



## **Fortify Security Report**

2024-6-21

ASUS

Executive Summary

Issues Overview

On 2024-6-21, a source code review was performed over the permafrost-engine code base. 2,400 files, 419,819 LOC (Executable) were scanned and reviewed for defects that could lead to potential security vulnerabilities. A total of 96 reviewed findings were uncovered during the analysis.

Issues by Fortify Priority Order

|          |    |
|----------|----|
| Critical | 54 |
| High     | 42 |

Recommendations and Conclusions

The Issues Category section provides Fortify recommendations for addressing issues at a generic level. The recommendations for specific fixes can be extrapolated from those generic recommendations by the development group.

## Project Summary

### Code Base Summary

Code location: C:/Users/ASUS/Desktop/Gitrepo/permafrost-engine

Number of Files: 2400

Lines of Code: 419819

Build Label: <No Build Label>

### Scan Information

Scan time: 01:48:03

SCA Engine version: 20.1.1.0007

Machine Name: DESKTOP-MK5UPFE

Username running scan: ASUS

### Results Certification

Results Certification Valid

Details:

Results Signature:

SCA Analysis Results has Valid signature

Rules Signature:

There were no custom rules used in this scan

### Attack Surface

Attack Surface:

Command Line Arguments:

null.null.null

Environment Variables:

null.null.null

os.null.getenv

File System:

null.null.open

null.file.\_\_init\_\_

null.file.read

null.file.readline

null.file.readlines

null.file.xreadlines

deps.Python.Lib.distutils.text\_file.TextFile.open

deps.Python.Lib.idlelib.FileList.FileList.open

deps.Python.Lib.idlelib.IOBinding.IOBinding.open

deps.Python.Lib.idlelib.ReplaceDialog.ReplaceDialog.open

deps.Python.Lib.idlelib.SearchDialogBase.SearchDialogBase.open

deps.Python.Lib.test.audiotests.UnseekableIO.\_\_init\_\_

```
deps.Python.Lib.test.test_descr.ClassPropertiesAndMethods.CountedInput.__init__
deps.Python.Lib.test.test_descr.ClassPropertiesAndMethods.CountedInput.readline
deps.Python.Lib.test.test_file2k.FileSubclassTests.C.__init__
deps.Python.Lib.urllib.URLopener.open
deps.Python.Lib.webbrowser.BackgroundBrowser.open
deps.Python.Lib.webbrowser.BaseBrowser.open
os.null.open
urllib.URLopener.open
urllib2.OpenerDirector.open
zipfile.ZipFile.open
```

#### Private Information:

```
null.null.null
null.dict.__getitem__
hashlib.null.pbkdf2_hmac
```

#### Serialized Data:

```
cPickle.null.Unpickler
cPickle.null.load
cPickle.null.loads
```

#### Standard Input Stream:

```
null.null.null
null.null.input
null.null.raw_input
deps.Python.Lib.idlelib.PyShell.PseudoInputFile.read
deps.Python.Lib.idlelib.PyShell.PseudoInputFile.readline
```

#### Stream:

```
os.null.read
```

#### System Information:

```
null.null.null
null.null.null
null.null.dir
null.null.globals
java.lang.System.getProperty
java.lang.Throwable.getMessage
os.null.getcwdu
os.null.geteuid
os.null.getgid
os.null.getgroups
os.null.getlogin
os.null.getpgrp
os.null.getpid
os.null.getsid
os.null.getuid
os.null.strerror
os.null.sysconf
site.null.getsitepackages
site.null.getuserbase
```

site.null.getusersitepackages  
ssl.SSLSocket.selected\_alpn\_protocol  
ssl.SSLSocket.selected\_npn\_protocol  
sys.null.\_current\_frames  
sys.null.\_getframe  
sys.null.exc\_info  
sys.null.getwindowsversion  
sysconfig.null.get\_config\_h\_filename  
sysconfig.null.get\_path  
sysconfig.null.get\_paths  
sysconfig.null.get\_platform  
sysconfig.null.get\_python\_version  
sysconfig.null.parse\_config\_h

### Filter Set Summary

Current Enabled Filter Set:

[Quick View](#)

Filter Set Details:

Folder Filters:

If [fortify priority order] contains critical Then set folder to Critical  
If [fortify priority order] contains high Then set folder to High  
If [fortify priority order] contains medium Then set folder to Medium  
If [fortify priority order] contains low Then set folder to Low

Visibility Filters:

If impact is not in range [2.5, 5.0] Then hide issue  
If likelihood is not in range (1.0, 5.0] Then hide issue

### Audit Guide Summary

J2EE Bad Practices

Hide warnings about J2EE bad practices.

Depending on whether your application is a J2EE application, J2EE bad practice warnings may or may not apply. AuditGuide can hide J2EE bad practice warnings.

Enable if J2EE bad practice warnings do not apply to your application because it is not a J2EE application.

Filters:

If category contains j2ee Then hide issue  
If category is race condition: static database connection Then hide issue

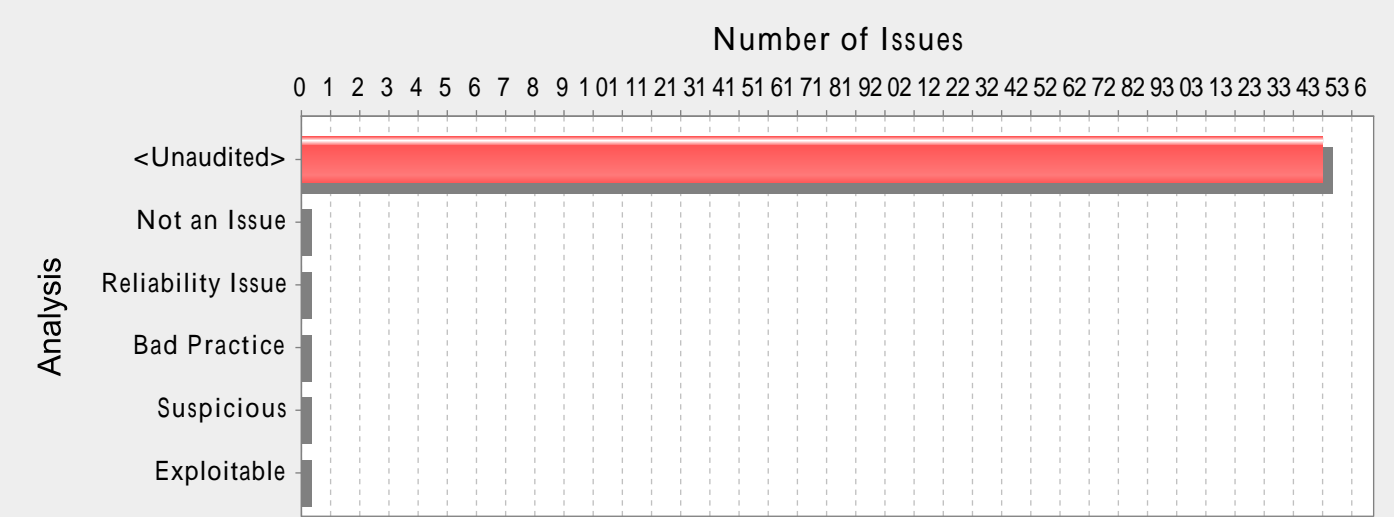
Results Outline

Overall number of results

The scan found 96 issues.

Vulnerability Examples by Category

Category: Key Management: Hardcoded Encryption Key (35 Issues)



Abstract:

Hardcoded 加密密钥可能会削弱系统安全性，一旦出现安全问题将无法轻易修正。

Explanation:

使用硬编码方式处理加密密钥绝非好方法。这不仅是因为所有项目开发人员都可以使用通过硬编码方式处理的加密密钥，而且还会使解决这一问题变得极其困难。在代码投入使用之后，必须对软件进行修补才能更改加密密钥。如果受加密密钥保护的帐户遭受入侵，系统所有者将必须在安全性和可用性之间做出选择。

示例：下列代码使用 hardcoded 加密密钥来加密信息：

```
...
from Crypto.Ciphers import AES
encryption_key = b'_hardcoded__key_'
cipher = AES.new(encryption_key, AES.MODE_CFB, iv)
msg = iv + cipher.encrypt(b'Attack at dawn')
...
```

此代码将成功运行，但任何有权访问此代码的人都可以获得加密密钥。一旦程序发布，除非修补该程序，否则可能无法更改硬编码的加密密钥 \_hardcoded\_\_key\_。心怀不轨的雇员可以利用其对此信息的访问权限来破坏系统加密的数据。

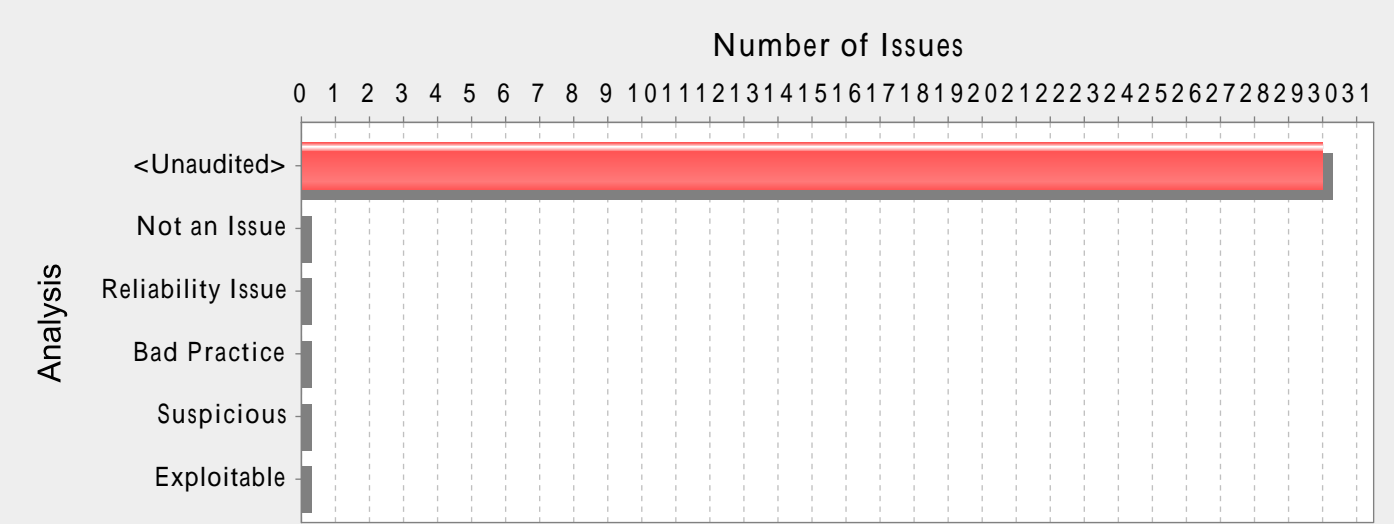
Recommendations:

绝不能对加密密钥进行硬编码。通常情况下，应对加密密钥加以模糊化，并在外部资源文件中进行管理。如果在系统中采用明文的形式存储加密密钥，任何有足够权限的人即可读取加密密钥，还可能误用这些密码。

bitvec.py, line 38 (Key Management: Hardcoded Encryption Key)

|                   |   |        |          |
|-------------------|---|--------|----------|
| Fortify Priority: | Critical                                    | Folder | Critical |
| Kingdom:          | Security Features                           |        |          |
| Abstract:         | Hardcoded 加密密钥可能会削弱系统安全性，一旦出现安全问题将无法轻易修正。   |        |          |
| Sink:             | bitvec.py:38 Operation()                    |        |          |
| 36                | if key < 0:                                 |        |          |
| 37                | key = key + len                             |        |          |
| 38                | if not 0 <= key < len:                      |        |          |
| 39                | raise IndexError, 'list index out of range' |        |          |
| 40                | return key                                  |        |          |

Category: Password Management: Hardcoded Password (30 Issues)



Abstract:

Hardcoded password 可能会削弱系统安全性，并且无法轻易修正出现的安全问题。

Explanation:

使用硬编码方式处理密码绝非好方法。这不仅是因为所有项目开发人员都可以使用通过硬编码方式处理的密码，而且还会使解决这一问题变得极其困难。在代码投入使用之后，除非对软件进行修补，否则将无法更改密码。如果受密码保护的帐户遭受入侵，系统所有者将必须在安全性和可用性之间做出选择。

示例：以下代码对密码进行了硬编码：

```
pwd = "tiger"
...
response.writeln("Password:" + pwd)
```

该代码可以正常运行，但是有权访问该代码的任何人都能得到这个密码。一旦程序发布，除非修补该程序，否则可能无法更改密码“tiger”。雇员可以利用手中掌握的信息访问权限入侵系统。更糟的是，如果攻击者能够访问应用程序的二进制码，他们就可以利用多种常用的反编译器来访问经过反汇编的代码，而在这些代码中恰恰包含着用户使用过的密码值。

Recommendations:

绝不能对密码进行硬编码。通常情况下，应对密码加以模糊化，并在外部资源文件中进行管理。在系统中采用明文的形式存储密码，会造成任何有充分权限的人读取和无意中误用密码。

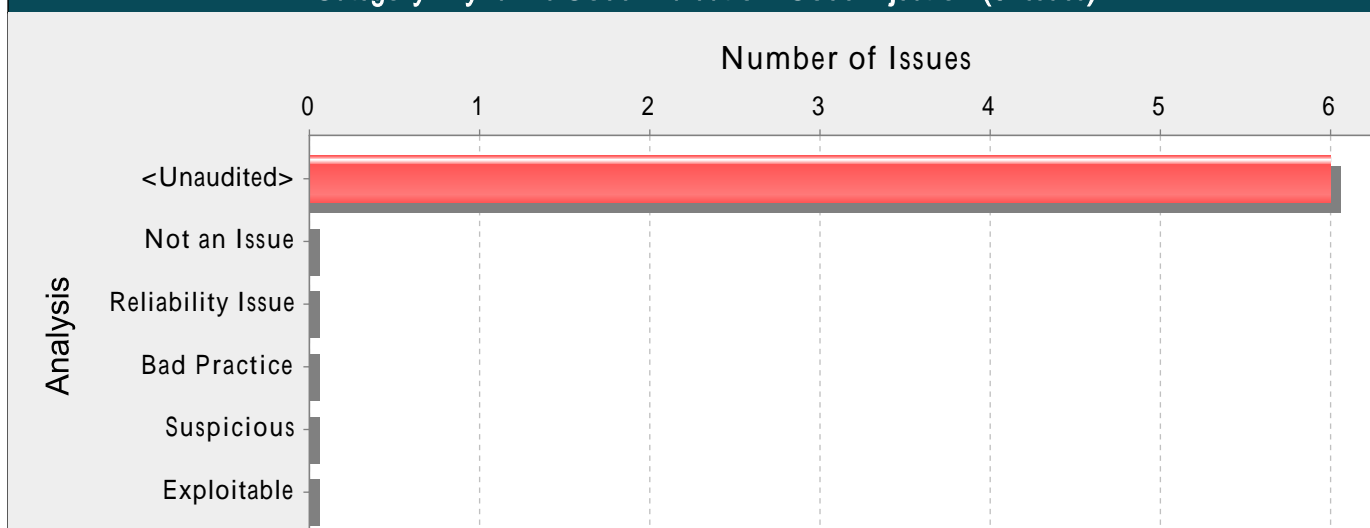
Tips:

- 1. 避免在源代码中对密码进行硬编码，还要避免使用默认密码。如果 hardcoded password 处于缺省状态，则需要修改密码，使其不出现在源代码中。
- 2. 识别 empty password 和 hardcoded password 时，默认规则只会考虑包含 password 字符的字段和变量。但是，Fortify Custom Rules Editor（Fortify 自定义规则编辑器）会提供 Password Management 向导，让您轻松创建能够从自定义名称的字段和变量中检测出 password management 问题的规则。

test\_register.py, line 22 (Password Management: Hardcoded Password)

|                   |   |        |      |
|-------------------|---|--------|------|
| Fortify Priority: | High  | Folder | High |
| Kingdom:          | Security Features                                       |        |      |
| Abstract:         | Hardcoded password 可能会削弱系统安全性，并且无法轻易修正出现的安全问题。          |        |      |
| Sink:             | test_register.py:22 VariableAccess: PYPIRC_NOPASSWORD() |        |      |
| 20                | docutils = None   |        |      |
| 21                |   |        |      |
| 22                | PYPIRC_NOPASSWORD =*****                                |        |      |
| 23                | [distutils]   |        |      |

## Category: Dynamic Code Evaluation: Code Injection (6 Issues)

**Abstract:**

在运行时中解析用户控制的指令，会让攻击者有机会执行恶意代码。

**Explanation:**

许多现代编程语言都允许动态解析源代码指令。这使得程序员可以执行基于用户输入的动态指令。当程序员错误地认为由用户直接提供的指令仅会执行一些无害的操作时（如对当前的用户对象进行简单的计算或修改用户的状态），就会出现 code injection 漏洞：然而，若经过适当的验证，用户指定的操作可能并不是程序员最初所期望的。

示例：在这个经典的 code injection 实例中，应用程序可以实施一个基本的计算器，该计算器允许用户指定执行命令。

```
...
userOps = request.GET['operation']
result = eval(userOps)
...
```

如果 operation 参数的值为良性值，程序就可以正常运行。例如，当该值为“8 + 7 \* 2”时，result 变量被赋予的值将为 22。然而，如果攻击者指定的语言操作是有效的，又是恶意的，那么，将在对主进程具有完全权限的情况下执行这些操作。如果底层语言提供了访问系统资源的途径或允许执行系统命令，这种攻击甚至会更加危险。例如，如果攻击者计划将“os.system('shutdown -h now')”指定为 operation 的值，主机系统就会执行关机命令。

**Recommendations:**

在任何时候，都应尽可能地避免动态的代码解析。如果程序的功能要求对代码进行动态的解析，您可以通过以下方式将此种攻击的可能性降低到最小：尽可能的限制程序中动态执行的代码数量，将此类代码应用到特定的应用程序和上下文中的基本编程语言的子集。

如果需要执行动态代码，应用程序绝不当直接执行和解析未验证的用户输入。而应采用间接方法：创建一份合法操作和数据对象列表，用户可以指定其中的内容，并且只能从中进行选择。利用这种方法，就绝不会直接执行由用户提供的输入。

## Dbm.py, line 48 (Dynamic Code Evaluation: Code Injection)

Fortify Priority: Critical Folder Critical

Kingdom: Input Validation and Representation

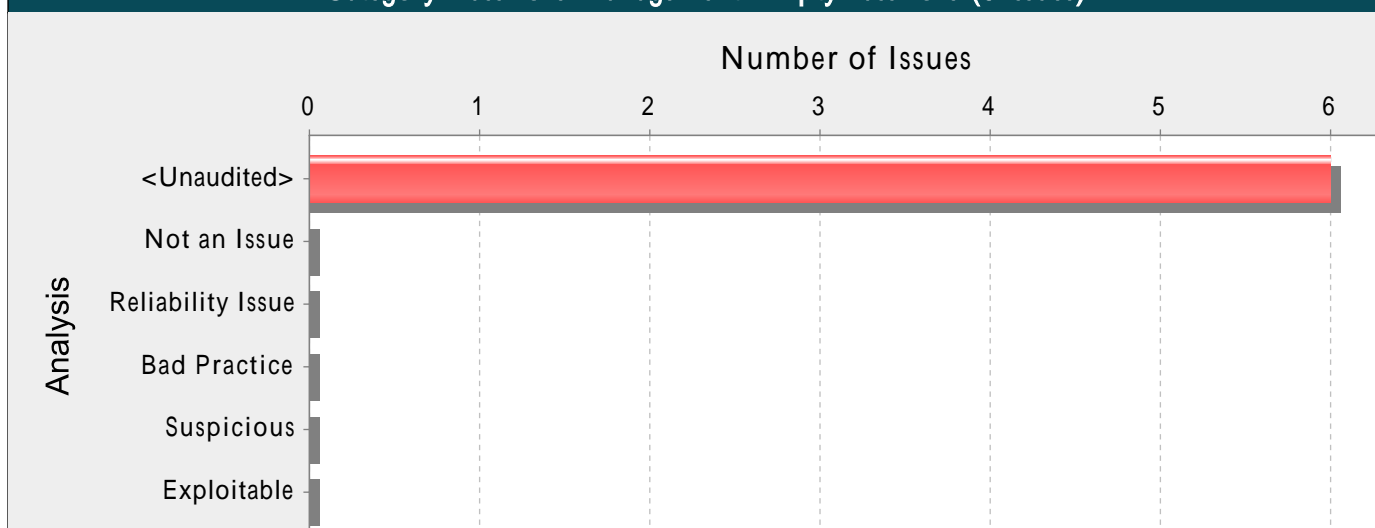
Abstract: 在运行时中解析用户控制的指令，会让攻击者有机会执行恶意代码。

Sink: Dbm.py:48 FunctionCall: input()

```
46     while 1:
47         try:
48             key = input('key: ')
49             if d.has_key(key):
50                 value = d[key]
```



## Category: Password Management: Empty Password (6 Issues)

**Abstract:**

Empty password 可能会危及系统安全，并且无法轻易修正出现的安全问题。

**Explanation:**

为密码变量指定空字符串绝非一个好方法。如果使用 empty password 成功通过其他系统的验证，那么相应帐户的安全性很可能会被减弱，原因是其接受了 empty password。如果在为变量指定一个合法的值之前，empty password 仅仅是一个占位符，那么它将给任何不熟悉代码的人造成困惑，而且还可能导致出现意外控制流路径方面的问题。

示例：以下代码尝试使用空密码连接到数据库。

```
...
db = mysql.connect("localhost","scott","", "mydb")
...
```

如果此示例中的代码成功执行，则表明数据库用户帐户“scott”配置有一个空密码，攻击者可以轻松地猜测到该密码。一旦程序发布，要更新此帐户以使用非空密码，就需要对代码进行更改。

**Recommendations:**

始终从加密的外部资源读取存储的密码值，并为密码变量指定有意义的值。确保从不使用空密码或 null 密码来保护敏感资源。

**Tips:**

1. 避免在源代码中使用 empty password，还要避免使用默认密码。
2. 在识别 null 密码、空密码和硬编码密码时，默认规则只会考虑包含 password 一词的字段和变量。但是，使用 Fortify Custom Rules Editor（Fortify 自定义规则编辑器）提供的“Password Management（密码管理）”向导可轻松创建用于在自定义命名字段和变量中检测 password management 问题的规则。

## register.py, line 136 (Password Management: Empty Password)

| Fortify Priority: | High | Folder | High |
|-------------------|------|--------|------|
|-------------------|------|--------|------|

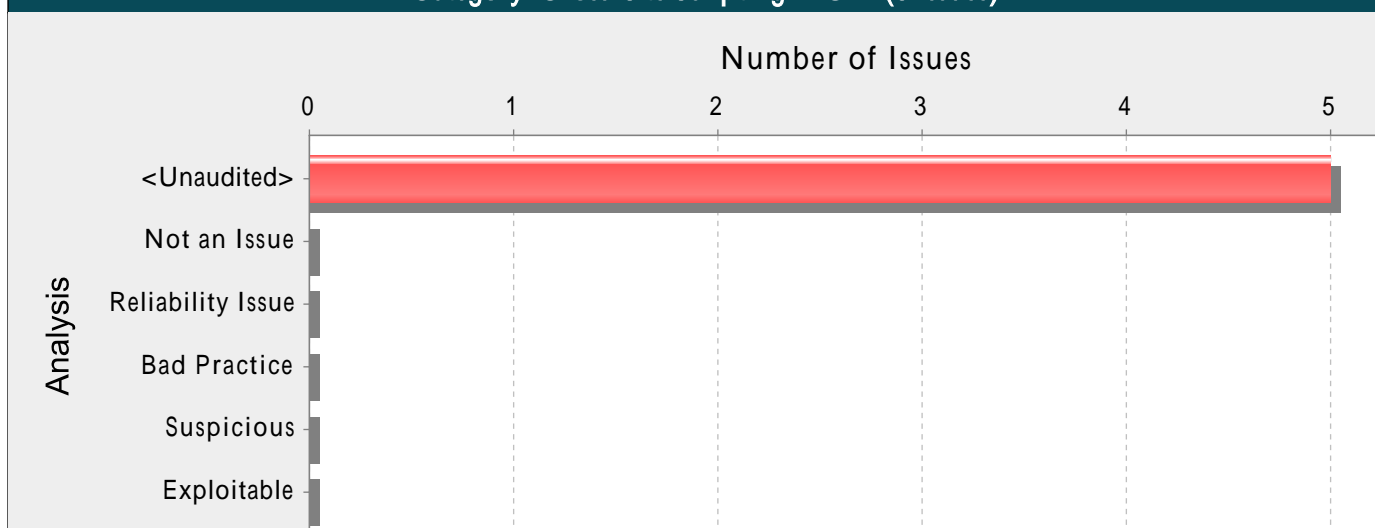
|          |                   |
|----------|-------------------|
| Kingdom: | Security Features |
|----------|-------------------|

**Abstract:** Empty password 可能会危及系统安全，并且无法轻易修正出现的安全问题。

**Sink:** register.py:136 VariableAccess: password()

```
134         else:
135             choice = 'x'
136             username = password = *****
137
138             # get the user's login info
```

## Category: Cross-Site Scripting: DOM (5 Issues)

**Abstract:**

switchers.js 中的方法 navigate\_to\_first\_existing() 向第 66 行的 Web 浏览器发送非法数据，从而导致浏览器执行恶意代码。

**Explanation:**

Cross-Site Scripting (XSS) 漏洞在以下情况下发生：

1. 数据通过一个不可信赖的数据源进入 Web 应用程序。对于基于 DOM 的 XSS，将从 URL 参数或浏览器中的其他值读取数据，并使用客户端代码将其重新写入该页面。对于 Reflected XSS，不可信赖的数据源通常为 Web 请求，而对于 Persisted（也称为 Stored）XSS，该数据源通常为数据库或其他后端数据存储。
2. 未检验包含在动态内容中的数据，便将其传送给了 Web 用户。对于基于 DOM 的 XSS，任何时候当受害人的浏览器解析 HTML 页面时，恶意内容都将作为 DOM（文档对象模型）创建的一部分执行。

传送到 Web 浏览器的恶意内容通常采用 JavaScript 代码片段的形式，但也可能会包含一些 HTML、Flash 或者其他任意一种可以被浏览器执行的代码。基于 XSS 的攻击手段花样百出，几乎是无穷无尽的，但通常它们都会包含传输给攻击者的私有数据（如 Cookie 或者其他会话信息）。在攻击者的控制下，指引受害者进入恶意的网络内容；或者利用易受攻击的站点，对用户的机器进行其他恶意操作。

例 1：下面的 JavaScript 代码片段可从 URL 中读取雇员 ID eid，并将其显示给用户。

```
<SCRIPT>
var pos=document.URL.indexOf("eid=")+4;
document.write(document.URL.substring(pos,document.URL.length));
</SCRIPT>
```

示例 2：考虑使用 HTML 表单：

```
<div id="myDiv">
Employee ID: <input type="text" id="eid"><br>
...
<button>Show results</button>
</div>
<div id="resultsDiv">
...
</div>
```

下面的 jQuery 代码片段可从表单中读取雇员 ID，并将其显示给用户。

```
$(document).ready(function(){
$("#myDiv").on("click", "button", function(){
var eid = $("#eid").val();
$("#resultsDiv").append(eid);
...
});
});
```

如果文本输入中 ID 为 eid 的雇员 ID 仅包含标准字母数字文本，则这些代码示例可正确运行。如果 eid 中的某个值包含元字符或源代码，则 Web 浏览器就会在显示 HTTP 响应时执行该代码。

示例 3：以下代码显示了 React 应用程序中基于 DOM 的 XSS 示例：

```
let element = JSON.parse(getUntrustedInput());
ReactDOM.render(<App>
{element}
</App>);
```

在 Example 3 中，如果攻击者可以控制从 getUntrustedInput() 检索到的整个 JSON 对象，他们可能就能够使 React 将 element 呈现为一个组件，从而可以使用他们自己控制的值传递具有 dangerouslySetInnerHTML 的对象，这是一种典型的 Cross-Site Scripting 攻击。

最初，这些代码看起来似乎不会轻易遭受攻击。毕竟，有谁会输入包含可在自己电脑上运行的恶意代码的内容呢？真正的危险在于攻击者会创建恶意的 URL，然后采用电子邮件或社交工程的欺骗手段诱使受害者访问此 URL 的链接。当受害者单击这个链接时，他们不知不觉地通过易受攻击的网络应用程序，将恶意内容带到了自己的电脑中。这种对易受攻击的 Web 应用程序进行盗取的机制通常被称为反射式 XSS。

正如例子中所显示的，XSS 漏洞是由于 HTTP 响应中包含了未验证的数据代码而引起的。受害者遭受 XSS 攻击的途径有三种：

- 系统从 HTTP 请求中直接读取数据，并在 HTTP 响应中返回数据。当攻击者诱使用户为易受攻击的 Web 应用程序提供危险内容，而这些危险内容随后会反馈给用户并在 Web 浏览器中执行时，就会发生 Reflected XSS 漏洞利用。发送恶意内容最常用的方法是，将恶意内容作为一个参数包含在公开发布或通过电子邮件直接发送给受害者的 URL 中。以这种手段构造的 URL 已成为多种网络钓鱼阴谋的核心，攻击者会借此诱骗受害者访问指向易受攻击站点的 URL。当该站点将攻击者的内容反馈给受害者后，便会执行这些内容，接下来会将用户计算机中的各种私密信息（比如可能包含会话信息的 Cookie）传输给攻击者，或者执行其他恶意活动。

— 应用程序将危险数据存储在数据库或其他可信赖的数据存储器中。这些危险数据随后会被回写到应用程序中，并包含在动态内容中。Persistent XSS 窃取发生在如下情况：攻击者将危险内容注入到数据存储器中，且该存储器之后会被读取并包含在动态内容中。从攻击者的角度看，注入恶意内容的最佳位置莫过于一个面向许多用户，尤其是相关用户显示的区域。相关用户通常在应用程序中具备较高的特权，或相互之间交换敏感数据，这些数据对攻击者来说有利用价值。如果某一个用户执行了恶意内容，攻击者就有可能以该用户的名义执行某些需要特权的操作，或者获得该用户个人所有的敏感数据的访问权限。

— 应用程序之外的数据源将危险数据储存在一个数据库或其他数据存储器中，随后这些危险数据被当作可信赖的数据回写到应用程序中，并储存在动态内容中。

## Recommendations:

针对 XSS 的解决方法是，确保在适当位置进行验证，并检验其属性是否正确。

由于 XSS 漏洞出现在应用程序的输出中包含恶意数据时，因此，合乎逻辑的做法是在数据流出应用程序的前一刻对其进行验证。然而，由于 Web 应用程序常常会包含复杂而难以理解的代码，用以生成动态内容，因此，这一方法容易产生遗漏错误（遗漏验证）。降低这一风险的有效途径是对 XSS 也执行输入验证。

由于 Web 应用程序必须验证输入信息以避免其他漏洞（如 SQL Injection），因此，一种相对简单的解决方法是，加强一个应用程序现有的输入验证机制，将 XSS 检测包括其中。尽管有一定的价值，但 XSS 输入验证并不能取代严格的输出验证。应用程序可能通过共享的数据存储或其他可信赖的数据源接受输入，而该数据存储所接受的输入源可能并未执行适当的输入验证。因此，应用程序不能间接地依赖于该数据或其他任意数据的安全性。这就意味着，避免 XSS 漏洞的最佳方法是验证所有进入应用程序以及由应用程序传送至用户端的数据。

针对 XSS 漏洞进行验证最安全的方式是，创建一份安全字符白名单，允许其中的字符出现在 HTTP 内容中，并且只接受完全由这些经认可的字符组成的输入。例如，有效的用户名可能仅包含字母数字字符，电话号码可能仅包含 0-9 的数字。然而，这种解决方法在 Web 应用程序中通常是行不通的，因为许多字符对浏览器来说都具有特殊的含义，在写入代码时，这些字符仍应被视为合法的输入，比如一个 Web 设计版就必须接受带有 HTML 代码片段的输入。

更灵活的解决方法称为黑名单方法，但其安全性较差，这种方法在进行输入之前就有选择地拒绝或避免了潜在的危险字符。为了创建这样一个列表，首先需要了解对于 Web 浏览器具有特殊含义的字符集。虽然 HTML 标准定义了哪些字符具有特殊含义，但是许多 Web 浏览器会设法更正 HTML 中的常见错误，并可能在特定的上下文中认为其他字符具有特殊含义，这就是我们不鼓励使用黑名单作为阻止 XSS 的方法的原因。卡耐基梅隆大学 (Carnegie Mellon University) 软件工程学院 (Software Engineering Institute) 下属的 CERT(R) (CERT(R) Coordination Center) 合作中心提供了有关各种上下文中认定的特殊字符的具体信息 [1]：

在有关块级元素的内容中（位于一段文本的中间）：

- "<" 是一个特殊字符，因为它可以引入一个标签。
- "&" 是一个特殊字符，因为它可以引入一个字符实体。
- ">" 是一个特殊字符，之所以某些浏览器将其认定为特殊字符，是基于一种假设，即该页的作者本想在前面添加一个 "<"，却错误地将其遗漏了。

下面的这些原则适用于属性值：

- 对于外加双引号的属性值，双引号是特殊字符，因为它们标记了该属性值的结束。
- 对于外加单引号的属性值，单引号是特殊字符，因为它们标记了该属性值的结束。

- 对于不带任何引号的属性值，空格字符（如空格符和制表符）是特殊字符。

- "&" 与某些特定变量一起使用时是特殊字符，因为它引入了一个字符实体。

例如，在 URL 中，搜索引擎可能会在结果页面内提供一个链接，用户可以点击该链接来重新运行搜索。可以将这一方法运用于编写 URL 中的搜索查询语句，这将引入更多特殊字符：

- 空格符、制表符和换行符是特殊字符，因为它们标记了 URL 的结束。

- "&" 是特殊字符，因为它可引入一个字符实体或分隔 CGI 参数。

- 非 ASCII 字符（即 ISO-8859-1 编码表中所有大于 127 的字符）不允许出现在 URL 中，因此这些字符在此环境下被视为特殊字符。

- 在服务器端对在 HTTP 转义序列中编码的参数进行解码时，必须过滤掉输入中的 "%" 符号。例如，当输入中出现 "%68%65%6C%6C%6F" 时，只有从输入的内容中过滤掉 "%"，上述字符串才能在网页上显示为 "hello"。

在 <SCRIPT> </SCRIPT> 的正文内：

- 如果可以将文本直接插入到已有的脚本标签中，应该过滤掉分号、省略号、中括号和换行符。

服务器端脚本：

- 如果服务器端脚本会将输入中的感叹号 (!) 转换成输出中的双引号 (")，则可能需要对此进行更多过滤。

其他可能出现的情况：

- 如果攻击者在 UTF-7 中提交了一个请求，那么特殊字符 "<" 可能会显示为 "+ADw-"，并可能会绕过过滤。如果输出包含在没有确切定义编码格式的网页中，有些浏览器就会设法根据内容自动识别编码（此处采用 UTF-7 格式）。

在应用程序中确定针对 XSS 攻击执行验证的正确要点，以及验证过程中要考虑的特殊字符之后，下一个难点就是确定验证过程中处理各种特殊字符的方式。如果应用程序认定某些特殊字符为无效输入，那么您可以拒绝任何带有这些无效特殊字符的输入。第二种选择就是采用过滤手段来删除这些特殊字符。然而，过滤的负面作用在于，过滤内容的显示将发生改变。在需要完整显示输入内容的情况下，过滤的这种负面作用可能是无法接受的。

如果必须接受带有特殊字符的输入，并将其准确地显示出来，验证机制一定要对所有特殊字符进行编码，以便删除其具有的含义。官方的 HTML 规范 [2] 提供了特殊字符对应的 ISO 8859-1 编码值的完整列表。

许多应用程序服务器都试图避免应用程序出现 Cross-Site Scripting 漏洞，具体做法是为负责设置特定 HTTP 响应内容的函数提供各种实现方式，以检验是否存在进行 Cross-Site Scripting 攻击必需的字符。不要依赖运行应用程序的服务器，以此确保该应用程序的安全。开发了某个应用程序后，并不能保证在其生命周期中它会在哪些应用程序服务器中运行。由于标准和已知盗取方式的演变，我们不能保证应用程序服务器也会保持同步。

### Tips:

1. Fortify 安全编码规则包将就 SQL Injection 和 Access Control 提出警告：当把不可信赖的数据写入数据库时，数据库将出现问题，并且会将数据库当作不可信赖的数据的来源，这会导致 XSS 漏洞。如果数据库在您的环境中是可信赖的资源，则使用自定义筛选器筛选出包含 DATABASE 污染标志或来自数据库源的数据流问题。尽管如此，对所有从数据库中读取的内容进行验证仍然是较好的做法。

2. 虽然使用 URL 对不可信数据进行编码可以防止许多 XSS 攻击，但部分浏览器（尤其是 Internet Explorer 6 和 7 以及其他浏览器）在将数据传递给 JavaScript 解释器之前，会自动在文档对象模型 (DOM) 中的特定位置对其内容进行解码。为了反映出其危险之处，规则包不再认为 URL 编码例程足以防御 cross-site scripting 攻击。如果对数据值进行 URL 编码并随后输出，Fortify 将会报告存在 Cross-Site Scripting: Poor Validation 漏洞。

3. 较早版本的 React 更容易受到 Cross-Site Scripting 攻击，因为它会控制整个文档。而较新的版本则会使用 Symbols 标识 React 组件，从而可以防止漏洞利用，但不支持 Symbol（本机或通过 Polyfill）的较早浏览器（例如所有版本的 Internet Explorer）仍然容易受到攻击。其他类型的 Cross-Site Scripting 攻击适用于所有浏览器以及各个版本的 React。

### switchers.js, line 66 (Cross-Site Scripting: DOM)

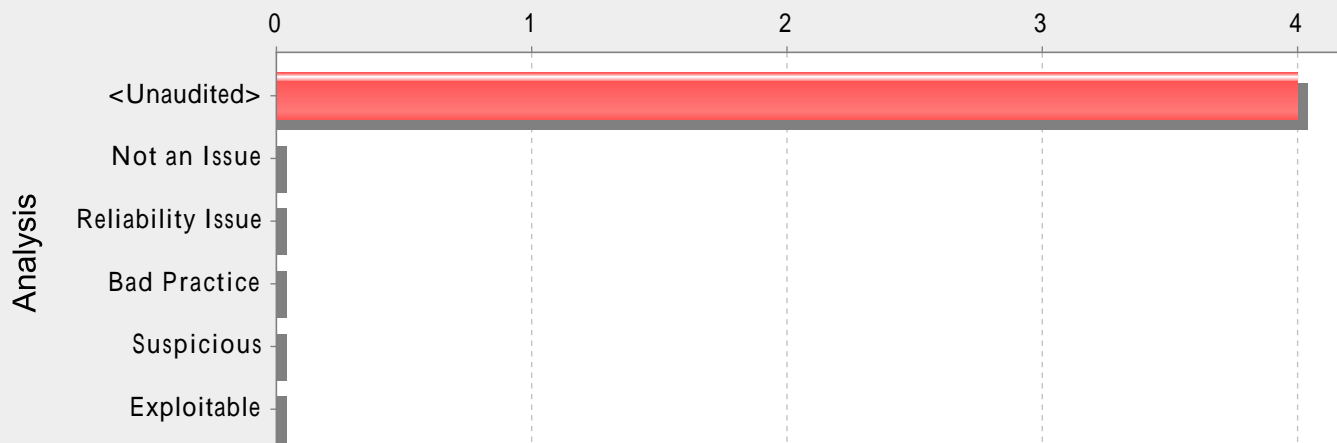
|                   |  |        |          |
|-------------------|--|--------|----------|
| Fortify Priority: | Critical   | Folder | Critical |
| Kingdom:          | Input Validation and Representation  |        |          |
| Abstract:         | switchers.js 中的方法 navigate_to_first_existing() 向第 66 行的 Web 浏览器发送非法数据，从而导致浏览器执行恶意代码。 |        |          |
| Source:           | switchers.js:82 Read window.location()   |        |          |
| 80                | function on_version_switch() {   |        |          |
| 81                | var selected_version = \$(this).children('option:selected').attr('value') + '/';     |        |          |
| 82                | var url = window.location.href;  |        |          |
| 83                | var current_language = language_segment_from_url(url);                               |        |          |
| 84                | var current_version = version_segment_in_url(url);                                   |        |          |
| Sink:             | switchers.js:66 Assignment to window.location.href()                                 |        |          |
| 64                | var url = urls.shift();  |        |          |
| 65                | if (urls.length == 0) {  |        |          |
| 66                | window.location.href = url;  |        |          |
| 67                | return;  |        |          |





## Category: Privacy Violation (4 Issues)

## Number of Issues

**Abstract:**

imaplib.py 文件会错误地处理第 1113 行的机密信息，从而危及到用户的个人隐私，这是一种非法行为。

**Explanation:**

Privacy Violation 会在以下情况下发生：

1. 用户私人信息进入了程序。
2. 数据被写到了一个外部介质，例如控制台、file system 或网络。

示例 1：以下代码包含了一个日志记录语句，该语句通过在日志文件中存储记录信息跟踪添加到数据库中的各条记录信息。在存储的其他数值中，有一个是 getPassword() 函数的返回值，该函数会返回与该帐户关联且由用户提供的明文密码。

```
pass = getPassword();
logger.warning('%s: %s %s %s', id, pass, type, tsstamp)
```

Example 1 中的代码会将明文密码记录到应用程序的事件日志中。虽然许多开发人员认为事件日志是存储数据的安全位置，但这并不是绝对的，特别是涉及到隐私问题时。

可以通过多种方式将私人数据输入到程序中：

- 以密码或个人信息的形式直接从用户处获取
- 由应用程序访问数据库或者其他数据存储形式
- 间接地从合作者或者第三方处获取

有时，某些数据并没有贴上私人数据标签，但在特定的上下文中也有可能成为私人信息。比如，通常认为学生的学号不是私人信息，因为学号中并没有明确而公开的信息用以定位特定学生的个人信息。但是，如果学校用学生的社会保障号码生成学号，那么这时学号应被视为私人信息。

安全和隐私似乎一直是一对矛盾。从安全的角度看，您应该记录所有重要的操作，以便日后可以鉴定那些非法的操作。然而，当其中牵涉到私人数据时，这种做法就存在一定风险了。

虽然私人数据处理不当的方式多种多样，但常见风险来自于盲目信任。程序员通常会信任运行程序的操作环境，因此认为将私人信息存放在文件系统、注册表或者其他本地控制的资源中是值得信任的。尽管已经限制了某些资源的访问权限，但仍无法保证所有访问这些资源的个体都是值得信任的。例如，2004 年，一个不道德的 AOL 员工将大约 9200 万个私有客户电子邮件地址卖给了一个通过垃圾邮件进行营销的境外赌博网站 [1]。

鉴于此类备受瞩目的信息盗取事件，私人信息的收集与管理正日益规范化。要求各个组织应根据其经营地点、所从事的业务类型及其处理的私人数据性质，遵守下列一个或若干个联邦和州的规定：

- Safe Harbor Privacy Framework [3]
- Gramm-Leach Bliley Act (GLBA) [4]
- Health Insurance Portability and Accountability Act (HIPAA) [5]
- California SB-1386 [6]

尽管制定了这些规范，Privacy Violation 漏洞仍时有发生。

**Recommendations:**

当安全和隐私的需要发生矛盾时，通常应优先考虑隐私的需要。为满足这一要求，同时又保证信息安全的需要，应在退出程序前清除所有私人信息。

为加强隐私信息的管理，应不断改进保护内部隐私的原则，并严格地加以执行。这一原则应具体说明应用程序应该如何处理各种私人数据。在贵组织受到联邦或者州法律的制约时，应确保您的隐私保护原则尽量与这些法律法规保持一致。即使没有针对贵组织的相应法规，您也应当保护好客户的私人信息，以免失去客户的信任。

保护私人数据的最好做法就是最大程度地减少私人数据的暴露。不应允许应用程序、流程处理以及员工访问任何私人数据，除非是出于职责以内的工作需要。正如最小授权原则一样，不应该授予访问者超出其需求的权限，对私人数据的访问权限应严格限制在尽可能小的范围内。

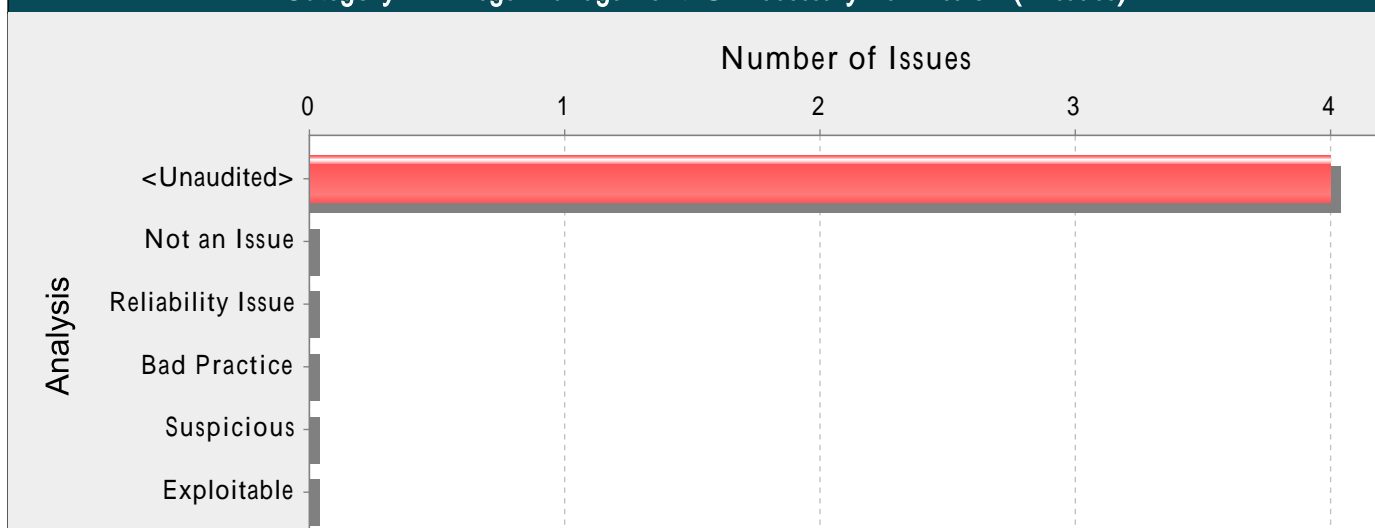
#### Tips:

1. 要彻底审计所有的 Privacy Violation 漏洞，措施之一就是确保自定义的规则可以识别所有进入程序的私人或敏感信息。无法自动识别多数私人数据信息。若不使用自定义规则，您执行的 Privacy Violation 漏洞检查可能是不完整的。

#### imaplib.py, line 1113 (Privacy Violation)

| Fortify Priority: | Critical   | Folder | Critical |
|-------------------|--|--------|----------|
| Kingdom:          | Security Features  |        |          |
| <b>Abstract:</b>  | imaplib.py 文件会错误地处理第 1113 行的机密信息，从而危及到用户的个人隐私，这是一种非法行为。  |        |          |
| <b>Source:</b>    | imaplib.py:1455 Read PASSWD()  |        |          |
| 1453              | test_msg = 'From: %(user)s@localhost%(lf)sSubject: IMAP4 test%(lf)s%(lf)sdata...%(lf)s' % {'user':USER, 'lf':'\n'} |        |          |
| 1454              | test_seq1 = (  |        |          |
| 1455              | ('login', (USER, PASSWD)),   |        |          |
| 1456              | ('create', ('/tmp/xxx 1',)),   |        |          |
| 1457              | ('rename', ('/tmp/xxx 1', '/tmp/yyy')),  |        |          |
| <b>Sink:</b>      | imaplib.py:1113 deps.Python.Lib.idlelib.PyShell.PseudoOutputFile.write()   |        |          |
| 1111              | secs = time.time()   |        |          |
| 1112              | tm = time.strftime('%M:%S', time.localtime(secs))  |        |          |
| 1113              | sys.stderr.write(' %s.%02d %s\n' % (tm, (secs*100)%100, s))  |        |          |
| 1114              | sys.stderr.flush()   |        |          |

## Category: Privilege Management: Unnecessary Permission (4 Issues)

**Abstract:**

应用程序若不能遵守最低权限原则，便会大大增加引发其他漏洞的风险。

**Explanation:**

应用程序应仅拥有正常执行所需的最小权限。权限过多会导致用户不愿意安装该应用程序。此权限对于该程序可能是不必要的。

**Recommendations:**

考虑应用程序是否需要请求的权限来保证正常运行。如果不需要，则应将相应的权限从 AndroidManifest.xml 文件中删除。除了请求应用程序真正需要的权限之外，切忌因请求更多权限而导致对应用程序过度授权。这会导致在设备上安装的其他恶意应用程序利用这种过度授权的应用程序对用户体验及存储的数据造成负面影响。另外，设置过多的权限可能会适得其反，导致客户不愿意安装您的应用程序。

## AndroidManifest.xml, line 30 (Privilege Management: Unnecessary Permission)

Fortify Priority: High Folder High

Kingdom: Security Features

Abstract: 应用程序若不能遵守最低权限原则，便会大大增加引发其他漏洞的风险。

Sink: AndroidManifest.xml:30 null()

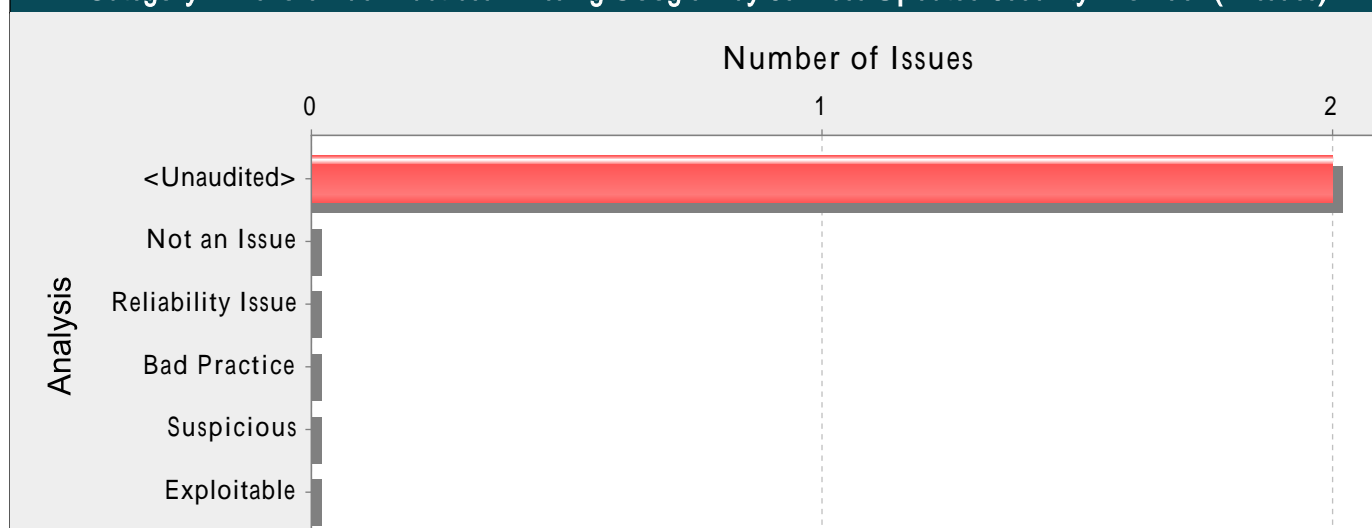
```

28
29      <!-- Allow writing to external storage -->
30      <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
31      <!-- Allow access to the vibrator -->
32      <uses-permission android:name="android.permission.VIBRATE" />

```



## Category: Android Bad Practices: Missing Google Play Services Updated Security Provider (2 Issues)

**Abstract:**

应用程序不使用 Google Play 服务更新的安全提供程序，这可能使其未来易遭受 OpenSSL 库中漏洞的攻击。

**Explanation:**

Android 依赖于可提供安全网络通信的安全提供程序。但是，有时漏洞存在于默认安全提供程序中。为了防范这些漏洞，Google Play 服务可提供用于自动更新设备安全提供程序的方法，以防御已知盗取手段。通过调用 Google Play 服务方法，您的应用程序可以确保其在具有最新更新的设备上运行，以防御已知盗取手段。

**Recommendations:**

修补安全提供程序最简单的方法是调用同步法 `installIfNeeded()`。如果在等待操作完成的过程中用户体验不会受到线程阻止的影响，则此方法适用，否则它应该以异步方式完成。

示例：以下代码可实现用于更新安全提供程序的同步适配器。由于同步适配器在后台运行，因此在等待安全提供程序更新的过程中若出现线程阻止也没有影响。同步适配器调用 `installIfNeeded()` 以更新安全提供程序。如果方法正常返回，则同步适配器了解安全提供程序为最新程序。如果方法抛出异常，则同步适配器可采取相应的操作（如提示用户更新 Google Play 服务）。

```
public class SyncAdapter extends AbstractThreadedSyncAdapter {
...
// This is called each time a sync is attempted; this is okay, since the
// overhead is negligible if the security provider is up-to-date.
@Override
public void onPerformSync(Account account, Bundle extras, String authority, ContentProviderClient provider, SyncResult
syncResult) {
try {
ProviderInstaller.installIfNeeded(getContext());
} catch (GooglePlayServicesRepairableException e) {
// Indicates that Google Play services is out of date, disabled, etc.
// Prompt the user to install/update/enable Google Play services.
GooglePlayServicesUtil.showErrorNotification(e.getConnectionStatusCode(), getContext());
// Notify the SyncManager that a soft error occurred.
syncResult.stats.numIOExceptions++;
return;
} catch (GooglePlayServicesNotAvailableException e) {
// Indicates a non-recoverable error; the ProviderInstaller is not able
// to install an up-to-date Provider.
// Notify the SyncManager that a hard error occurred.
syncResult.stats.numAuthExceptions++;
return;
}
// If this is reached, you know that the provider was already up-to-date,
```

```
// or was successfully updated.  
}  
}
```

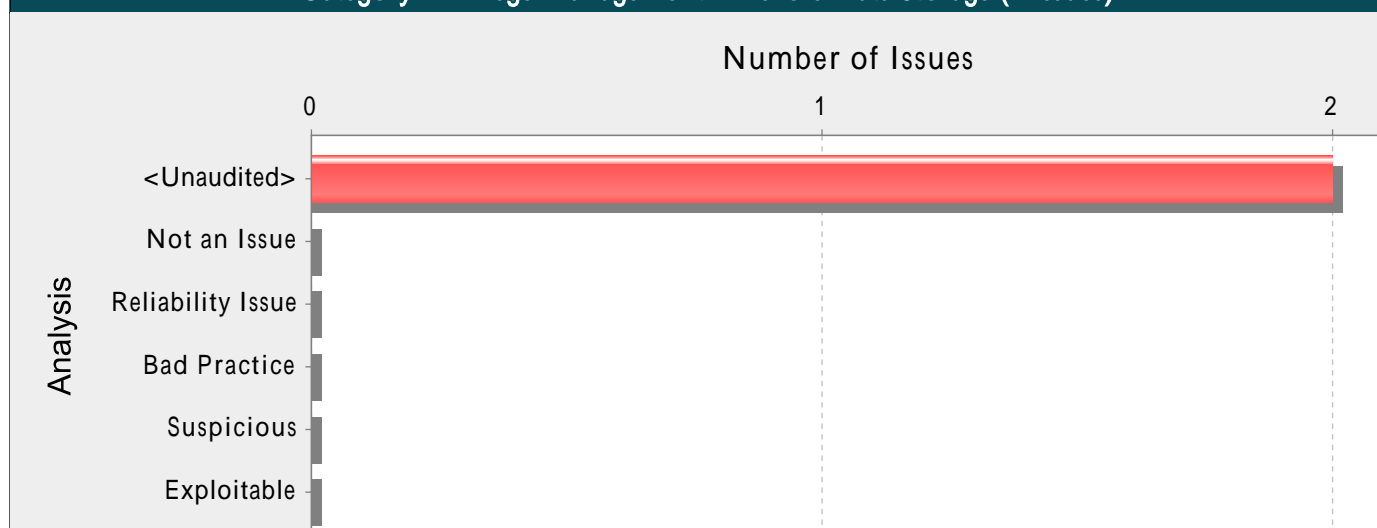
AndroidManifest.xml, line 49 (Android Bad Practices: Missing Google Play Services Updated Security Provider)

|                   |                   |        |      |
|-------------------|-------------------|--------|------|
| Fortify Priority: | High              | Folder | High |
| Kingdom:          | Security Features |        |      |

**Abstract:** 应用程序不使用 Google Play 服务更新的安全提供程序，这可能使其未来易遭受 OpenSSL 库中漏洞的攻击。

**Sink:** AndroidManifest.xml:49 null()  
47        android:allowBackup="true"  
48        android:theme="@android:style/Theme.NoTitleBar.Fullscreen"  
49        android:hardwareAccelerated="true" >  
50  
51        <!-- Example of setting SDL hints from AndroidManifest.xml:

## Category: Privilege Management: Android Data Storage (2 Issues)

**Abstract:**

程序在 AndroidManifest.xml 的第 30 行请求将数据写入 Android 外部存储的权限。

**Explanation:**

写入外部存储的文件可被任意程序与用户读写。程序不可将个人可识别信息等敏感信息写入外部存储中。通过 USB 将 Android 设备连接到电脑或其他设备时，就会启用 USB 海量存储模式。在此模式下，可以读取和修改写入外部存储的任意文件。此外，即使卸载了写入文件的应用程序，这些文件仍会保留在外部存储中，因而提高了敏感信息被盗用的风险。

例 1：AndroidManifest.xml 的 <uses-permission .../> 元素包含危险属性。

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

**Recommendations:**

请勿将以后要使用的受信敏感信息或数据写入外部存储中。而应将其写入程序特定的位置，例如 SQLite 数据库（由 Android 平台提供）。程序内的任意类都可以按名称访问您所创建的任意数据库，而程序外的类则不能。

例 2.通过创建 SQLiteOpenHelper 的子类和替代 onCreate() 方法来创建一个新的 SQLite 数据库。

```
public class MyDbOpenHelper extends SQLiteOpenHelper {
    private static final int DATABASE_VERSION = 2;
    private static final String DICTIONARY_TABLE_NAME = "dictionary";
    private static final String DICTIONARY_TABLE_CREATE =
        "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
        KEY_WORD + " TEXT, " +
        KEY_DEFINITION + " TEXT);";

    DictionaryOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DICTIONARY_TABLE_CREATE);
    }
}
```

另一种选择则是写入该设备的内部存储中。默认情况下，保存到内部存储中的文件为该程序专用的，其他程序和用户无法直接访问。用户卸载程序时，保存在内部存储中的文件也会随之删除，保证不会留下任何重要的信息。

例 3：以下代码创建了一个专用文件并将其写入设备的内部存储中。此 Context.MODE\_PRIVATE 声明会创建一个文件（或是替换同名文件），并将其设定为当前程序的专用文件。

```
String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(string.getBytes());
```

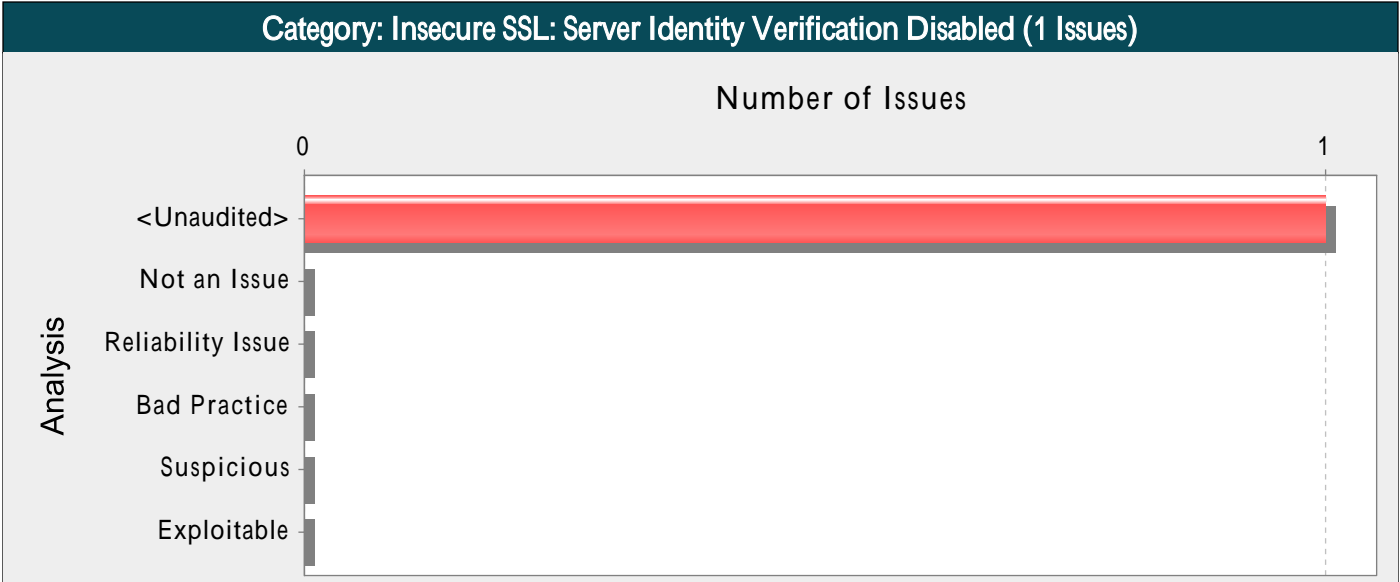
fos.close();

AndroidManifest.xml, line 30 (Privilege Management: Android Data Storage)

|                   |                   |        |      |
|-------------------|-------------------|--------|------|
| Fortify Priority: | High              | Folder | High |
| Kingdom:          | Security Features |        |      |

**Abstract:** 程序在 AndroidManifest.xml 的第 30 行请求将数据写入 Android 外部存储的权限。

|              |  |
|--------------|--|
| <b>Sink:</b> | AndroidManifest.xml:30 null()  |
| 28           |  |
| 29           | <!-- Allow writing to external storage -->                                   |
| 30           | <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" /> |
| 31           | <!-- Allow access to the vibrator -->  |
| 32           | <uses-permission android:name="android.permission.VIBRATE" />                |



Abstract:

在进行 SSL 连接时，通过 test\_urllibnet.py 中的 urlopen() 建立的连接不验证服务器证书。这使得应用程序易受到中间人攻击。

Explanation:

在一些使用 SSL 连接的库中，可以禁用服务器证书验证。这相当于信任所有证书。

例 1：此应用程序在默认情况下不会验证服务器证书：

```
...
import ssl
ssl_sock = ssl.wrap_socket(s)
...
```

当尝试连接到有效主机时，此应用程序将随时接受颁发给“hackedserver.com”的证书。此时，当服务器被黑客攻击发生 SSL 连接中断时，应用程序可能会泄漏用户敏感信息。

Recommendations:

当进行 SSL 连接时，不要忘记服务器验证检查。根据所使用的库，一定要验证服务器身份并建立安全的 SSL 连接。

例 2：此应用程序明确地验证服务器证书。

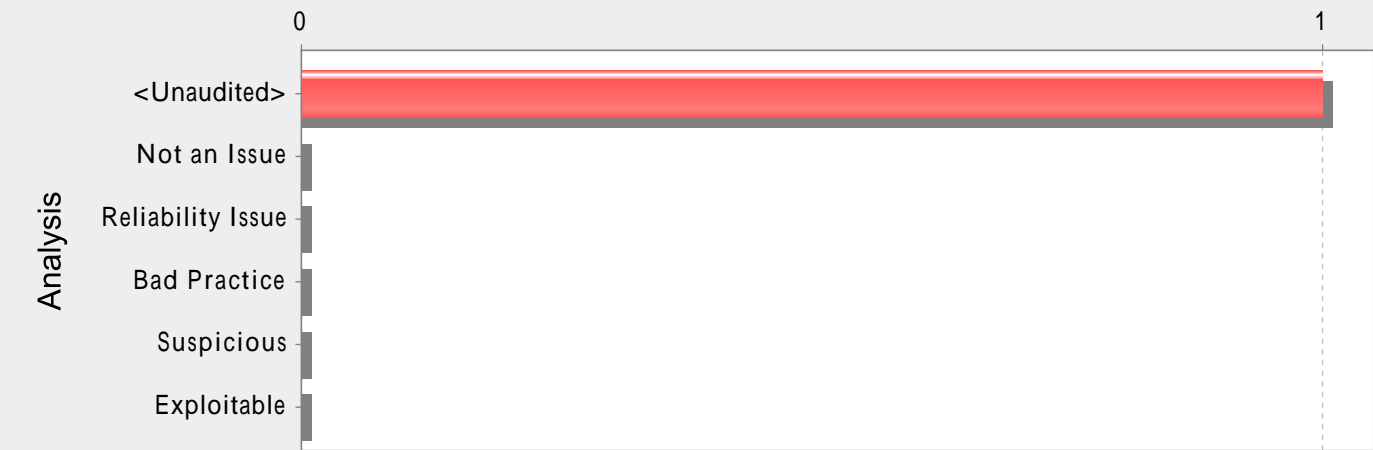
```
...
ssl_sock = ssl.wrap_socket(s, ca_certs="/etc/ca_certs_file", cert_reqs=ssl.CERT_REQUIRED)
...
```

test\_urllibnet.py, line 213 (Insecure SSL: Server Identity Verification Disabled)

|                   |  |        |          |
|-------------------|--|--------|----------|
| Fortify Priority: | Critical   | Folder | Critical |
| Kingdom:          | Security Features  |        |          |
| Abstract:         | 在进行 SSL 连接时，通过 test_urllibnet.py 中的 urlopen() 建立的连接不验证服务器证书。这使得应用程序易受到中间人攻击。     |        |          |
| Sink:             | test_urllibnet.py:213 FunctionCall: urlopen()                                    |        |          |
| 211               | def test_context_argument(self):   |        |          |
| 212               | context = ssl.create_default_context(cafile=CERT_selfsigned_pythontestdotnet)    |        |          |
| 213               | response = urllib.urlopen("https://self-signed.pythontest.net", context=context) |        |          |
| 214               | self.assertIn("Python", response.read())   |        |          |

Category: Path Manipulation (1 Issues)

Number of Issues



Abstract:

攻击者可以控制 test\_urllib.py 中第 463 行的 \_\_init\_\_() 文件系统路径参数，借此访问或修改其他受保护的文件。

Explanation:

当满足以下两个条件时，就会产生 path manipulation 错误：

- 1. 攻击者可以指定某一文件系统操作中所使用的路径。
  - 2. 攻击者可以通过指定特定资源来获取某种权限，而这种权限在一般情况下是不可能获得的。
- 例如，在某一程序中，攻击者可以获得特定的权限，以重写指定的文件或是在其控制的配置环境下运行程序。

例 1：下面的代码使用来自于 HTTP 请求的输入来创建一个文件名。程序员没有考虑到攻击者可能使用像 "../tomcat/conf/server.xml" 一样的文件名，从而导致应用程序删除它自己的配置文件。

```
rName = req.field('reportName')
rFile = os.open("/usr/local/apfr/reports/" + rName)
...
os.unlink(rFile);
```

示例 2：以下代码使用来自于配置文件的输入来决定打开哪个文件，并返回给用户。如果程序以足够的权限运行，且恶意用户能够篡改配置文件，那么他们可以通过程序读取系统中以扩展名 .txt 结尾的任何文件。

```
...
filename = CONFIG_TXT['sub'] + ".txt";
handle = os.open(filename)
print handle
...
```

Recommendations:

防止 path manipulation 的最佳方法是采用一些间接手段：例如创建一份合法资源名的列表，并且规定用户只能选择其中的文件名。通过这种方法，用户就不能直接由自己来指定资源的名称了。

但在某些情况下，这种方法并不可行，因为这样一份合法资源名的列表过于庞大、难以跟踪。因此，程序员通常在这种情况下采用黑名单的办法。在输入之前，黑名单会有选择地拒绝或避免潜在的危险字符。但是，任何这样一份黑名单都不可能是完整的，而且将随着时间的推移而过时。更好的方法是创建一份白名单，允许其中的字符出现在资源名称中，且只接受完全由这些被认可的字符组成的输入。

Tips:

- 1. 如果程序正在执行您认为合理的自定义输入验证，请使用 Fortify Custom Rules Editor（自定义规则编辑器）为该验证例程创建清理规则。
- 2. 执行有效的黑名单是一件非常困难的事情。如果验证逻辑依赖于黑名单，那么有必要对这种逻辑进行质疑。鉴于不同类型的输入编码以及各种元字符集在不同的操作系统、数据库或其他资源中可能有不同的含义，确定随着需求的不变化，黑名单能否方便、正确、完整地进行更新。

test\_urllib.py, line 463 (Path Manipulation)

|                   |                                     |        |          |
|-------------------|-------------------------------------|--------|----------|
| Fortify Priority: | Critical                            | Folder | Critical |
| Kingdom:          | Input Validation and Representation |        |          |

|                  |  |
|------------------|--|
| <b>Abstract:</b> | 攻击者可以控制 test_urllib.py 中第 463 行的 __init__() 文件系统路径参数，借此访问或修改其他受保护的文件。  |
| <b>Source:</b>   | test_urllib.py:459 urllib.urlretrieve()<br>457 self.registerFileForCleanUp(second_temp)<br>458 result = urllib.urlretrieve(self.constructLocalFileUrl(<br>459 test_support.TESTFN), second_temp)<br>460 self.assertEqual(second_temp, result[0])<br>461 self.assertTrue(os.path.exists(second_temp), "copy of the file was not " |
| <b>Sink:</b>     | test_urllib.py:463 file.__init__()<br>461 self.assertTrue(os.path.exists(second_temp), "copy of the file was not "<br>462 "made")<br>463 FILE = file(second_temp, 'rb')<br>464 try:<br>465 text = FILE.read()  |

## Issue Count by Category

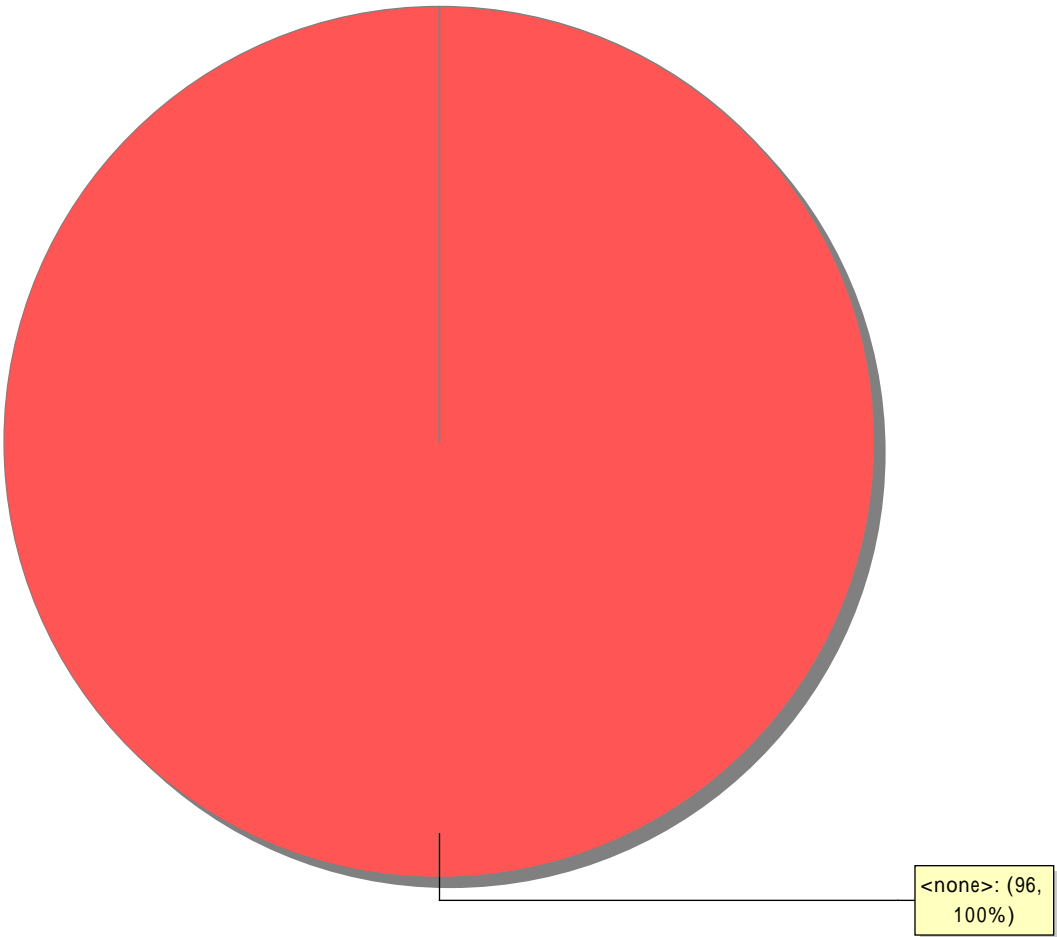
### Issues by Category


|   |    |
|---|----|
| Key Management: Hardcoded Encryption Key                                      | 35 |
| Password Management: Hardcoded Password                                       | 30 |
| Dynamic Code Evaluation: Code Injection                                       | 6  |
| Password Management: Empty Password   | 6  |
| Cross-Site Scripting: DOM   | 5  |
| Privacy Violation   | 4  |
| Privilege Management: Unnecessary Permission                                  | 4  |
| Android Bad Practices: Missing Google Play Services Updated Security Provider | 2  |
| Privilege Management: Android Data Storage                                    | 2  |
| Insecure SSL: Server Identity Verification Disabled                           | 1  |
| Path Manipulation   | 1  |



# Issue Breakdown by Analysis

Issues by Analysis



 <none>