ctf安全_受伤的Android CTF撰写

翻译

weixin_26722031 于 2020-07-30 21:26:24 发布 406 收藏 1
文章标签： python java
原文链接：https://medium.com/bugbountywriteup/injuredandroid-ctf-writeup-41dd40165cfa
版权
ctf安全

In this article, I will be walking through the InjuredAndroid CTF. This is a vulnerable Android application with CTF examples based on bug bounty findings, exploitation concepts, and pure creativity. I have left a link to the creators Github and the GitHub I used to download the APK in the references below for anyone interested in trying out the CTF themselves.

在本文中，我将逐步介绍InjuredAndroid CTF。 这是一个易受攻击的Android应用程序，带有基于漏洞赏金发现，利用概念和纯粹创造力的CTF示例。 我留下了指向创建者Github和我用来下载APK的GitHub的链接，这些链接在下面的参考资料中供有兴趣尝试CTF本身的任何人使用。

免责声明 (Disclaimer)

This writeup will obviously contain spoilers and I encourage readers to attempt this CTF before looking at this article. You will learn more by attempting it yourself first and will gain more satisfaction from solving the challenges yourself.

该文章显然将包含破坏者，我鼓励读者在阅读本文之前尝试使用CTF。 您将可以先尝试一下，从中学到更多，并从解决挑战中获得更多的满足感。

The author of this CTF has also mentioned that:

该CTF的作者还提到：

> Looking at the source code of the applications in the InjuredAndroid directory, InjuredAndroid FlagWalkthroughs.md file, or binary source code in the Binaries directory will spoil some if not all of the ctf challenges.
>
> 在InjuredAndroid目录中查看应用程序的源代码，InjuredAndroid FlagWalkthroughs.md文件或Binaries目录中的二进制源代码将破坏一些(甚至不是全部)ctf挑战。

I must also point out that challenge seven and eight for the release of the APK I used do not function properly and do not have flags. I discovered this after starting the writeup and decided to continue on anyways. With all that said, it’s time to move onto to the writeup!

我还必须指出，针对我使用的APK的发布，挑战7和8不能正常运行且没有标志。 我在开始撰写文章后就发现了这一点，并决定继续进行下去。 综上所述，现在该进行写作了！

最初设定 (Initial Setup)

For this CTF, I will be using a Kali Linux virtual machine as my host device and a Samsung Galaxy S8 emulator created with Genymotion with the following specs:

对于此CTF，我将使用Kali Linux虚拟机作为主机设备，并使用由Genymotion创建的Samsung Galaxy S8模拟器，其规格如下：

Image for post

To begin the CTF, i connected to my emulator using Android Debug Bridge (ADB) and installed the "injuredandroid.apk" file.

为了开始CTF，我使用Android调试桥(ADB)连接到仿真器并安装了" injuredandroid.apk"文件。

Image for post

Looking at my emulator, I can see that the application has been installed successfully.

查看我的模拟器，可以看到该应用程序已成功安装。

Image for post

The CTF author also highly recommends decompiling the "injuredandroid.apk". To accomplish this, I will be using a tool called Mobile Security Framework (MobSF). MobSF automates the process of decompiling the APK, reading the manifest file, identifying issues in the source code and in the Manifest file, extracting the certificate of the application etc. and saves me from having to do this manually. The image below shows the application has been successfully decompiled by MobSF.

CTF作者还强烈建议反编译" injuredandroid.apk"。 为此，我将使用一个称为"移动安全框架"(MobSF)的工具。MobSF可以自动执行以下过程：反编译APK，读取清单文件，识别源代码和清单文件中的问题，提取应用程序的证书等，从而使我不必手动执行此操作。 下图显示了该应用程序已被MobSF成功反编译。

Image for post

With the initial setup out of the way, I can now move on to the challenges.

通过最初的设置，我现在可以继续挑战。

XSS测试 (XSS Test)

Opening the application, I am greeted with the following main activity.

打开该应用程序，我将看到以下主要活动。

Image for post

There appears to be eight flags in total. According to the author:

似乎总共有八个标志。 根据作者：

> XSSTEST is just for fun and to raise awareness on how WebViews can be made vulnerable to XSS.
>
> XSSTEST只是为了娱乐，并提高了人们对如何使WebView容易受到XSS攻击的认识。

Looking at the XSSTEST activity, I am presented with a simple input field where I can submit text.

在XSSTEST活动中，我看到一个简单的输入字段，可以在其中提交文本。

Image for post

I can enter some simple JavaScript that will create and alert box to demonstrate if the vulnerability exists.

我可以输入一些简单JavaScript，它们将创建并在警告框中显示该漏洞是否存在。

```
<script>alert('XSS!!')</script>
```

Image for post

Entering this input causes an alert box to be generated when the activity used to display our input is loaded.

输入此输入将导致在加载用于显示输入的活动时生成警报框。

Image for post

The challenge recommends looking at the “DisplayPostXSS” activity to determine what makes this activity vulnerable. The source code for the “DisplayPostXSS” activity can be seen in the image below.

挑战建议查看“ DisplayPostXSS”活动，以确定导致该活动易受攻击的原因。 下图显示了“ DisplayPostXSS”活动的源代码。

Image for post

Examining the source code, I can see that a new WebView object is created which allows developers to display web content as part of their activity layout. This activity is vulnerable to XSS because the developer has enabled JavaScript execution as seen highlighted in red above. This is a nice, simple example of how developers can leave WebViews vulnerable to XSS.

通过检查源代码，我可以看到创建了一个新的WebView对象，该对象允许开发人员在其活动布局中显示Web内容。 此活动容易受到XSS的攻击，因为开发人员已启用JavaScript执行，如上面红色突出显示。 这是一个很好的简单示例，说明开发人员如何使WebView易于受到XSS的攻击。

标记一：登录 (Flag One: Login)

In the first challenge with a flag, I am presented with an activity which requires me to enter the flag and login.

在带有标志的第一个挑战中，向我展示了一项活动，要求我输入标志并登录。

Image for post

For this challenge, I started by examining the “FlagOneLoginActivity” source code. Examining the source code revealed that the activity was checking if my input was equal to the flag, which was hardcoded in plaintext.

为了应对这一挑战，我首先检查了“ FlagOneLoginActivity”源代码。 检查源代码后发现，该活动正在检查我的输入是否等于以纯文本硬编码的标志。

Image for post

Entering this flag presents me with a new activity congratulating me on finding the flag. Nice and easy □.

输入此标志将为我带来一个新的活动，祝贺我找到该标志。 很好，很容易。

标记二：导出的活动 (Flag Two: Exported Activity)

Moving on to challenge two, I am presented with an activity which explains that I can bypass the main activity and call other activities that are exported.

继续挑战两个，我看到一个活动，该活动解释说我可以绕过主要活动并调用其他导出的活动。

Image for post

I started this challenge by looking at the AndroidManifest file for the application to see what activities were exported. This file acts as a blueprint for the application.

通过查看应用程序的AndroidManifest文件以查看导出了哪些活动，我开始了这一挑战。 该文件充当应用程序的蓝图。

Image for post

There were three activities exported according to the AndroidManifest file. These activities included the “MainActivity” which is exported by default due to having am intent filter set, an activity called “b25lActivity” and an activity called “TestBroadcastReciever”. The “b25lActivity” stands out as an unusual name for an activity. Using ADB as seen in the image below, I can start this exported activity.

根据AndroidManifest文件，导出了三个活动。 这些活动包括“ MainActivity”(由于设置了意图过滤器而默认导出)，一个名为“ b25lActivity”的活动和一个名为“ TestBroadcastReciever”的活动。 “ b25lActivity”是一项活动的不寻常名称。 如下图所示，使用ADB，我可以启动此导出的活动。

Image for post

This activity presents me with the flag for this challenge.

这项活动向我展示了这一挑战的旗帜。

Image for post

标志三：资源 (Flag Three: Resources)

In this challenge, I am asked to input and submit the flag.

在此挑战中，要求我输入并提交标志。

Image for post

Looking at the source code for the “FlagThreeActivity”, I can see that the developer is again using the “equals()” method to compare my input with the flag. However, this time my input is being compared to a string that is retrieved from a resource file.

查看“ FlagThreeActivity”的源代码，我可以看到开发人员再次使用“ equals()”方法将输入与标志进行比较。 但是，这次将我的输入与从资源文件中检索的字符串进行比较。

Image for post

A resource can be referenced in Java by typing “R.string.<string_name>”. In this instance, the string name being referenced is “cmVzb3VyY2VzX3lv”. Since I used MobSF to decompile the application, I can examine the strings recovered from the resource files for the application. Searching for the string name “cmVzb3VyY2VzX3lv” reveals the flag for this challenge.

可以通过键入“ R.string。<string_name>”在Java中引用资源。 在这种情况下，所引用的字符串名称为“ cmVzb3VyY2VzX3lv”。 由于我使用MobSF来反编译应用程序，因此我可以检查从应用程序的资源文件中恢复的字符串。 搜索字符串名称“ cmVzb3VyY2VzX3lv”将显示该挑战的标志。

Image for post

It is also possible to retrieve this flag by decompiling the application using APKTool and examining the “strings.xml” file.

也可以通过使用APKTool反编译应用程序并检查“ strings.xml”文件来检索此标志。

标志四：登录2 (Flag Four: Login 2)

This challenge builds on top of challenge one. As seen earlier, the challenge is asking me to login by submitting the flag.

此挑战基于挑战一。 如前所述，挑战是要求我通过提交标志来登录。

Image for post

Looking at the source code for this activity, I can see that it is retrieving data from a different class called “Decoder” and then comparing it to my input.

查看此活动的源代码，我可以看到它正在从另一个名为“解码器”的类中检索数据，然后将其与我的输入进行比较。

Image for post

I can examine this “Decoder” class java file which reveals a string of text that is base64 encoded.

我可以检查这个“ Decoder”类java文件，该文件显示了经过base64编码的文本字符串。

Image for post

Decoding this base64 encoded string using an online tool such as CyberChef gives me the flag for this challenge.

使用诸如CyberChef之类的在线工具对该base64编码的字符串进行解码为我提供了应对这一挑战的标志。

Image for post

标志五：导出的广播接收器 (Flag Five: Exported Broadcast Receiver)

Looking at the activity for this challenge, there is no text or challenge description provided. Instead, a message is provided each time I click the link to the challenge. After visiting the challenge’s activity several times, the flag is broadcasted to us.

在针对该挑战的活动方面，没有提供文本或挑战描述。 相反，每当我单击挑战的链接时，都会提供一条消息。 多次访问挑战活动后，该标志会广播给我们。

Image for post

Obviously not satisfied with just getting the flag, I decided to look at the source code for this activity in order to understand how the challenge worked. As the name of this challenge suggests, the application has exported a broadcast receiver. Broadcast receivers are designed to listen to system wide events called broadcasts (e.g. network activity, application updates, etc.) and then trigger something if the broadcast message matches the current parameters inside the Broadcast Receiver. The source code for this challenge’s activity creates a new intent and then broadcasts this intent.

显然对获得标志并不满意，我决定查看此活动的源代码，以了解挑战的工作原理。 顾名思义，该应用程序已导出广播接收器。 广播接收器旨在侦听称为广播的系统范围事件(例如，网络活动，应用程序更新等)，然后在广播消息与广播接收器内部的当前参数匹配时触发某些事件。 该挑战活动的源代码创建一个新的意图，然后广播该意图。

Image for post

A class called “FlagFiveReceiver” handles what the BroadcastReceiver does when receiving an Intent broadcast.

名为“ FlagFiveReceiver”的类负责处理Intent广播时BroadcastReceiver的工作。

Image for post

The class’s source code starts by declaring a string variable which is the action that triggers the broadcast receiver. A variable called “wtf” is also declared which is incremented each time the activity is interacted with. An “if else” block statement is used to print the flag after the “wtf” variable has been incremented to the value of 2. An interesting class with a random name can be seen which has a function called “decrypt()”. I can assume that the text being decrypted is the flag and I decided to take a closer look at this class.

该类的源代码首先声明一个字符串变量，该变量是触发广播接收器的动作。 还声明了一个名为“ wtf”的变量，该变量在每次与活动进行交互时都会增加。 在“ wtf”变量增加到值2之后，将使用“ if else”块语句来打印标志。可以看到带有随机名称的有趣类，该类具有名为“ decrypt()”的函数。 我可以假设正在解密的文本是标志，因此我决定仔细看一下此类。

Image for post

Looking at the source code, I can see that the flag was encrypted using the deprecated DES encryption algorithm. The “decrypt()” function takes in a string parameter and checks if it is base64 encoded and then proceeds to decrypt the flag. In order to decrypt the flag, the key used for initially encrypting the flag is required and is seen to be retrieved from a class called “Hide” using the function “getKey()”. The source code for this class can be seen below.

查看源代码，我可以看到该标志是使用不赞成使用的DES加密算法加密的。“ decrypt()”函数接收一个字符串参数，并检查其是否为base64编码，然后继续对该标志进行解密。 为了解密标志，需要使用用于初始加密标志的密钥，并且可以使用函数“ getKey()”从称为“隐藏”的类中检索该密钥。 此类的源代码如下所示。

Image for post

Looking at the source code in the image above, I can see the “getKey()” function returns a variable called “encKey” which contains the base64 decoded key. Using an online tool like CyberChef, I can retrieve the plaintext key used to encrypt the flag.

查看上图中的源代码，我可以看到“ getKey()”函数返回了一个名为“ encKey”的变量，其中包含base64解码密钥。 使用像CyberChef这样的在线工具，我可以检索用于加密标志的纯文本密钥。

Image for post

标志六：登录3 (Flag Six: Login 3)

As seen with the previous challenges of this type, I am asked to provide the flag in order to login.

从以前的此类挑战中可以看出，要求我提供标志以便登录。

Image for post

Looking at the source code of the activity, I can see that it is using the “decrypt()” method from the class called “VGV4dEVuY3J5cHRpb25Ud28” that was used in the last challenge, to decrypt a string and compare it to my input.

查看活动的源代码，我可以看到它正在使用上一个挑战中使用的名为“ VGV4dEVuY3J5cHRpb25Ud28”的类中的“ decrypt()”方法来解密字符串并将其与我的输入进行比较。

Image for post

As seen in the image above, the flag is hardcoded but needs to be decrypted. One possible solution to this challenge is to use a tool called **Frida**. Frida is a free and open source Dynamic Instrumentation Toolkit. This tool allows you to inject your own code and to programmatically and interactively inspect and change running processes. Frida can be used to hook methods and then inject them with your own code.

如上图所示，该标志是硬编码的，但需要解密。 解决此难题的一种可能的解决方案是使用一个名为**Frida**的工具。 Frida是免费的开源动态仪表工具包。 该工具使您可以注入自己的代码，并以编程方式和交互方式检查和更改正在运行的流程。 Frida可用于挂钩方法，然后使用您自己的代码注入它们。

> In programming, the term hooking covers a range of techniques used to alter or augment the behavior of an OS, app or other software components by intercepting function calls or messages or events passed between software components. Code that handles such intercepted function calls, events or messages is called a hook.
>
> 在编程中，术语“钩子”涵盖了一系列技术，这些技术通过截获软件组件之间传递的函数调用或消息或事件来更改或增强OS，应用程序或其他软件组件的行为。 处理此类拦截的函数调用，事件或消息的代码称为钩子。

Frida allows me to **insert JavaScript code** (hook) inside functions of a running application. I can also use **python** to **call** the hooks and even to **interact** with the **hooks**. I have left a link to an excellent guide on how to setup and use Frida with examples in the references below. This guide provided a useful example (see references) of how I could hook and call a function with my own parameter. This can be used to hook the “decrypt()” function and then call it with my own parameter (i.e. the encrypted flag) which will then be decrypted. The JavaScript code (hook) is provided below:

Frida允许我在正在运行的应用程序的函数内**插入JavaScript代码** (钩子)。 我还可以使用**python 调用**钩子，甚至与**钩子**进行**交互** 。 我留下了一个很好的指南链接，该指南提供了以下示例中如何设置和使用Frida的示例。 本指南提供了一个有用的示例(请参阅参考资料)，说明了如何使用自己的参数进行钩子和调用函数。 这可以用来钩住“ decrypt()”函数，然后用我自己的参数(即，加密标志)调用它，然后将其解密。 以下提供了JavaScript代码(钩子)：

Image for post

The python script used to load this hook is provided below.

下面提供了用于加载此钩子的python脚本。

Image for post

I make sure that the Frida server is running on my emulator and then execute my python script to inject the JavaScript code.

我确保Frida服务器在模拟器上运行，然后执行python脚本以注入JavaScript代码。

Image for post

My script was loaded successfully and now I need to submit a flag in order to call the decrypt method.

我的脚本已成功加载，现在我需要提交一个标志才能调用解密方法。

Image for post

Submitting the word “test” causes the “decrypt()” function to be called, which I then overwrite with my own input (i.e. encrypted flag). This gives me the flag.

提交单词“ test”会导致调用“ decrypt()”函数，然后我用自己的输入(即，加密标志)将其覆盖。 这给了我旗子。

标志七：**SQLite (Flag Seven: SQLite)**

Moving on to challenge seven, I found that I was initially unable to access the activity for the challenge. Looking at the source code for the “MainActivity”, I discovered that challenges one to six must be completed first before I can attempt challenge seven.

继续挑战七，我发现我最初无法参加挑战的活动。 通过查看“ MainActivity”的源代码，我发现必须先完成第一个至第六个挑战，然后才能尝试挑战七。

Image for post

An “if” statement is used to check the boolean value of six variables located in the “FlagsOverview” class. If all six of these boolean values are true, then the activity for challenge seven can be started. I realized that each time I closed the application, these boolean variables would be reset and I would have to enter the flags again before I could do challenge seven.

“ if”语句用于检查位于“ FlagsOverview”类中的六个变量的布尔值。 如果所有六个布尔值都为true，则可以开始挑战7的活动。 我意识到，每次关闭应用程序时，这些布尔变量都将被重置，因此我必须再次输入标志才能挑战七个。

Image for post

Since I am lazy and I don’t want to enter all six flags each time I closed the application, I created a Frida script that sets all flags to true □.

由于我很懒，并且每次关闭应用程序时都不想输入所有六个标志，因此我创建了一个Frida脚本，将所有标志设置为true。

Image for post

Using the same python loader script as before in challenge six, I injected the application with my JavaScript code and changed the values of the variables.

使用与挑战六中相同的python loader脚本，我向应用程序注入了JavaScript代码并更改了变量的值。

Image for post

The output above shows that the values of the variables have been changed. Looking at the flags overview activity, I can also see the color of the buttons have changed to green. I can now access the activity for challenge seven.

上面的输出显示变量的值已更改。 查看标志概述活动，我还可以看到按钮的颜色已变为绿色。 我现在可以访问挑战七的活动。

Image for post

The activity for challenge seven does not contain any challenge description or text. Looking at the source code for the activity, I can see that a class called “DatabaseSchema” is used to create an SQLite database file.

挑战七的活动不包含任何挑战说明或文本。 查看活动的源代码，我可以看到一个名为“ DatabaseSchema”的类用于创建SQLite数据库文件。

Image for post

Further down in the source code, the database file is retrieved and some values are entered into the database.

在源代码的后面，检索数据库文件，并将一些值输入数据库。

Image for post

It is also important to note that the database file is deleted when the challenge seven activity is destroyed.

同样重要的是要注意，当挑战七活动被破坏时，数据库文件将被删除。

Image for post

Looking at the “DatabaseSchema” class source code, I can see that the name of the SQLite database file is called “Thisisatest.db”.

查看“ DatabaseSchema”类源代码，可以看到SQLite数据库文件的名称称为“ Thisisatest.db”。

Image for post

The file can be found in the applications data directory (N.B. Make sure the challenge seven activity is open or the file won’t be created).

可以在应用程序数据目录中找到该文件(注意，请确保打开挑战七活动，否则将不会创建该文件)。

Image for post

Using ADB, I can pull the file from it’s location on the emulator and view it using a tool, such as sqlitebrowser.

使用ADB，我可以从模拟器上的文件位置提取文件，并使用诸如sqlitebrowser之类的工具进行查看。

Image for post

It appears the database has two entries which include a hash and what appears to be a URL of some kind. Using an online tool for cracking MD5 hashes, I can recover the hashed text as seen below.

看来数据库有两个条目，其中包括散列和看起来像某种URL的条目。 使用在线工具破解MD5哈希，我可以恢复散列文本，如下所示。

Image for post

For the URL, I recognized that it had been encrypted using ROT13, since only the letters had been rotated and not the special characters. Using CyberChef, I was able to recover the URL.

对于URL，我意识到它已经使用ROT13进行了加密，因为只旋转了字母，而不旋转了特殊字符。 使用Cyber Chef，我能够恢复URL。

Image for post

Visiting this URL gave me a “note not found error”.

访问此URL给我一个“找不到笔记的错误”。

Image for post

This is unfortunately where the challenge ends □. Referring to the walkthrough notes available on the GitHub repository revealed that the URL has now changed. There is also no where to submit the URL and hashed text. For the purposes of this challenge, I’ll just have to settle for the cracked hash (i.e. “hunter2”) being the flag.

不幸的是，这是挑战结束的地方。 查阅GitHub存储库上的演练笔记后，发现URL现在已更改。 也没有提交URL和哈希文本的位置。 出于此挑战的目的，我只需要解决破裂的哈希(即“ hunter2”)作为标志的问题。

标志八：AWS (Flag Eight: AWS)

The challenge description explains that there are AWS credentials hidden in the application somewhere but unfortunately none were found. After sometime looking for these credentials, I referred to the walkthrough notes on the GitHub repository. I discovered that the APK release I am using does not possess these credentials and so I cannot solve this challenge.

挑战说明解释说，应用程序中某处隐藏了AWS凭证，但不幸的是没有找到。 在寻找了这些凭证之后，我参考了GitHub存储库上的演练说明。 我发现我使用的APK版本不具备这些凭据，因此我无法解决此挑战。

结束语 (Closing Remarks)

This was a fun Android themed CTF that I really enjoyed completing. Despite the last two challenges not being solvable, I believe this CTF was able to showcase some simple security issues that can be present in Android applications and can be a fun way for beginners to get started in Android application security. Big thanks to the author of the CTF and thanks for reading to the end □!

这是我非常喜欢完成的有趣的Android主题CTF。 尽管无法解决最后两个挑战，但我相信CTF能够展示一些简单的安全性问题，这些问题可能会出现在Android应用程序中，并且对于初学者来说，这是一种有趣的方式来开始使用Android应用程序安全性。 非常感谢CTF的作者，并感谢您阅读全文the！

> 翻译自: https://medium.com/bugbountywriteup/injuredandroid-ctf-writeup-41dd40165cfa

ctf安全