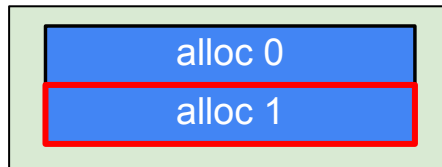
Heap groom

SELECT ... big string ...



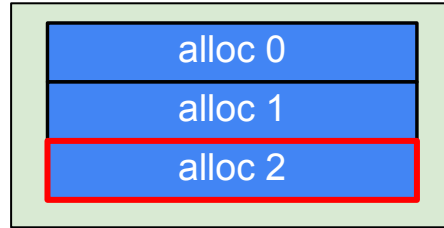
Heap groom

```
SELECT ... big string ...  
UNION SELECT ... big string ...
```



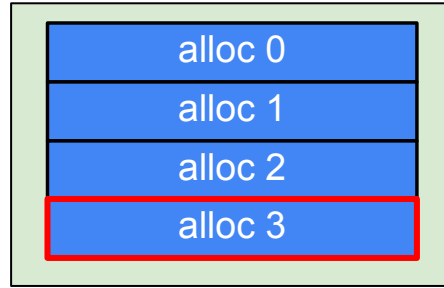
Heap groom

```
SELECT ... big string ...  
UNION SELECT ... big string ...  
UNION SELECT ... big string ...
```



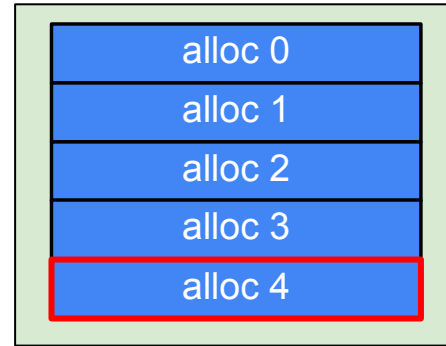
Heap groom

```
SELECT ... big string ...  
UNION SELECT ... big string ...  
UNION SELECT ... big string ...  
UNION SELECT ... big string ...
```



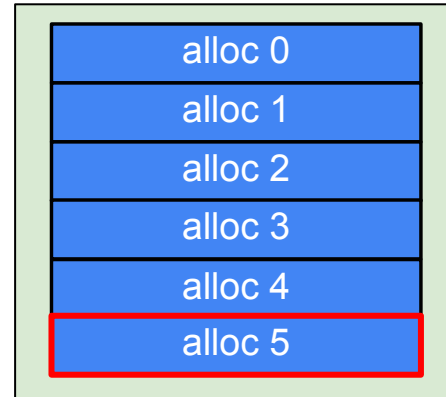
Heap groom

```
SELECT ... big string ...  
UNION SELECT ... big string ...  
UNION SELECT ... big string ...  
UNION SELECT ... big string ...  
UNION SELECT ... big string ...
```



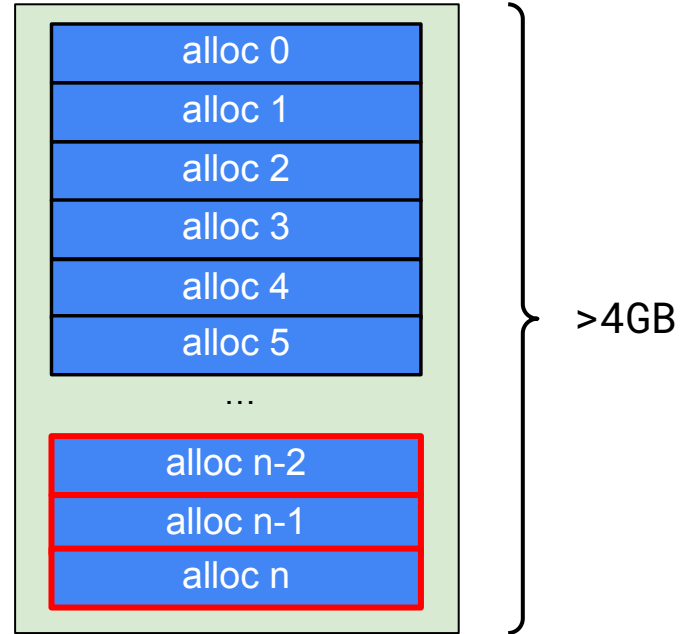
Heap groom

```
SELECT ... big string ...  
UNION SELECT ... big string ...  
UNION SELECT ... big string ...  
UNION SELECT ... big string ...  
UNION SELECT ... big string ...  
UNION SELECT ... big string ...
```

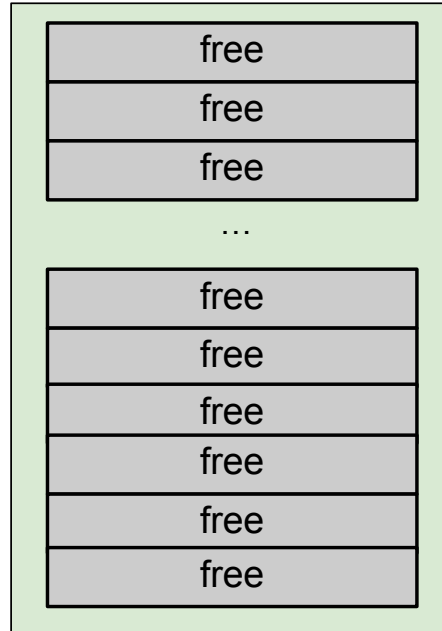


Heap groom

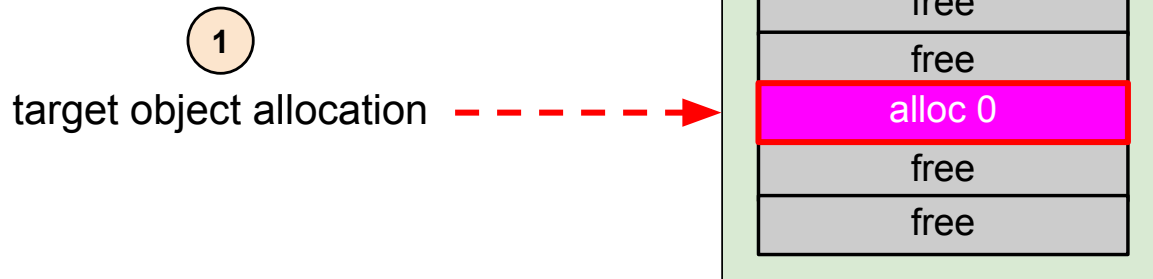
```
SELECT ... big string ...  
UNION SELECT ... big string ...  
UNION SELECT ... big string ...  
UNION SELECT ... big string ...  
UNION SELECT ... big string ...  
UNION SELECT ... big string ...  
...  
UNION SELECT ... big string ...  
UNION SELECT ... big string ...  
UNION SELECT ... big string ...
```



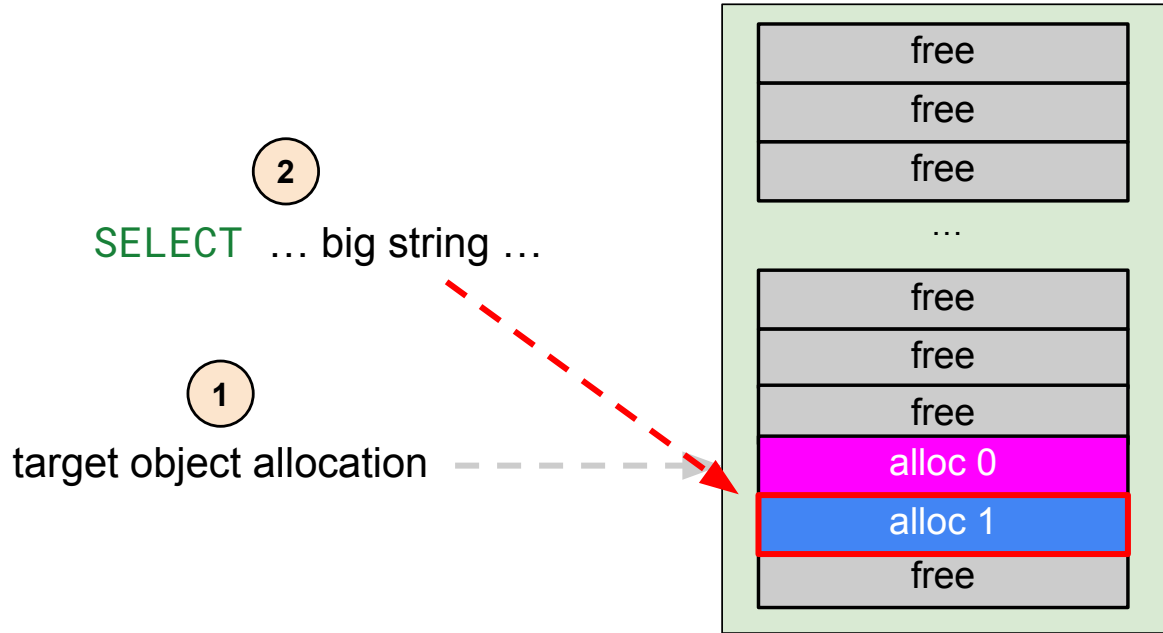
Heap groom



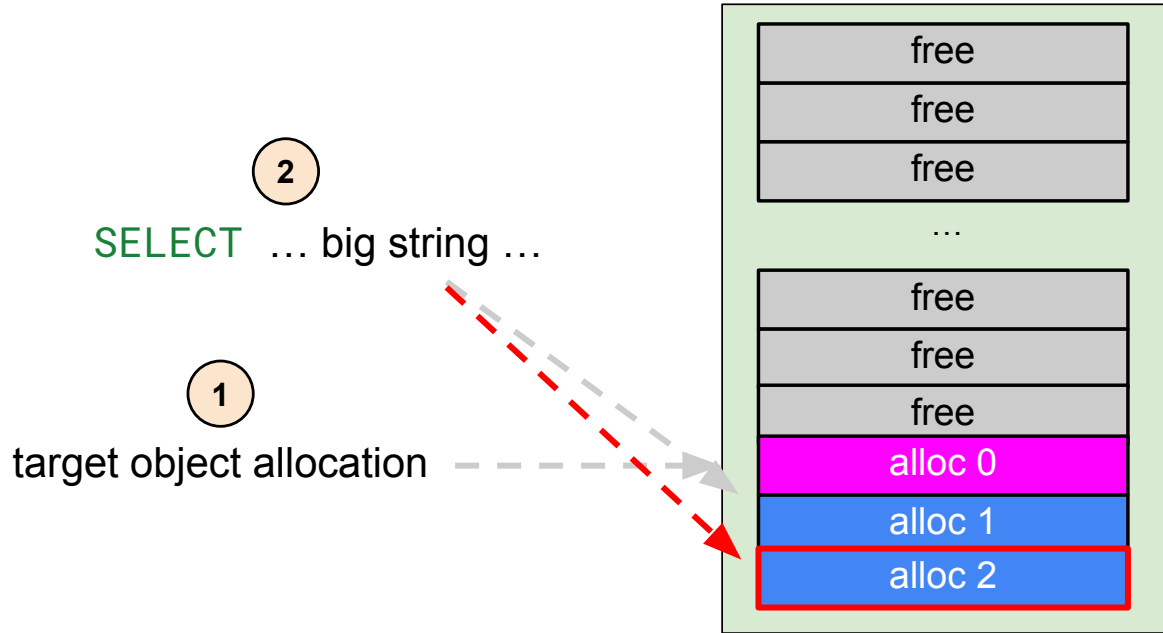
Overwriting Target



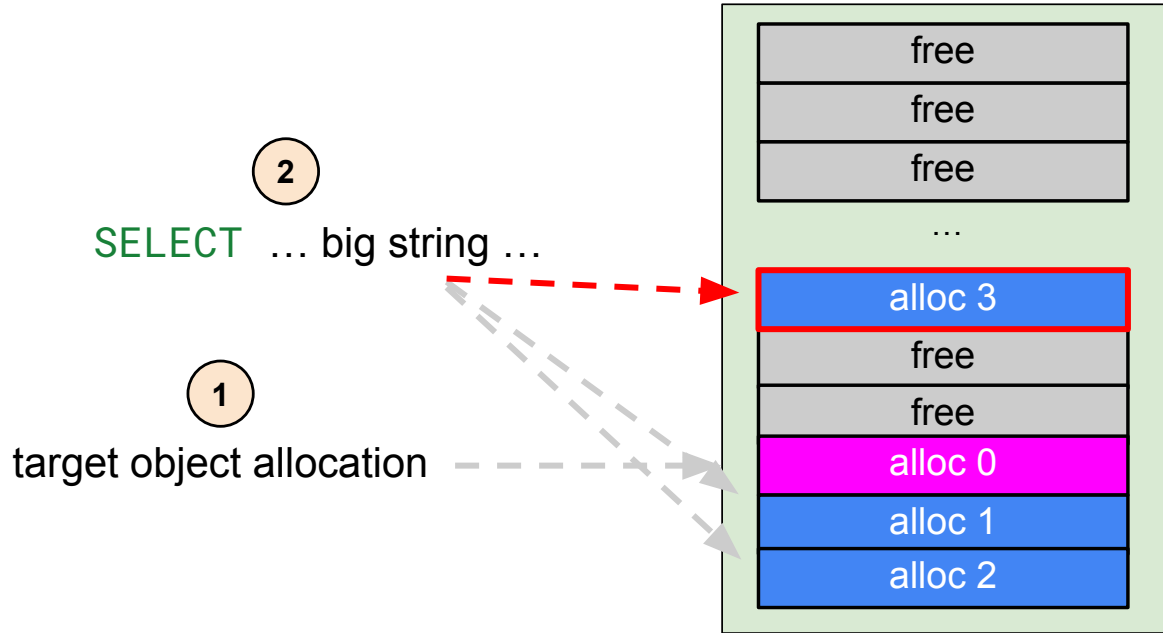
Overwriting Target



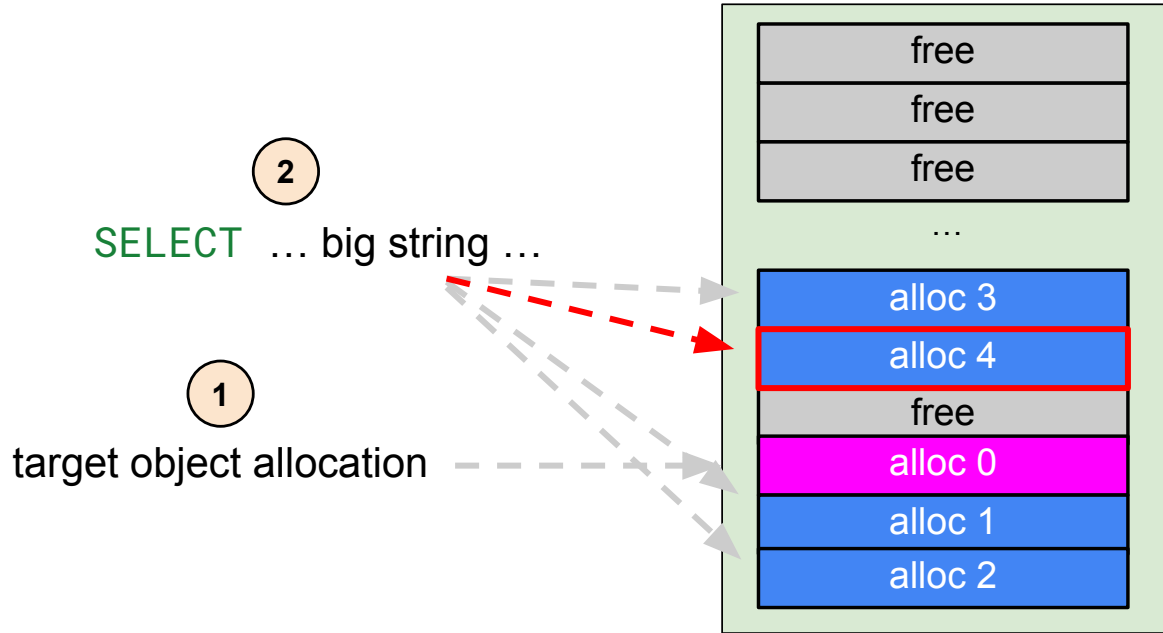
Overwriting Target



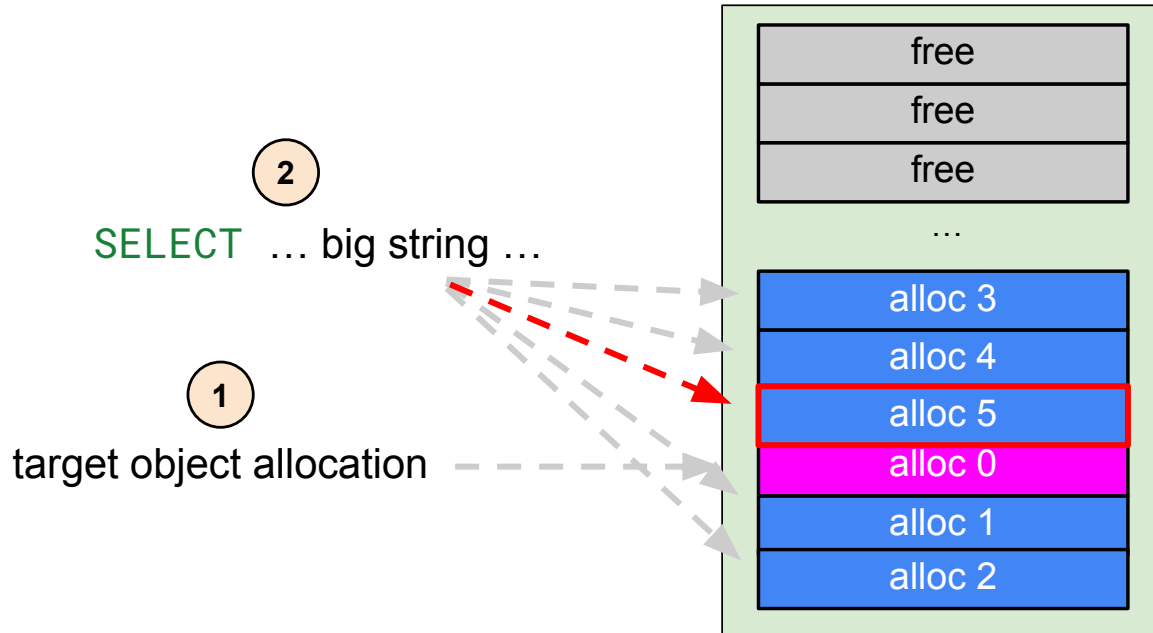
Overwriting Target



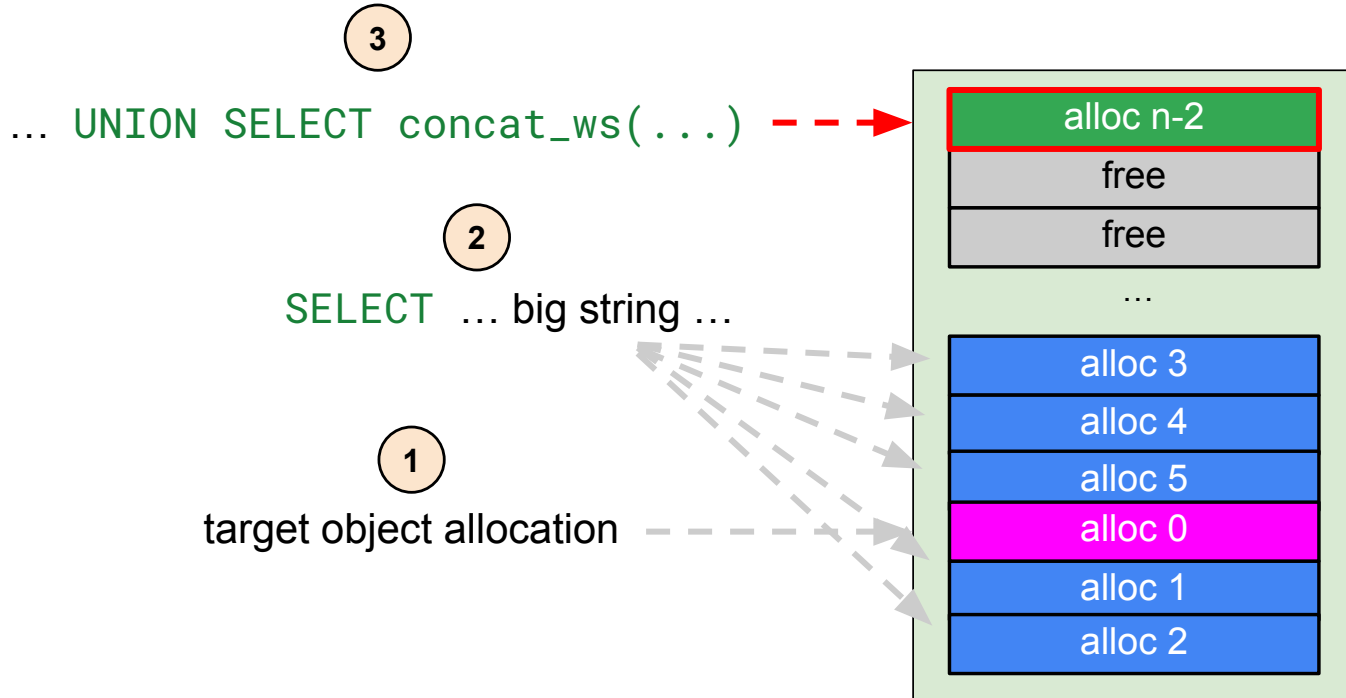
Overwriting Target



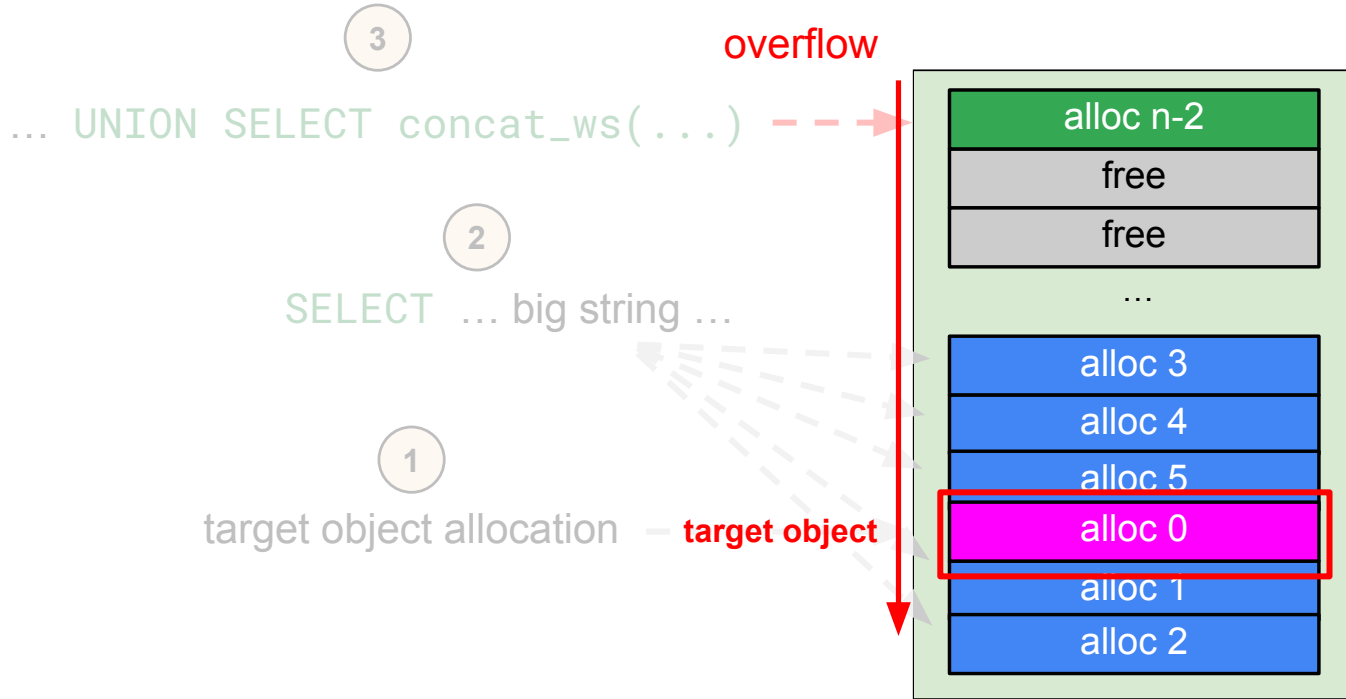
Overwriting Target



Overwriting Target



Overwriting Target



Target Object



1. Can be allocated from a SELECT query

Target Object



1. Can be allocated from a SELECT query
2. Size is controllable

Target Object



1. Can be allocated from a SELECT query
2. Size is controllable
3. Contains data that allows gaining code execution

Target Object



```
/* Allocate and populate the Fts3Table structure. */
nByte = sizeof(Fts3Table) + /* Fts3Table */
        nCol * sizeof(char *) + /* azColumn */
        nIndex * sizeof(struct Fts3Index) + /* aIndex */
        nCol * sizeof(u8) + /* abNotindexed */
        nName + /* zName */
        nDb + /* zDb */
        nString; /* Space for azColumn strings */

p = (Fts3Table*)sqlite3_malloc64(nByte);
```

Target Object



```
/* Allocate and populate the Fts3Table structure. */
nByte = sizeof(Fts3Table) + /* Fts3Table */
        nCol * sizeof(char *) + /* azColumn */
        nIndex * sizeof(struct Fts3Index) + /* aIndex */
        nCol * sizeof(u8) + /* abNotindexed */
        nName + /* zName */
        nDb + /* zDb */
        nString; /* Space for azColumn strings */

p = (Fts3Table*)sqlite3_malloc64(nByte);
```

Target Object



```
struct Fts3Table {  
    sqlite3_vtab base;           /* Base class used by SQLite core */  
    sqlite3 *db;                /* The database connection */  
    const char *zDb;            /* logical database name */  
    const char *zName;          /* virtual table name */  
    // ...  
  
    struct sqlite3_vtab {  
        const sqlite3_module *pModule; /* The module for this virtual table */  
        int nRef;                       /* Number of open cursors */  
        char *zErrMsg;                  /* Error message from sqlite3_mprintf() */  
        /* Virtual table implementations will typically add additional fields */  
    };  
};
```

Target Object



```
struct Fts3Table {  
    sqlite3_vtab base; /* Base class used by SQLite core */  
    sqlite3 *db; /* The database connection */  
    const char *zDb; /* logical database name */  
    const char *zName; /* virtual table name */  
    // ...  
  
    struct sqlite3_vtab {  
        const sqlite3_module *pModule; /* The module for this virtual table */  
        int nRef; /* Number of open cursors */  
        char *zErrMsg; /* Error message from sqlite3_mprintf() */  
        /* Virtual table implementations will typically add additional fields */  
    };  
};
```

Target Object



```
struct sqlite3_module {
    int iVersion;
    int (*xCreate)(sqlite3*, void *pAux,
                   int argc, const char *const*argv,
                   sqlite3_vtab **ppVTab, char**);
    int (*xConnect)(sqlite3*, void *pAux,
                    int argc, const char *const*argv,
                    sqlite3_vtab **ppVTab, char**);
    int (*xBestIndex)(sqlite3_vtab *pVTab, sqlite3_index_info*);
    int (*xDisconnect)(sqlite3_vtab *pVTab);
    int (*xDestroy)(sqlite3_vtab *pVTab);
    int (*xOpen)(sqlite3_vtab *pVTab, sqlite3_vtab_cursor **ppCursor);
    int (*xClose)(sqlite3_vtab_cursor*);
    // ...
};
```

Target Object



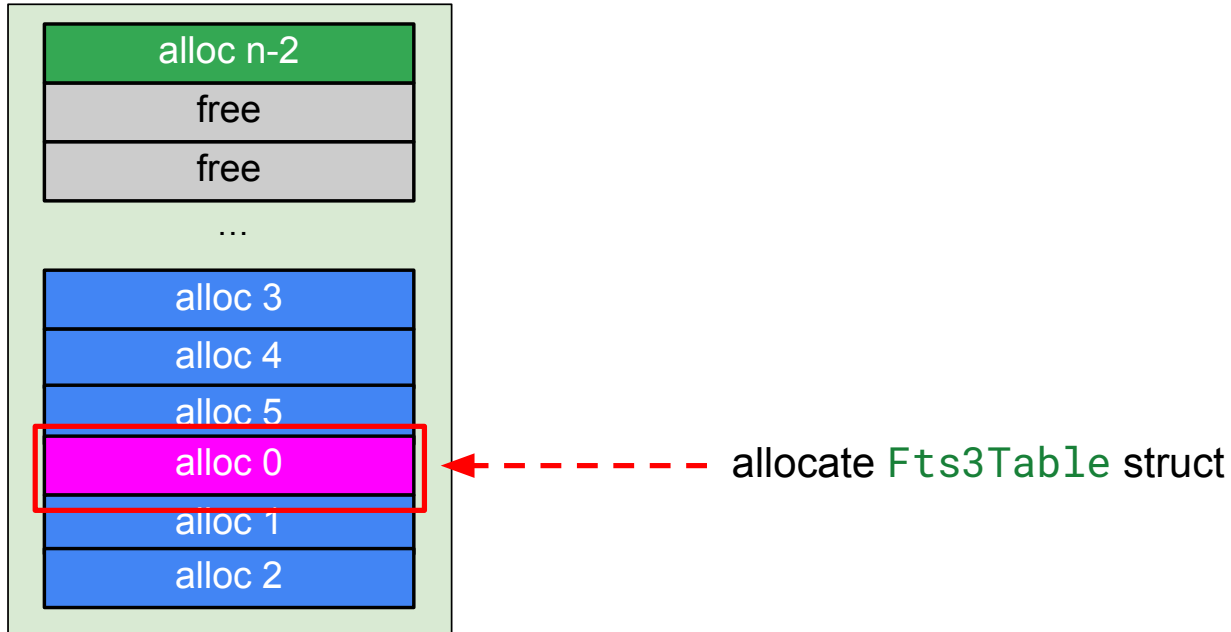
```
struct sqlite3_module {
    int iVersion;
    int (*xCreate)(sqlite3*, void *pAux,
                  int argc, const char *const*argv,
                  sqlite3_vtab **ppVTab, char**);
    int (*xConnect)(sqlite3*, void *pAux,
                   int argc, const char *const*argv,
                   sqlite3_vtab **ppVTab, char**);
    int (*xBestIndex)(sqlite3_vtab *pVTab, sqlite3_index_info*);
    int (*xDisconnect)(sqlite3_vtab *pVTab);
    int (*xDestroy)(sqlite3_vtab *pVTab);
    int (*xOpen)(sqlite3_vtab *pVTab, sqlite3_vtab_cursor **ppCursor);
    int (*xClose)(sqlite3_vtab_cursor*);
    // ...
};
```

Target Object

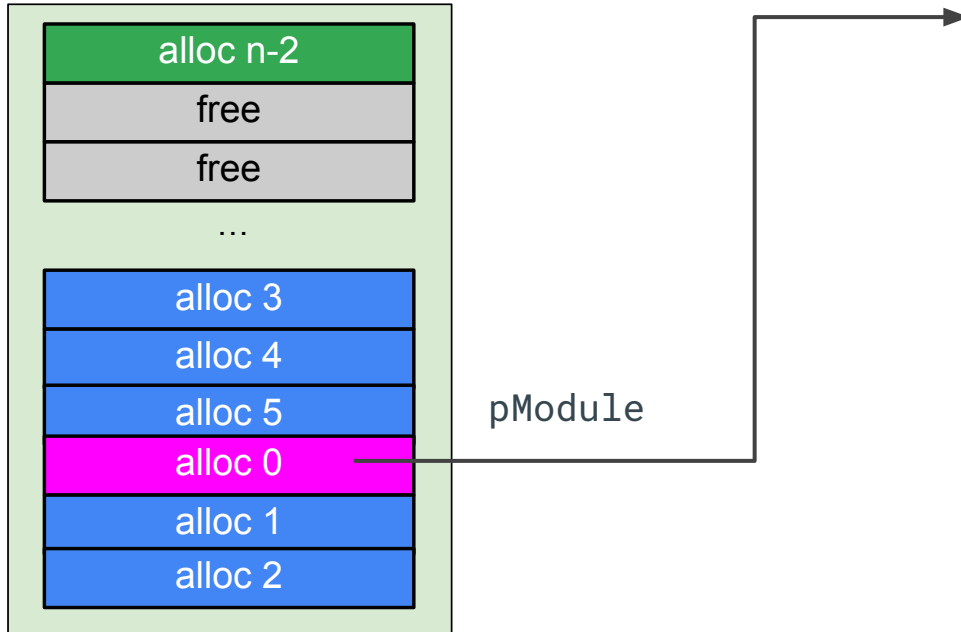


```
SQLITE_PRIVATE void sqlite3VdbeFreeCursorNN(Vdbe *p, VdbeCursor *pCx){  
    // ...  
    case CURTYPE_VTAB: {  
        sqlite3_vtab_cursor *pVCur = pCx->uc.pVCur;  
        const sqlite3_module *pModule = pVCur->pVtab->pModule;  
        assert( pVCur->pVtab->nRef>0 );  
        pVCur->pVtab->nRef--;  
        pModule->xClose(pVCur);  
        // ...  
    }
```

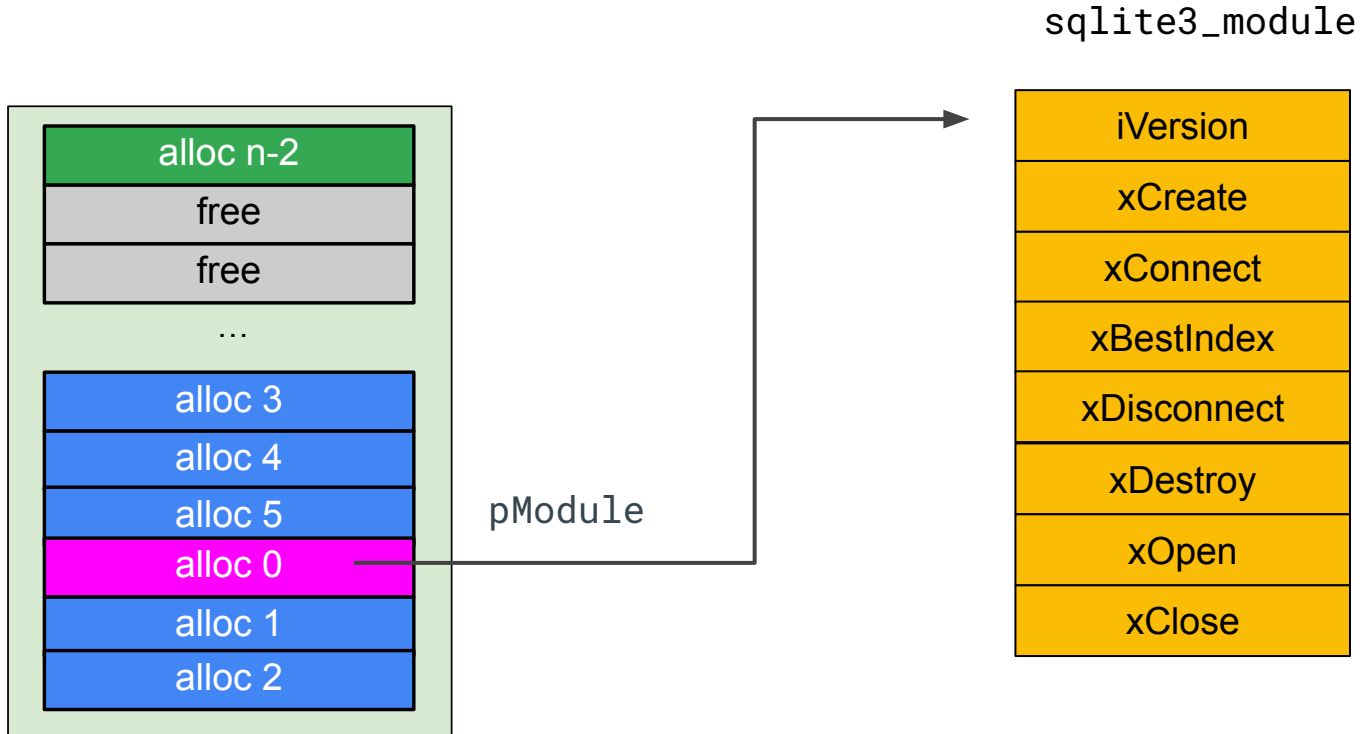
Target Object



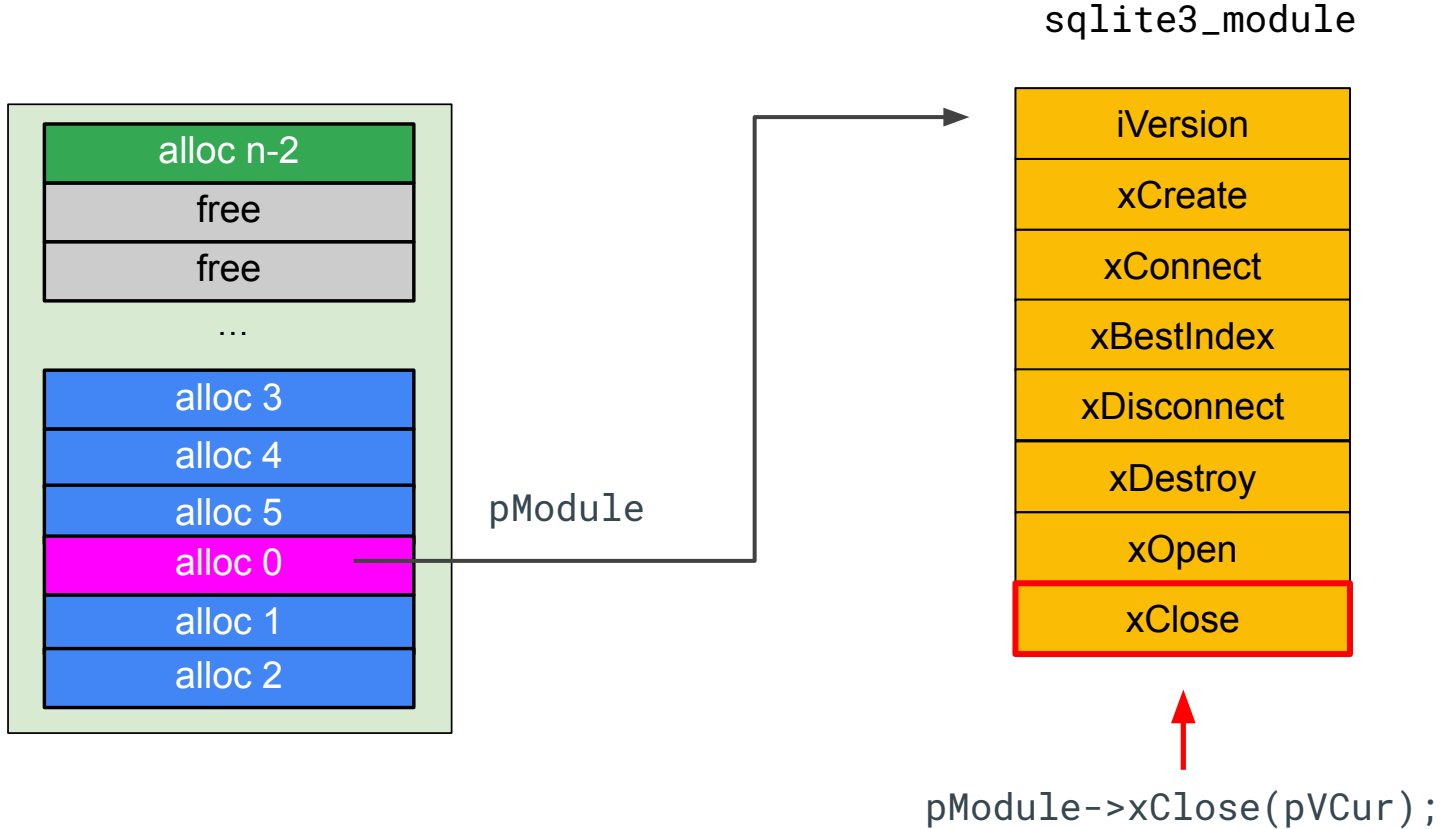
Target Object



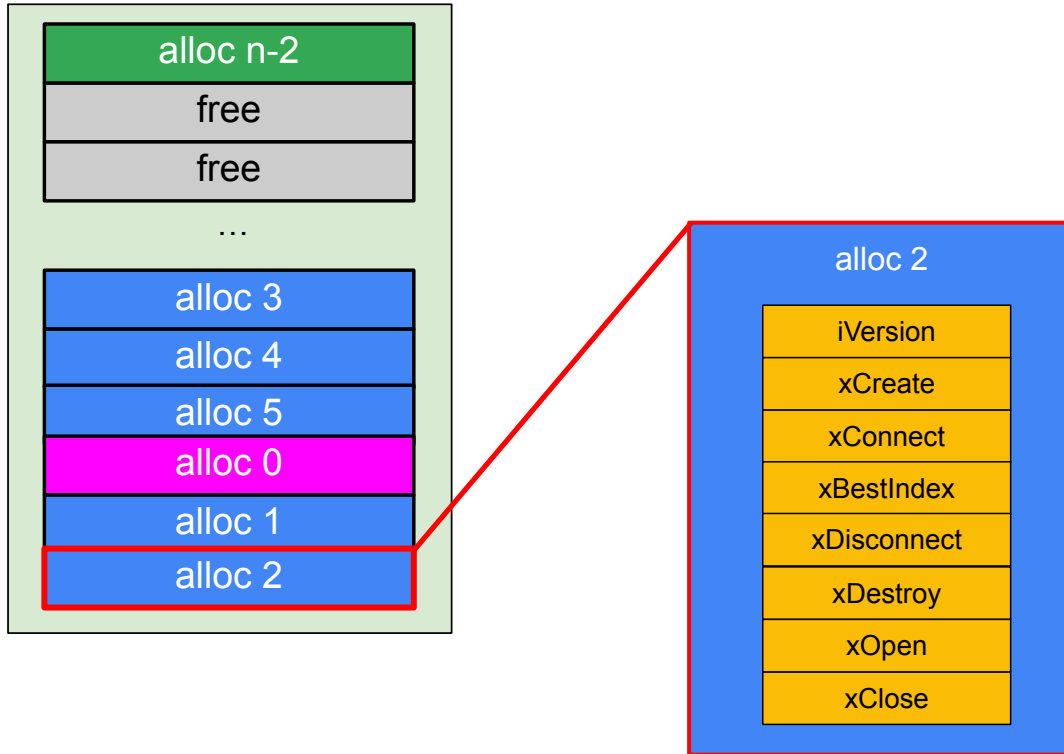
Target Object



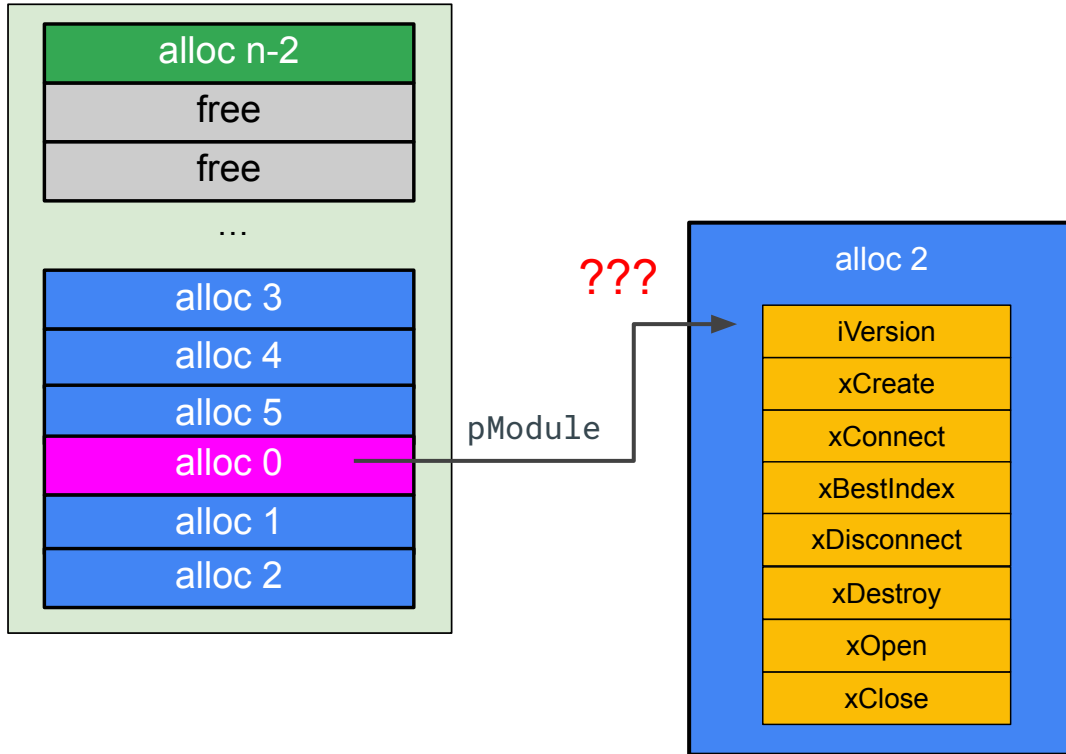
Target Object



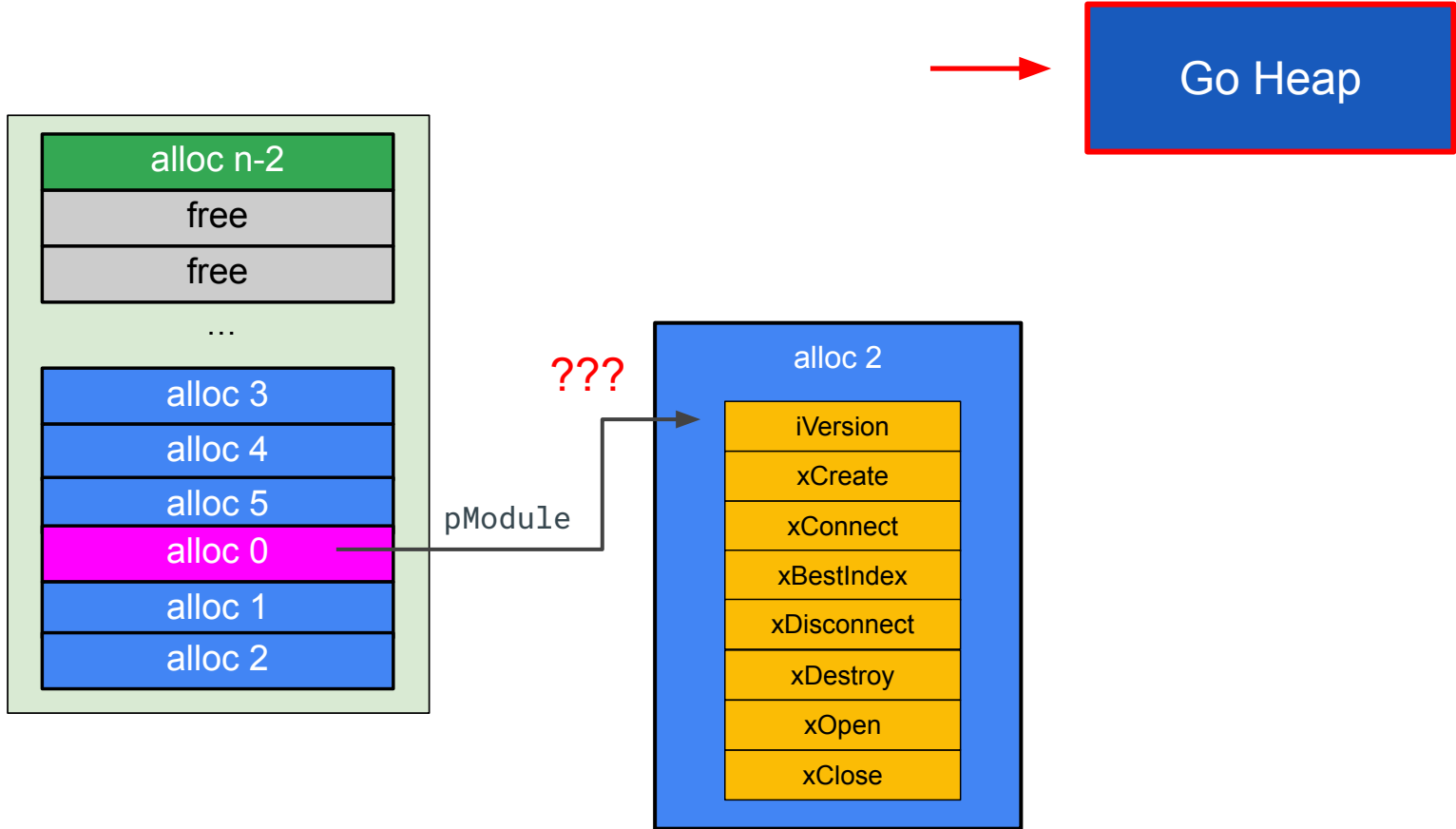
Storing Fake Struct



Storing Fake Struct



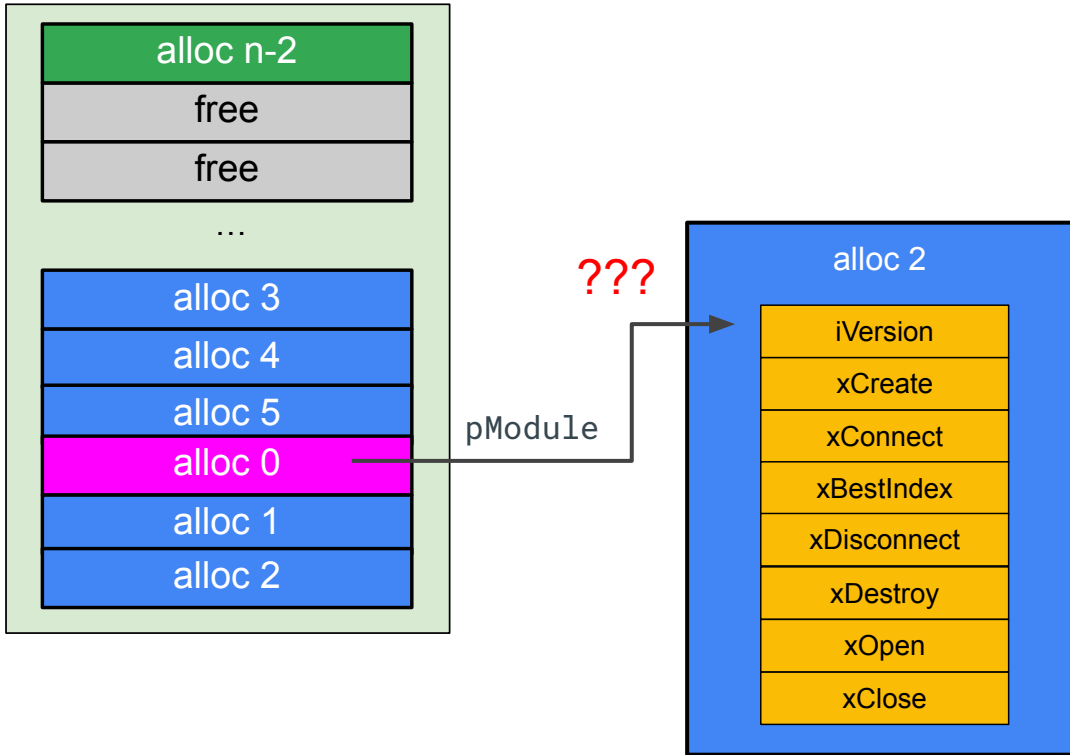
Storing Fake Struct



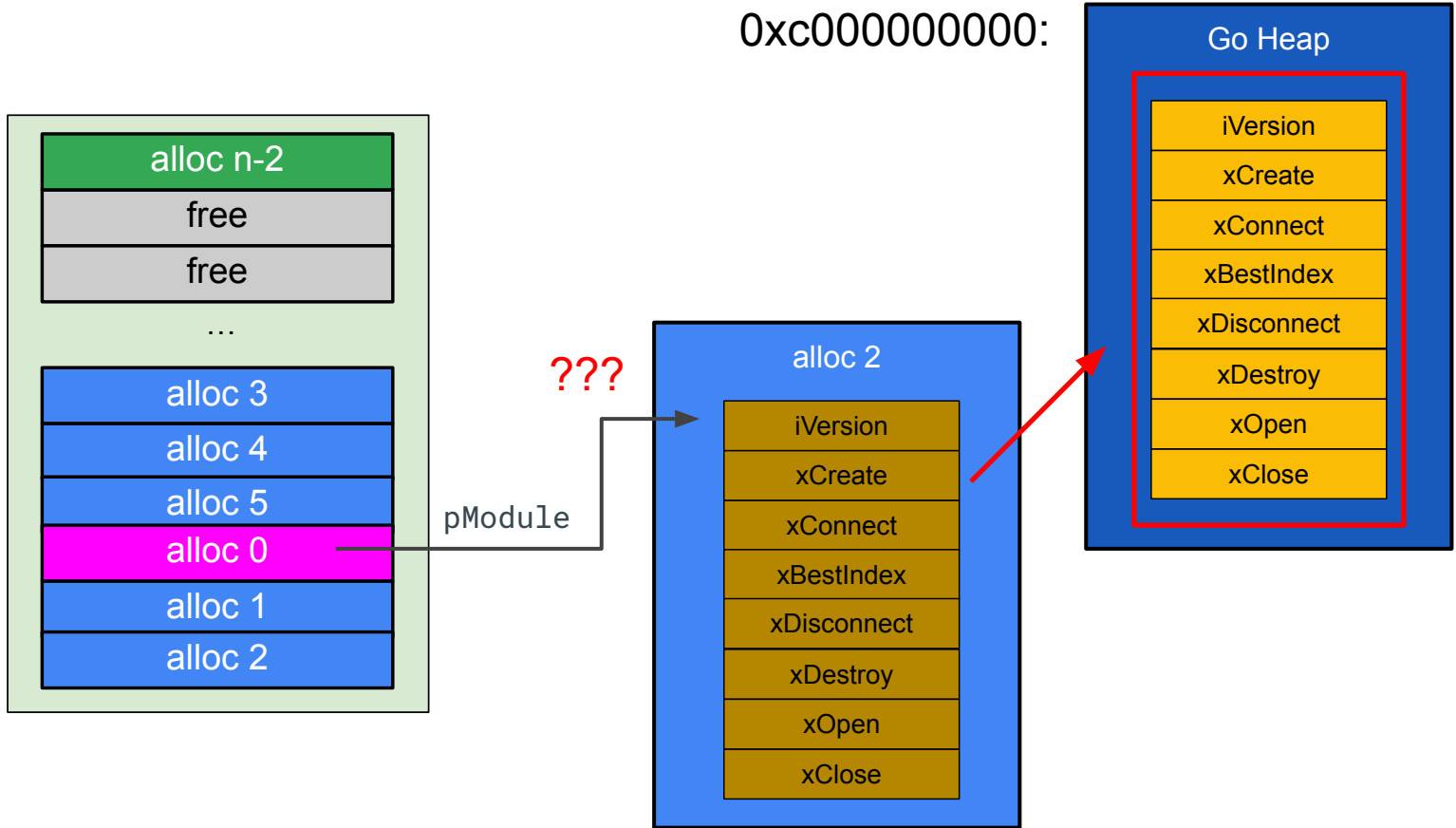
Storing Fake Struct

0xc000000000:

Go Heap



Storing Fake Struct

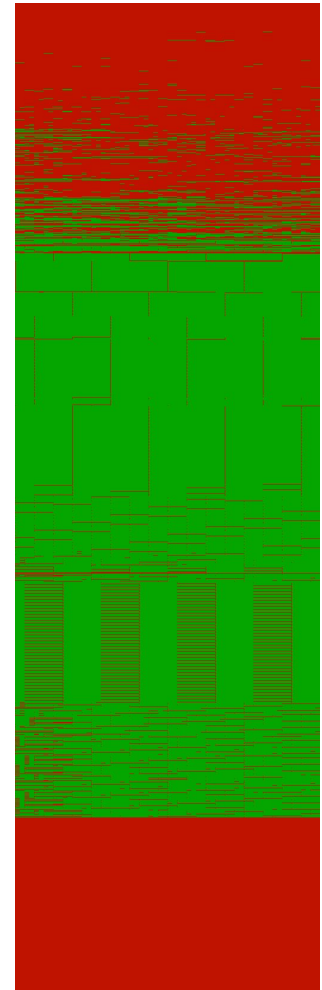


Go Heap Spray



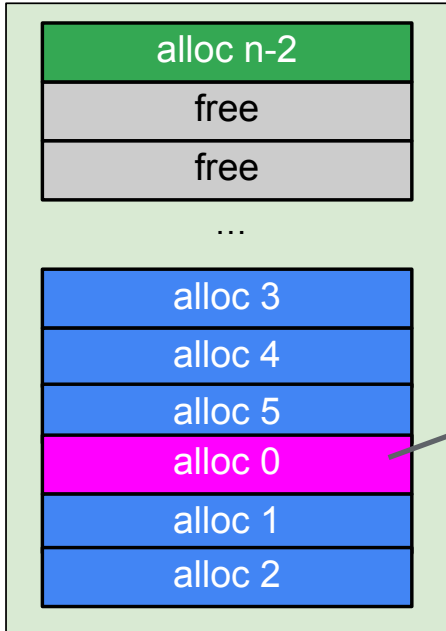
0xc000000000:

spray!

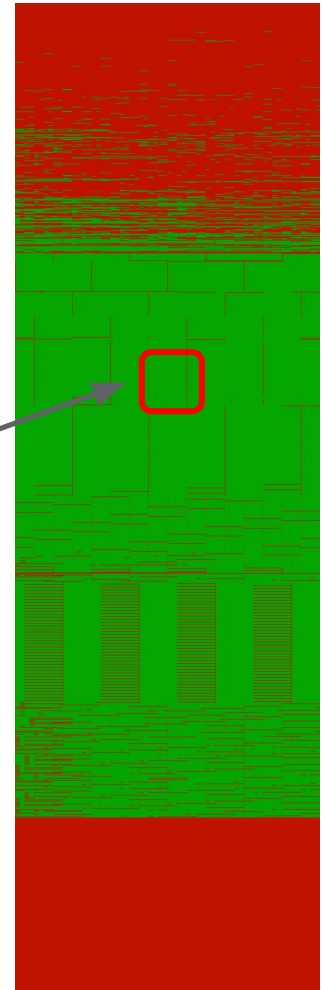


Go Heap Spray

0xc000000000:



0xc002388010





Demo





Conclusion



Summary

- We leveraged a Path Traversal issue into a Conditional Corruption primitive
 - We used a binary signal (error or success) on the File-Write to only trigger a Buffer Overflow vulnerability if we are on a worker with an expected ASLR offset
- We used a Security Vulnerability in SQLite without a previously assigned CVE to gain Remote Code Execution
 - Credit: Yuelin Wang

Conclusion

We demonstrated an end-to-end Memory Corruption chain in GCP that could have been exploited without access to the source code and binaries of the service

We also referenced other case studies where we used variants

We believe that Memory Corruption in Cloud should be considered a practical risk that can be exploited with specialized bugs and a repository of specialized exploitation techniques



Questions?

