Fortify Security Report

2024-6-21

ASUS

Executive Summary

Issues Overview

On 2024-6-21, a source code review was performed over the RetroArch code base. 344 files, 7,297 LOC (Executable) were scanned and reviewed for defects that could lead to potential security vulnerabilities. A total of 24 reviewed findings were uncovered during the analysis.

Issues by Fortify Priority Order	
High	18
Critical	6

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/RetroArch
Number of Files: 344
Lines of Code: 7297
Build Label: <No Build Label>

Scan Information

Scan time: 02:31
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
null.T0Comp.Main

Environment Variables:
null.null.null
os.null.getenv

File System:
null.null._handleFileTreeEntry
null.null.g
null.null.open
null.file.read
null.file.readlines
null.~JS_Generic.readEntries
java.io.FileInputStream.FileInputStream
java.io.FileInputStream.FileInputStream

Stream:
java.io.InputStream.read

System Information:
null.null.null
null.null.null
null.null._handleFileTreeEntry
null.null._readdir
null.null._remove
null.null.g
null.null.i
null.null.lambda
null.null.mkdir
null.null.open
null.null.pipe
null.null.rename
null.null.sync
java.io.File.listFiles
java.lang.Throwable.getMessage
os.null.listdir

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

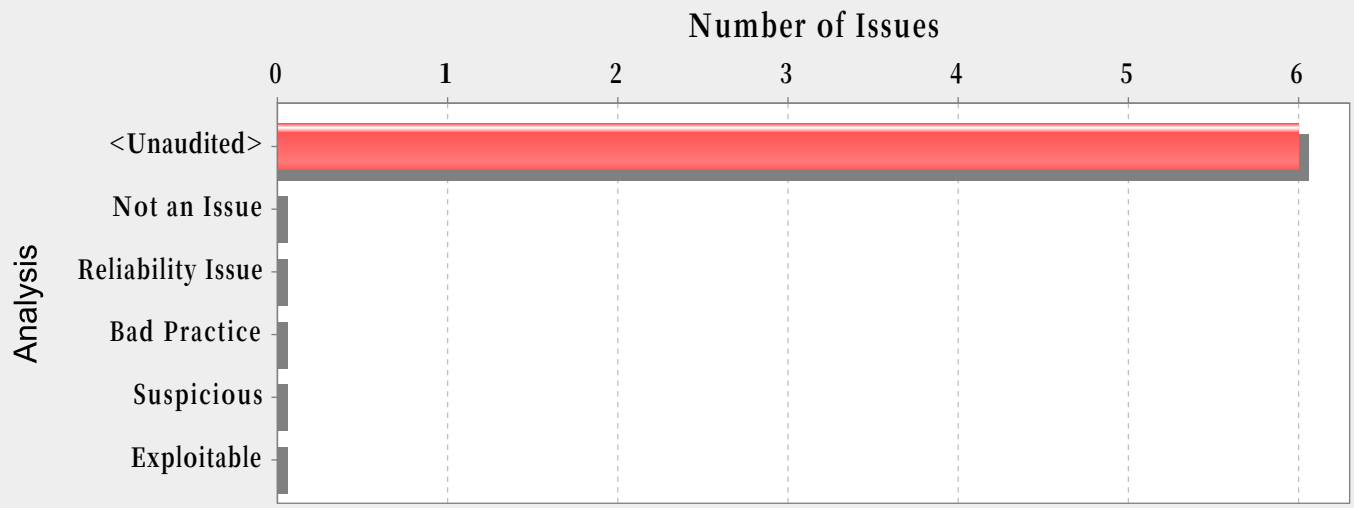
Results Outline

Overall number of results

The scan found 24 issues.

Vulnerability Examples by Category

Category: Privilege Management: Unnecessary Permission (6 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。

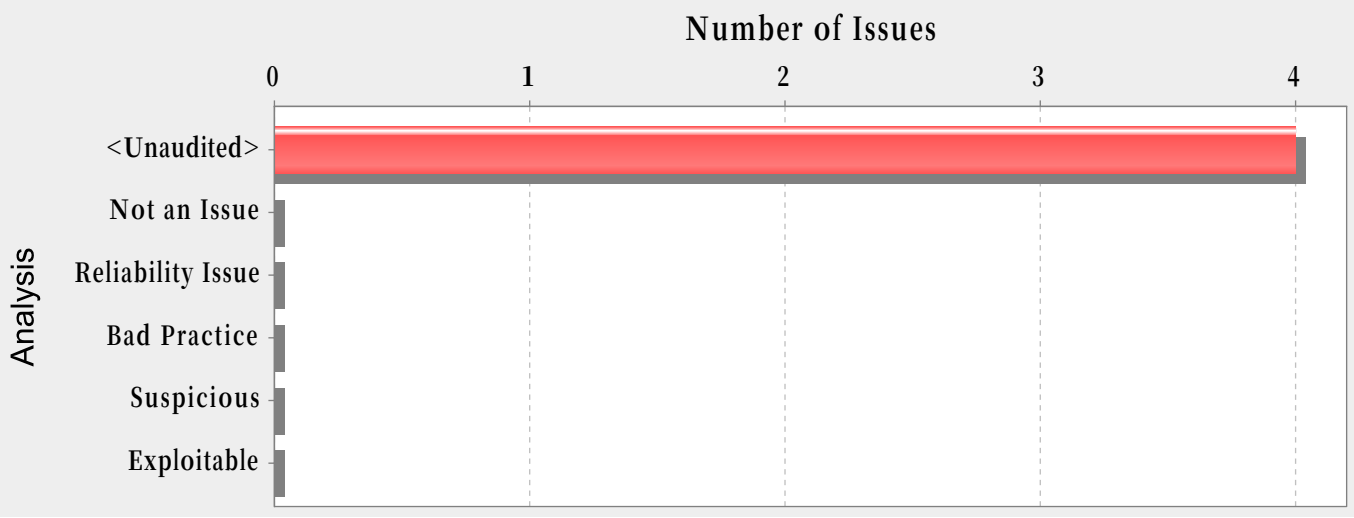
Recommendations:
考虑应用程序是否需要请求的权限来保证正常运行。如果不需要，则应将相应的权限从 AndroidManifest.xml 文件中删除。除了请求应用程序真正需要的权限之外，切忌因请求更多权限而导致对应用程序过度授权。这会导致在设备上安装的其他恶意应用程序利用这种过度授权的应用程序对用户体验及存储的数据造成负面影响。另外，设置过多的权限可能会适得其反，导致客户不愿意安装您的应用程序。

AndroidManifest.xml, line 14 (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:14 null()		

```
12      android:minSdkVersion="9"
13      android:targetSdkVersion="16" />
14    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
15    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
16    <uses-permission android:name="android.permission.INTERNET" />
```

Category: Android Bad Practices: Missing Component Permission (4 Issues)



Abstract:
在 AndroidManifest.xml 的第 35 行，程序未用导出标记或权限对此公共组件进行保护。

Explanation:
任何应用程序都能访问在显式定义中未明确分配访问权限的公共组件。

Android 应用程序中活动、接收者和服务组件的 exported 属性的默认值取决于是否存在 intent-filter。如果存在 intent-filter，则说明该组件供外部使用。因此应将 exported 属性设为 true。现在，Android 平台上的任何其他应用程序都可以访问该组件。

示例 1：以下是一个 Android 活动示例，该示例存在 intent-filter 且未设置明确访问权限。

```
<activity android:name=".AndroidActivity"/>
<intent-filter android:label="activityName"/>
<action android:name=".someFunAction"/>
</intent-filter>
...
</activity>
```

该活动可能被恶意应用程序利用。

Recommendations:
没有明确的访问权限的组件应该为异常。除非该组件确实需要被所有应用程序访问，否则开发人员应该通过在显式文件中明确定义访问权限来保护组件，以免其被恶意应用程序滥用。

示例 2：下面是Example 1 中使用明确分配的访问权限进行重写的活动组件声明。

```
<activity android:name=".AndroidActivity" android:permission="FriendsOnly"/>
<intent-filter android:label="activityName"/>
<action android:name=".someFunAction"/>
</intent-filter>
...
</activity>
```

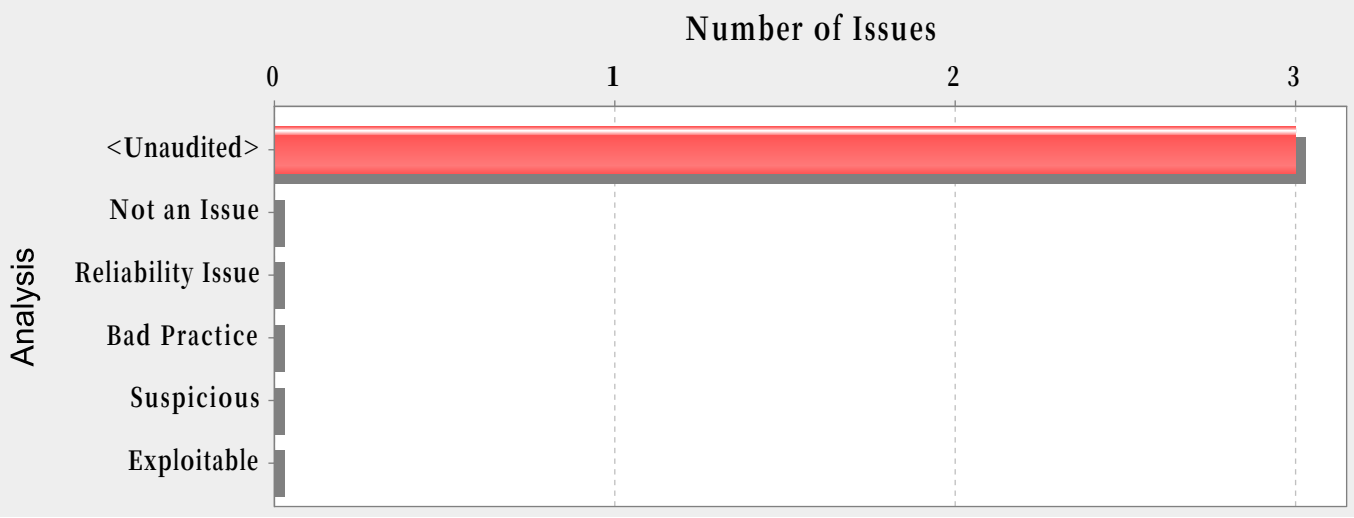
AndroidManifest.xml, line 35 (Android Bad Practices: Missing Component Permission)

Fortify Priority:	High	**Folder**	High
Kingdom:	Security Features		
Abstract:	在 AndroidManifest.xml 的第 35 行，程序未用导出标记或权限对此公共组件进行保护。		

Sink: AndroidManifest.xml:35 null()

```
33          </intent-filter>
34        </activity>
35          <activity android:name="com.retroarch.browser.retroactivity.RetroActivityPast" android:exported="true"
android:configChanges="mcc|mnc|locale|touchscreen|keyboard|keyboardHidden|navigation|orientation|screenLayout|uiMode|screenSize|smallestScreenSize|fontScale" android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
android:launchMode="singleInstance">
36            <meta-data android:name="android.app.lib_name" android:value="retroarch-activity" />
37            <meta-data android:name="android.app.func_name" android:value="ANativeActivity_onCreate" />
```

Category: Android Bad Practices: Missing Google Play Services Updated Security Provider (3 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）。

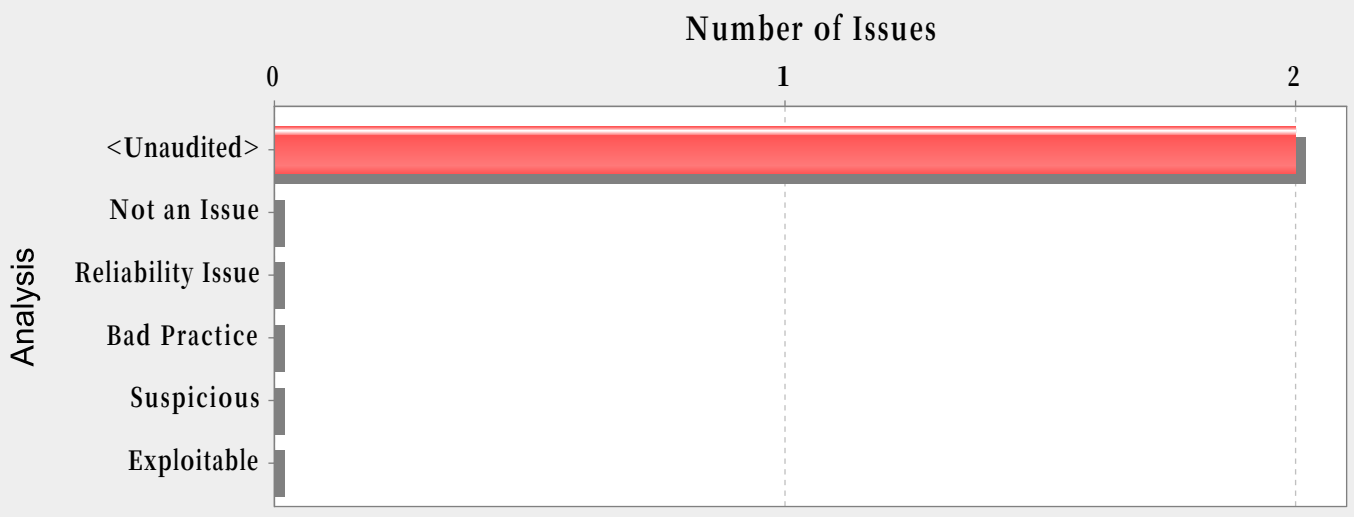
```
public class SyncAdapter extends AbstractThreadedSyncAdapter {

...

// 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 26 (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:26 null()		

```
24      android:hasCode="true"
25      android:isGame="true"
26      android:banner="@drawable/banner">
27      <activity android:name="com.retroarch.browser.mainmenu.MainMenuActivity" android:exported="true" android:launchMode="singleInstance">
28        <intent-filter>
```

Category: Dynamic Code Evaluation: Unsafe Deserialization (2 Issues)



Abstract:

在运行时对用户控制的对象流进行反序列化，会让攻击者有机会在服务器上执行任意代码、滥用应用程序逻辑和/或导致 Denial of Service。

Explanation:

Java 序列化会将对象图转换为字节流（包含对象本身和必要的元数据），以便通过字节流进行重构。开发人员可以创建自定义代码，以协助 Java 对象反序列化过程，在此期间，他们可以使用其他对象或代理替代反序列化对象。在对象重构过程中，并在对象返回至应用程序并转换为预期的类型之前，会执行自定义反序列化过程。到开发人员尝试强制执行预期的类型时，代码可能已被执行。

在必须存在于运行时类路径中且无法由攻击者注入的可序列化类中，会自定义反序列化例程，所以这些攻击的可利用性取决于应用程序环境中的可用类。令人遗憾的是，常用的第三方类，甚至 JDK 类都可以被滥用，导致 JVM 资源耗尽、部署恶意文件或运行任意代码。

Android 在底层使用 Java 序列化。从不受信任的 Intent 捆绑包获取任何额外项时可能会使攻击者发起攻击。为了在应用程序的上下文中执行任意代码，将要求应用程序在其中使用具有远程代码执行小工具的库。但是，攻击者将始终能够通过发送恶意生成的负载针对应用程序执行 Denial Of Service (DoS)。

Recommendations:

如果可能，在没有验证对象流的内容的情况下，请勿对不可信数据进行反序列化。为了验证要进行反序列化的类，应使用前瞻反序列化模式。

对象流首先将包含类描述元数据，然后包含其成员字段的序列化字节。Java 序列化过程可以让开发人员读取类描述，并确定是继续进行对象的反序列化还是中止对象的反序列化。为此，需要在应执行类验证和确认的位置，子类化 java.io.ObjectInputStream 并提供 resolveClass(ObjectStreamClass desc) 方法的自定义实现。

已有易于使用的前瞻模式实现方式，例如 Apache Commons IO (org.apache.commons.io.serialization.ValidatingObjectInputStream)。始终使用严格的白名单方法，以仅允许对预期类型进行反序列化。不建议使用黑名单方法，因为攻击者可以使用许多可用小工具绕过黑名单。此外，请谨记，尽管用于执行代码的某些类已公开，但是还可能存在其他未知或未公开的类，因此，白名单方法始终都是首选方法。应审计白名单中允许的任何类，以确保对其进行反序列化是安全的。

在库或框架中执行反序列化时（例如，使用 JMX、RMI、JMS、HTTP Invoker 时），上述建议并不适用，因为它超出了开发人员的控制范围。在这些情况下，您可能需要确保这些协议满足以下要求：

- 未公开披露。
- 使用身份验证。
- 使用完整性检查。
- 使用加密。

此外，每当应用程序通过 ObjectInputStream 执行反序列化时，Fortify Runtime（Fortify 运行时）都会提供要强制执行的安全控制，以此同时保护应用程序代码以及库和框架代码，防止遭到此类攻击。

Tips:

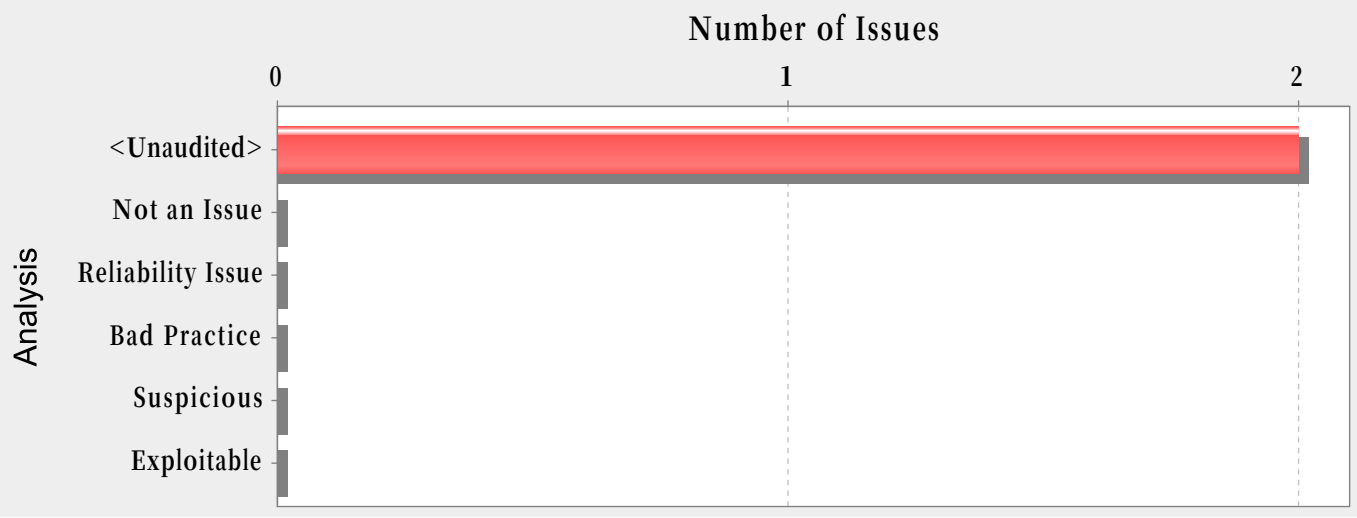
1. 由于 ObjectInputStream 实现中的现有缺陷，以及很难对可能用于执行拒绝服务 (DoS) 攻击的基本类列出黑名单，因此即使实现前瞻 ObjectInputStream 也依然会报告此问题，但其严重性会降为 Medium。

CoreSideloadActivity.java, line 31 (Dynamic Code Evaluation: Unsafe Deserialization)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Input Validation and Representation		
Abstract:	在运行时对用户控制的对象流进行反序列化，会让攻击者有机会在服务器上执行任意代码、滥用应用程序逻辑和/或导致 Denial of Service。		

Sink:	CoreSideloadActivity.java:31 Android Exported Activity()
29	* Usage : see Phoenix Gradle Build README.md
30	*/
31	public class CoreSideloadActivity extends Activity
32	{
33	private static final String EXTRA_CORE = "LIBRETRO";

Category: Key Management: Hardcoded Encryption Key (2 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。

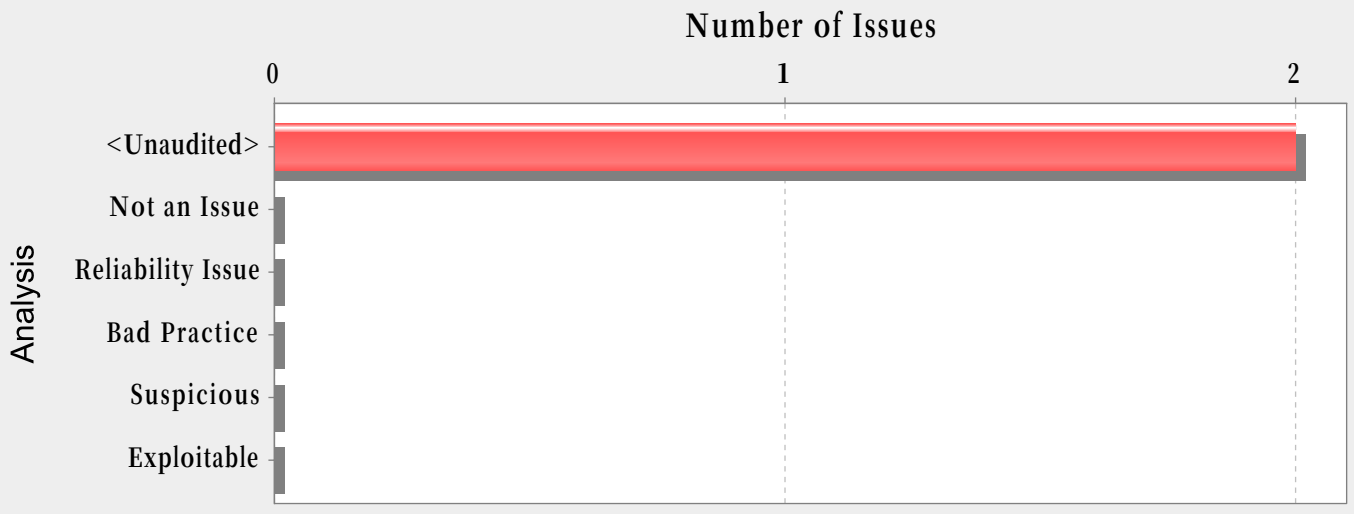
github-contributors.py, line 100 (Key Management: Hardcoded Encryption Key)

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: github-contributors.py:100 Operation()

```
98    login = node['login']
99
100   if 'cursor' in key and key['cursor'] and len(key['cursor']) > 0:
101     cursor = key['cursor']
```

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



Abstract:

程序在 AndroidManifest.xml 的第 14 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。

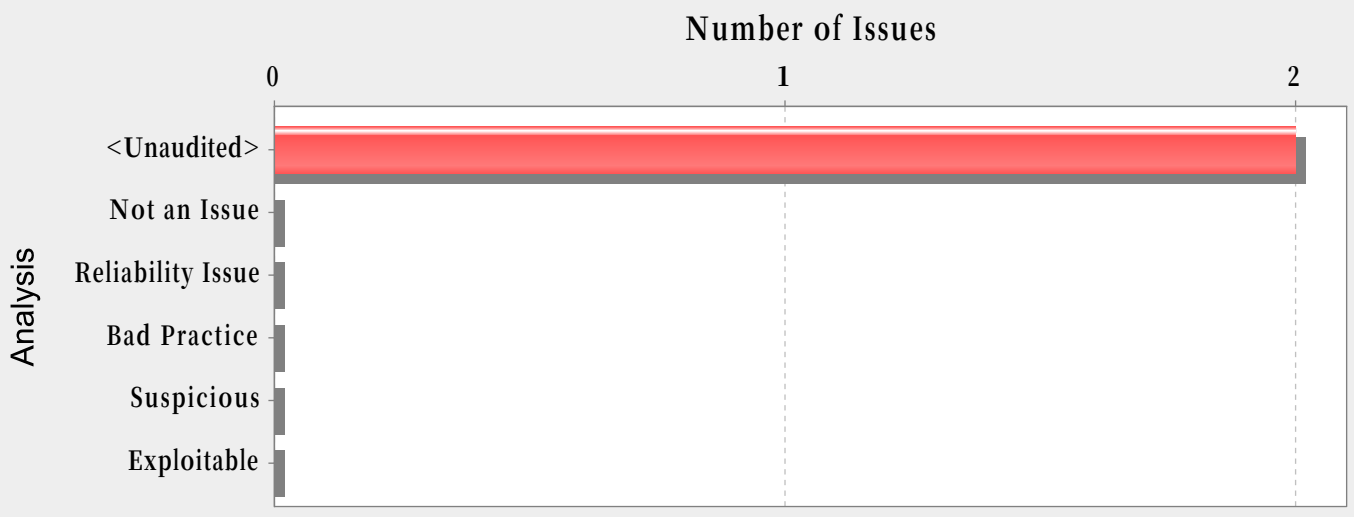
```
String FILENAME = "hello_file";
String string = "hello world!";

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

```
fos.close();
```

AndroidManifest.xml, line 14 (Privilege Management: Android Data Storage)			
Fortify Priority:	High	**Folder**	High
Kingdom:	Security Features		
Abstract:	程序在 AndroidManifest.xml 的第 14 行请求将数据写入 Android 外部存储的权限。		
Sink:	AndroidManifest.xml:14 null()		

```
12      android:minSdkVersion="9"
13      android:targetSdkVersion="16" />
14    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
15    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
16    <uses-permission android:name="android.permission.INTERNET" />
```

Category: Privilege Management: Android Network (2 Issues)



Abstract:
程序在 AndroidManifest.xml 的第 16 行请求建立网络连接的权限。

Explanation:
授予此权限会使该软件能够打开网络套接字。这个权限会授予程序
对设备的控制权，从而对用户造成负面影响。因为此类型的权限会带来潜在风险，系统不会自动将此权限授予请求者
。

示例 1：以下 AndroidManifest.xml 中的 <uses-permission .../> 元素包含一个网络权限属性。

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

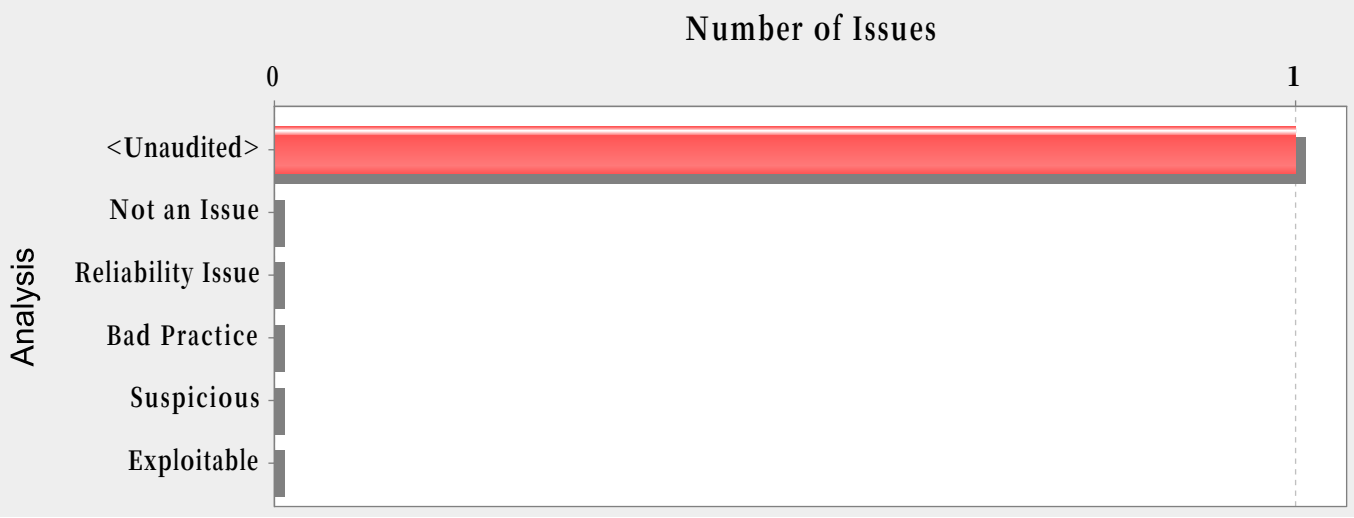
Recommendations:
请求此权限时请慎重考虑。如果程序不需要此权限，
用户可能会拒绝安装。

AndroidManifest.xml, line 16 (Privilege Management: Android Network)

Fortify Priority:	High	**Folder**	High
Kingdom:	Security Features		
Abstract:	程序在 AndroidManifest.xml 的第 16 行请求建立网络连接的权限。		
Sink:	AndroidManifest.xml:16 null()		

```
14    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
15    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
16    <uses-permission android:name="android.permission.INTERNET" />
17    <uses-permission android:name="android.permission.VIBRATE" />
```

Category: Android Bad Practices: Provider Permission Defined (1 Issues)



Abstract:
在 AndroidManifest.xml 的第 48 行中，程序声明内容提供者具有读取和写入 permission。

Explanation:
声明为具有读取和写入 permission 的内容提供者可供请求对此提供者进行读取或写入访问的实体访问。但在许多情况下，就像文件系统中的文件一样，这些需要读取访问由内容提供者存储的数据的实体不应该被允许修改数据。设置 permission 属性不允许区分数据用户与影响数据完整性的交互。

示例 1：下面是一个使用读取和写入permission声明的内容提供者示例。

```
<provider android:name=".ContentProvider" android:permission="content.permission.READ_AND_WRITE_CONTENT"/>
```

Recommendations:
与 UNIX 文件权限中的情况类似，Android 权限框架允许为内容提供者定义单独的读取和写入权限。即使读取和写入权限之间的区别并不明显，开发者也应该始终定义单独的读取和写入权限。

示例 2：下面是Example 1 中使用单独的读取和写入权限声明的内容提供者示例。

```
<provider android:name=".ContentProvider"
android:readPermission="content.permission.READ_CONTENT"
android:writePermission="content.permission.WRITE_CONTENT">
</provider>
```

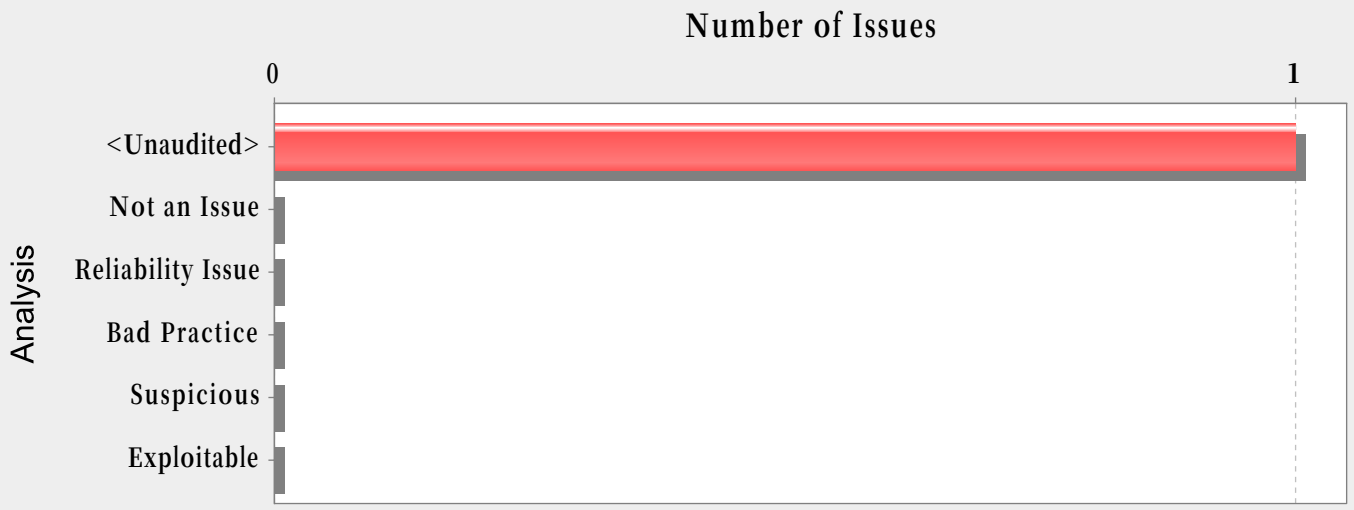
AndroidManifest.xml, line 48 (Android Bad Practices: Provider Permission Defined)

Fortify Priority:	High	**Folder**	High
Kingdom:	Security Features		
Abstract:	在 AndroidManifest.xml 的第 48 行中，程序声明内容提供者具有读取和写入 permission。		

Sink: AndroidManifest.xml:48 null()

```
46      android:enabled="@bool/document_provider_enabled"
47      android:exported="true"
48      android:permission="android.permission.MANAGE_DOCUMENTS">
49      <intent-filter>
50       <action android:name="android.content.action.DOCUMENTS_PROVIDER" />
```

Category: Cross-Site Scripting: DOM (1 Issues)



Abstract:

libretro.js 中的方法 lambda() 向第 321 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 盗取发生在如下情况：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]：

在有关块级别元素的内容中（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。

libretro.js, line 321 (Cross-Site Scripting: DOM)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Input Validation and Representation		
Abstract:	libretro.js 中的方法 lambda() 向第 321 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。		

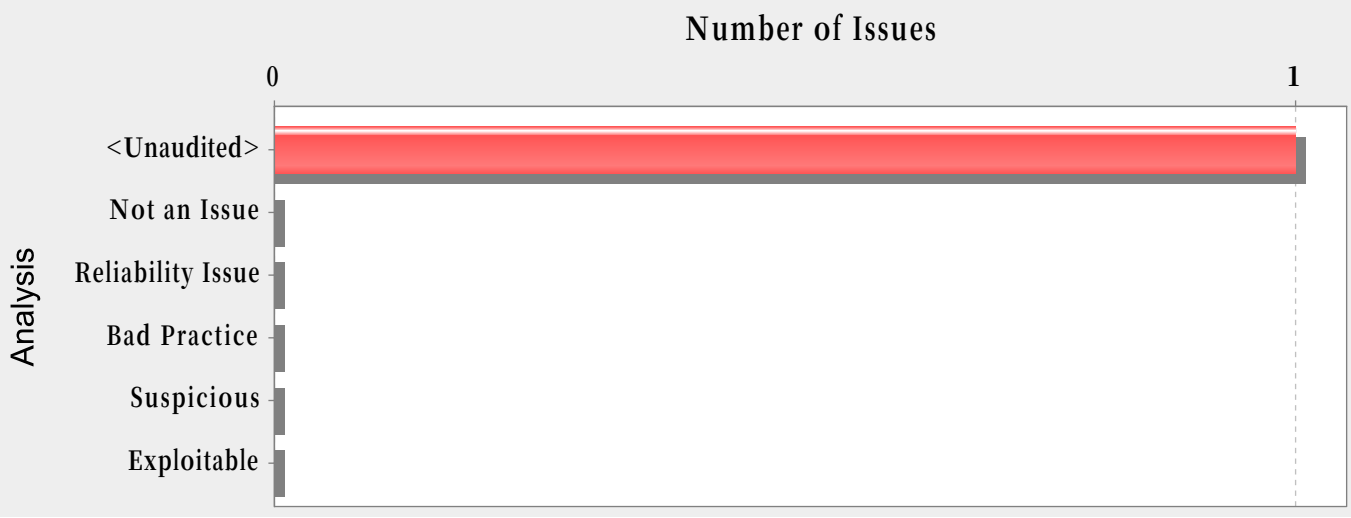
Source: libretro.js:316 ~JS_Generic.getItem()

```
314
315   // Find which core to load.
316   var core = localStorage.getItem("core", core);
317   if (!core) {
318     core = 'gambatte';
```

Sink: libretro.js:321 environment~object.$()

```
319   }
320   // Make the core the selected core in the UI.
321   var coreTitle = $('#core-selector a[data-core="' + core + '"]').addClass('active').text();
322   $('#dropdownMenu1').text(coreTitle);
```

Category: Dynamic Code Evaluation: Script Injection (1 Issues)



Abstract:

libretro.js文件将未经验证的用户输入解析为第 325 行的源代码。在运行时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，若不经过适当的验证，用户指定的操作可能并不是程序员最初所期望的。

示例：在这一典型的代码注入示例中，应用程序实施的基本计算器允许用户指定要执行的命令。

```
...
userOp = form.operation.value;
calcResult = eval(userOp);
...
```

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，calcResult 变量被赋予的值将为 22。然而，如果攻击者指定的语言操作既有可能是有效的，又有可能是恶意的，那么，只有在对主进程具有完全权限的情况下才能执行这些操作。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。对于 JavaScript，攻击者还可以利用这种漏洞进行 cross-site scripting 攻击。

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，应用程序绝不应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。

libretro.js, line 325 (Dynamic Code Evaluation: Script Injection)

Fortify Priority:	Critical	**Folder**	Critical
Kingdom:	Input Validation and Representation		
Abstract:	libretro.js文件将未经验证的用户输入解析为第 325 行的源代码。在运行时解析用户控制的指令，会让攻击者有机会执行恶意代码。		

Source: libretro.js:316 ~JS_Generic.getItem()

```
314
315   // Find which core to load.
316   var core = localStorage.getItem("core", core);
317   if (!core) {
318     core = 'gambatte';
```

Sink: libretro.js:325 ~JS_Generic.getScript()

```
323
324   // Load the Core's related JavaScript.
325   $.getScript(core + '_libretro.js', function ()
326   {
327     $('#icnRun').removeClass('fa-spinner').removeClass('fa-spin');
```

Issue Count by Category

Issues by Category	
Privilege Management: Unnecessary Permission	6
Android Bad Practices: Missing Component Permission	4
Android Bad Practices: Missing Google Play Services Updated Security Provider	3
Dynamic Code Evaluation: Unsafe Deserialization	2
Key Management: Hardcoded Encryption Key	2
Privilege Management: Android Data Storage	2
Privilege Management: Android Network	2
Android Bad Practices: Provider Permission Defined	1
Cross-Site Scripting: DOM	1
Dynamic Code Evaluation: Script Injection	1

Issue Breakdown by Analysis

Issues by Analysis