Fortify Security Report

2024-6-21

ASUS

Executive Summary

Issues Overview

On 2024-6-21, a source code review was performed over the awtk code base. 474 files, 14,708 LOC (Executable) were scanned and reviewed for defects that could lead to potential security vulnerabilities. A total of 11 reviewed findings were uncovered during the analysis.

Issues by Fortify Priority Order	
High	8
Critical	3

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/awtk
Number of Files: 474
Lines of Code: 14708
Build Label: <No Build Label>

Scan Information

Scan time: 01:57
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
os.null.open
urllib.URLopener.open

Private Information:
null.null.null

Serialized Data:

cPickle.null.load

Standard Input Stream:
null.null.null
null.null.raw_input

System Information:
null.null.null
null.null.null
null.~JS_Generic.cwd
java.lang.System.getProperty
os.null.getcwd
os.null.getcwdu
os.null.listdir
os.null.uname
sys.null.exc_info

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

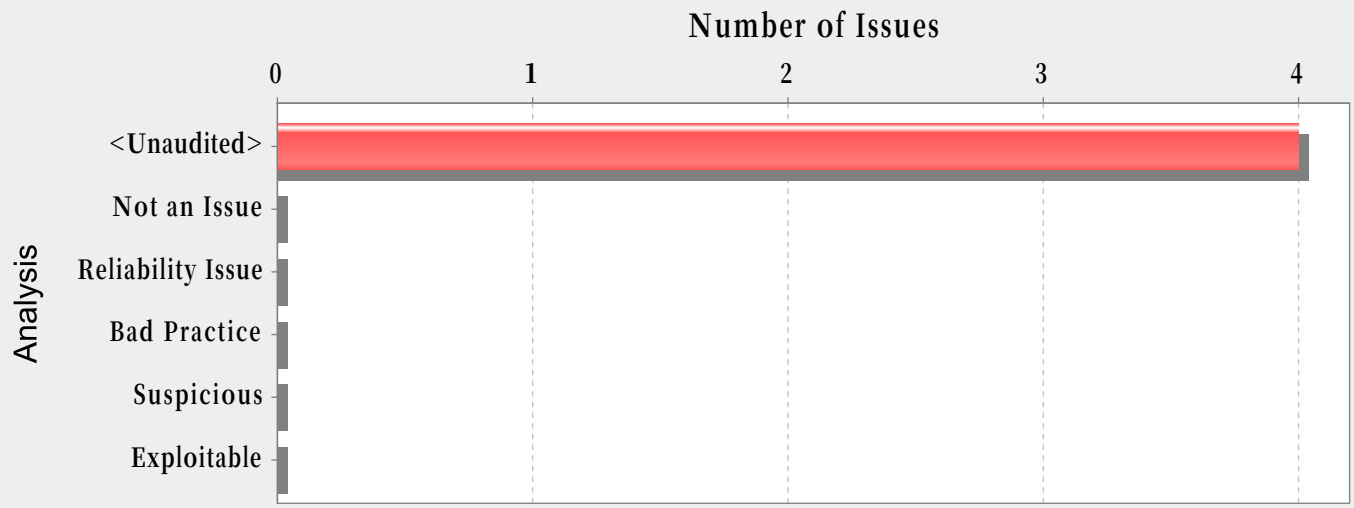
Results Outline

Overall number of results

The scan found 11 issues.

Vulnerability Examples by Category

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。

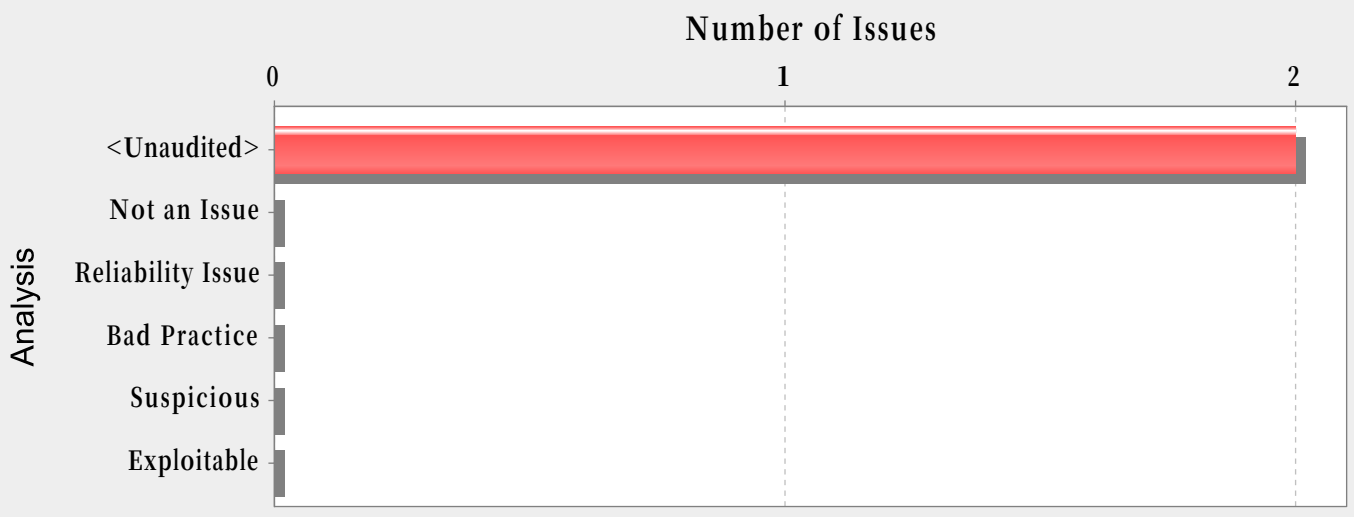
Recommendations:
考虑应用程序是否需要请求的权限来保证正常运行。如果不需要，则应将相应的权限从 AndroidManifest.xml 文件中删除。除了请求应用程序真正需要的权限之外，切忌因请求更多权限而导致对应用程序过度授权。这会导致在设备上安装的其他恶意应用程序利用这种过度授权的应用程序对用户体验及存储的数据造成负面影响。另外，设置过多的权限可能会适得其反，导致客户不愿意安装您的应用程序。

AndroidManifest.xml, line 33 (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:33 null()		

```
31
32    <!-- Allow writing to external storage -->
33    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
34    <!-- Allow access to the vibrator -->
35    <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）。

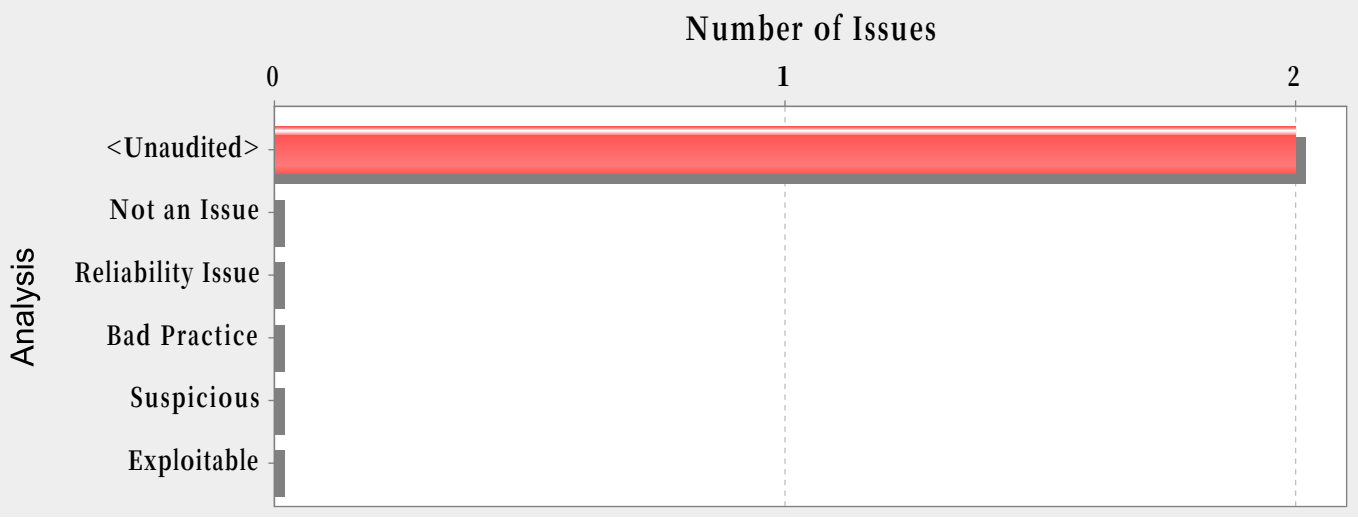
```
public class SyncAdapter extends AbstractThreadedSyncAdapter {

...

// 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 52 (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:52 null()		

```
50      android:allowBackup="true"
51      android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
52      android:hardwareAccelerated="true" >
53
54      <!-- Example of setting SDL hints from AndroidManifest.xml:
```

Category: Privacy Violation (2 Issues)



Abstract:

upload.py 文件会错误地处理第 165 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。

upload.py, line 165 (Privacy Violation)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Security Features		
Abstract:	upload.py 文件会错误地处理第 165 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。		

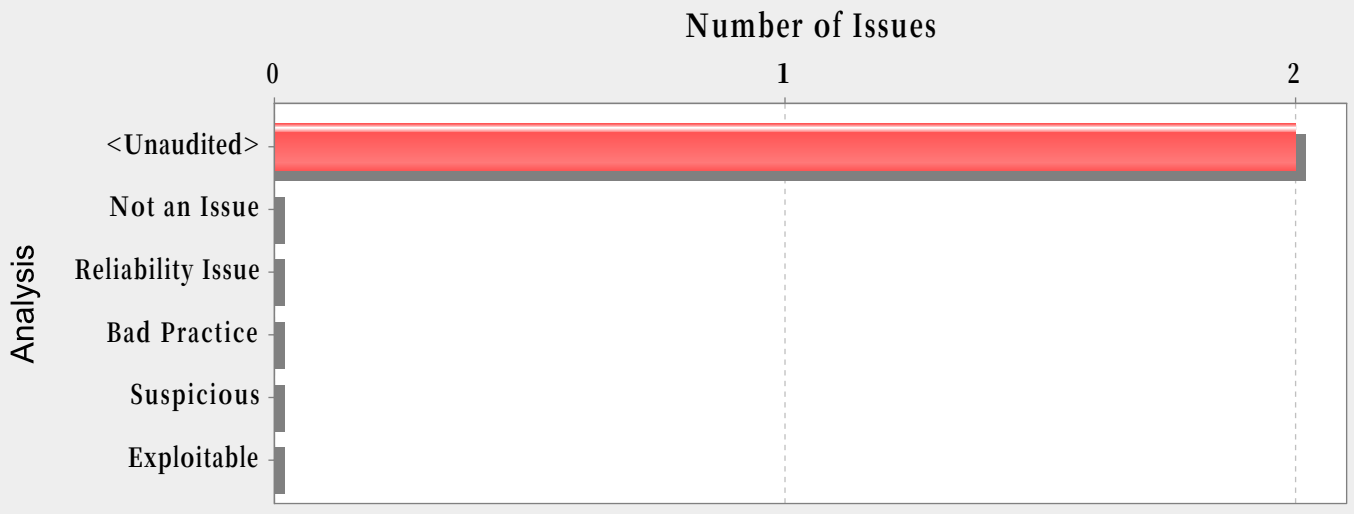
Source: upload.py:194 Read password()

```
192     data=urllib.urlencode({
193       "Email": email,
194       "Passwd": password,
195       "service": "ah",
196       "source": "rietveld-codereview-upload",
```

Sink: upload.py:165 urllib2.Request.__init__()

```
163   """Creates a new urllib request."""
164   logging.debug("Creating request for: '%s' with payload:\n%s", url, data)
165   req = urllib2.Request(url, data=data)
166   if self.host_override:
167     req.add_header("Host", self.host_override)
```

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



Abstract:

程序在 AndroidManifest.xml 的第 33 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 .../%gt; 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。

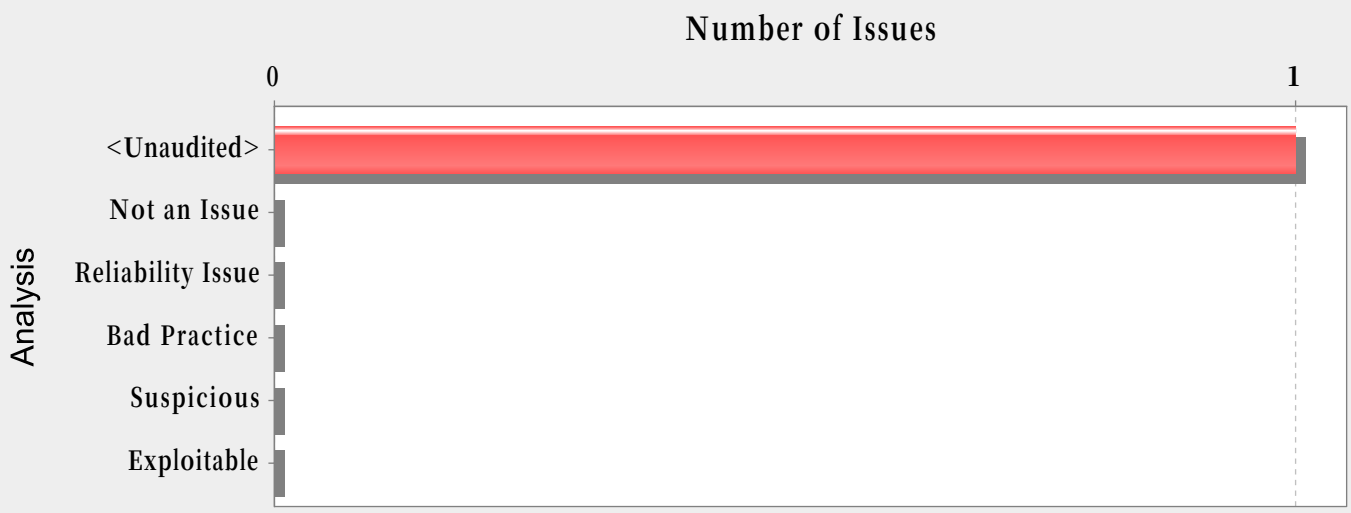
```
String FILENAME = "hello_file";
String string = "hello world!";

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

fos.close();

AndroidManifest.xml, line 33 (Privilege Management: Android Data Storage)			
Fortify Priority:	High	**Folder**	High
Kingdom:	Security Features		
Abstract:	程序在 AndroidManifest.xml 的第 33 行请求将数据写入 Android 外部存储的权限。		
Sink:	AndroidManifest.xml:33 null()		

```
31
32    <!-- Allow writing to external storage -->
33    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
34    <!-- Allow access to the vibrator -->
35    <uses-permission android:name="android.permission.VIBRATE" />
```

Category: Key Management: Hardcoded Encryption Key (1 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。

ast.py, line 1403 (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: ast.py:1403 Operation()

```
1401        key = tokens[i].name
1402        i += 1
1403        if keywords.IsKeyword(key) or key == ',':
1404            continue
1405        type_name = default = None
```

Issue Count by Category

Issues by Category	
Privilege Management: Unnecessary Permission	4
Android Bad Practices: Missing Google Play Services Updated Security Provider	2
Privacy Violation	2
Privilege Management: Android Data Storage	2
Key Management: Hardcoded Encryption Key	1

Issue Breakdown by Analysis

Issues by Analysis