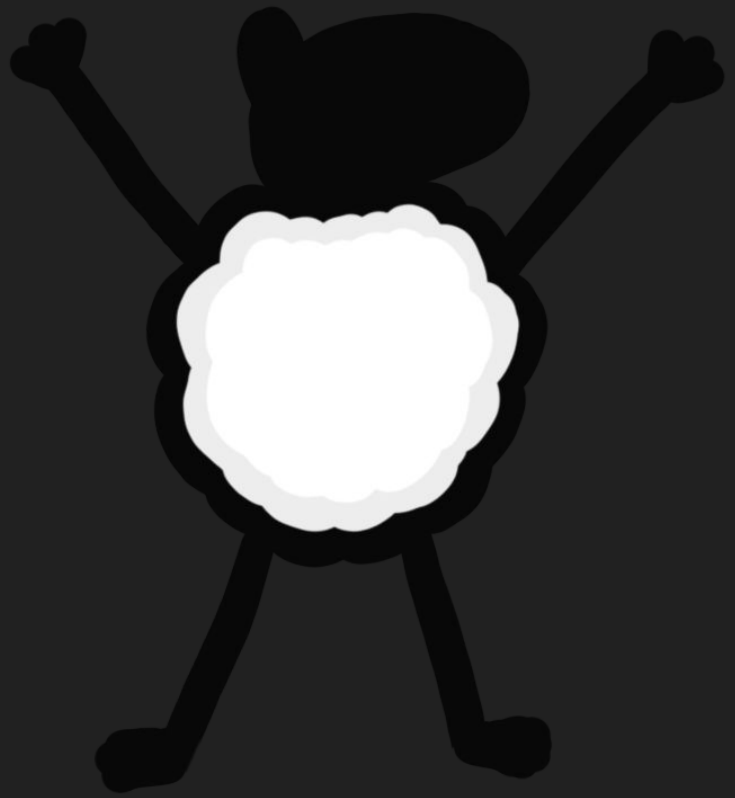
Bug Bounties With Bash

TomNomNom

Me

- Trainer
- @TomNomNom online
- Question lover
- Mediocre bug hunter



Obligatory Disclaimer

- The Computer Misuse Act is serious business
- Don’t do things unless you have explicit permission
- I am *not* your supervisor

Bug Bounties

- Companies have bugs
- You find the bugs
- You tell the companies
- The companies give you money
  - ...or ‘swag’
- I like bug bounties :)

Bash

- Bash is a shell
- ...it’s a botany metaphor!
- A shell wraps the kernel so you can launch processes
- There are other shells…
  - zsh
  - fish
  - ksh
  - explorer.exe…
- I like bash :)

Bug Bounties *and* Bash?

- Why not?
- There are many purpose-made security tools that *nearly* do what you want
- Sometimes you just have to make tools

Y u no gui?

- GUIs are nice
- They provide better discoverability
- But if they don’t support your use case you’re SOOL (:

Bash Basics

- This is the bit where I run some commands in a terminal and you all say “oooh!” and “aaah!” like you’re impressed.
- ...seriously, I could really use the ego boost.

Some Core Utils

- grep - search for patterns in files or stdin
- sed - edit the input stream
- awk - general purpose text-processing language
- cat - con*cat*enate files
- find - list files recursively and apply filters
- sort - sort the lines from stdin
- uniq - remove duplicate lines from stdin
- xargs - run a command using each line from stdin as an argument
- tee - copy stdin to a file and to the screen

IO Streams

- A linux process has three standard streams:
  - stdin (file descriptor 0)
  - stdout (file descriptor 1)
  - stderr (file descriptor 2)
- stdin defaults to your keyboard
- stdout and stderr default to your screen
- You can redirect the standard streams
  - ‘< file’ connects a file to stdin
  - ‘> file’ redirects stdout to a file
  - ‘2> file’ redirects stderr to a file
  - ‘&> file’ redirects stdout *and* stderr to a file
  - ‘2>&1’ redirects stderr to stdout!
- Demo time...

Subshell Tricks

- <(cmd) - returns the output of ‘cmd’ as a file descriptor
  - Handy if you want to diff the output of two commands…
  - diff <(cmd-one) <(cmd-two)
- $(cmd) - returns the output text of ‘cmd’
  - Handy if you want to store the command output in a variable
  - myvar=$(cmd)

Methodology

- I like recon :)
- Let’s:
  - Enumerate subdomains
  - Check for dangling CNAMEs
  - Request all the pages
  - Look for things in the results
- Maybe then I’ll take some requests :)

Enumerating Subdomains

- We *could* use external services
  - hackertarget.com
  - crt.sh
  - certspotter.com
- But it’s nice to complement that with good-old brute force
- You will need:
  - A target
  - A wordlist
  - Bash :)

Does it resolve? Only humans know for sure

```
Terminal
tom@scan:~▶ host example.com
example.com has address 93.184.216.34
example.com has IPv6 address 2606:2800:220:1:248:1893:25c8:1946
tom@scan:~▶ host lolwtfamidoing.com
Host lolwtfamidoing.com not found: 3(NXDOMAIN)
tom@scan:~▶
```

Enter Exit Codes

Terminal

```
tom@scan:~▶ host example.com
example.com has address 93.184.216.34
example.com has IPv6 address 2606:2800:220:1:248:1893:25c8:1946
tom@scan:~▶ echo $?
0
tom@scan:~▶ host lolwtfamidoing.com
Host lolwtfamidoing.com not found: 3(NXDOMAIN)
tom@scan:~▶ echo $?
1
tom@scan:~▶
```

Conditionals

lol.sh (~) - VIM

lol.sh buffers

```
1 #!/bin/bash
2
3 if this-command-works; then
4     run-this-command
5 fi
6
~
~
~
~
~
~
~
~
~
```

NORMAL lol.sh sh 100% ☰ 6/6 ㏑ : 1

"lol.sh" 6L, 66C written

Demo Time

- Yay! Demo time!

Command Oriented Programming

```
Terminal
tom@scan:~▶ if host example.com; then echo "IT RESOLVES \o/"; fi
example.com has address 93.184.216.34
example.com has IPv6 address 2606:2800:220:1:248:1893:25c8:1946
IT RESOLVES \o/
tom@scan:~▶ if host lolwtfamidoing.com; then echo "IT RESOLVES \o/"; fi
Host lolwtfamidoing.com not found: 3(NXDOMAIN)
tom@scan:~▶
```

Tidying It Up A Little

```
tom@scan:~▶ if host example.com &> /dev/null; then echo "IT RESOLVES!"; fi
IT RESOLVES!
tom@scan:~▶ if host lolwtfamidoing.com &> /dev/null; then echo "IT RESOLVES!"; fi

tom@scan:~▶
```

Loops

lol.sh (~) - VIM

lol.sh buffers

```
1 #!/bin/bash
2
3 while this-command-works; do
4     this-command
5 done
6
~
~
~
~
~
~
~
~
~
```

NORMAL lol.sh sh 100% ☰ 6/6 ㏑ : 1

"lol.sh" 6L, 65C written

More Demo Time

- I love demo time (:

Looping Over stdin

Terminal

```
tom@scan:~▶ while read sub; do echo "$sub.example.com"; done < subdomains.txt
www.example.com
m.example.com
test.example.com
staging.example.com
admin.example.com
cms.example.com
blog.example.com
tom@scan:~▶
```

Putting It Together

```
Terminal
tom@scan:~▶ while read sub; do if host "$sub.example.com" &> /dev/null; then echo
 "$sub.example.com"; fi; done < subdomains.txt
www.example.com
tom@scan:~▶
tom@scan:~▶ # This is getting messy :/
tom@scan:~▶
```

If you liked it you shoulda put a .sh on it

lol.sh (~) - VIM

lol.sh buffers

```
1 #!/bin/bash
2
3 while read sub; do
4     if host "$sub.example.com" &> /dev/null; then
5         echo "$sub.example.com"
6     fi
7 done < subdomains.txt
8
~
~
~
~
~
~
~
NORMAL  lol.sh                                  sh  100% ☰  8/8 ℅ :  1
"lol.sh" 8L, 144C
```

I Like It Generic

lol.sh (~) - VIM

lol.sh buffers

```
1 #!/bin/bash
2
3 domain=$1
4 while read sub; do
5     if host "$sub.$domain" &> /dev/null; then
6         echo "$sub.$domain"
7     fi
8 done
9
~
~
~
~
~
~
```

NORMAL lol.sh sh 100% ☰ 9/9 ㏑ : 1

"lol.sh" 9L, 129C

Permissions

lol.sh (~) - VIM

```
tom@scan:~▶ mv lol.sh subs.sh
tom@scan:~▶ ./subs.sh example.com < subdomains.txt
-bash: ./subs.sh: Permission denied
tom@scan:~▶ chmod +x subs.sh
tom@scan:~▶ ./subs.sh example.com < subdomains.txt
www.example.com
tom@scan:~▶ cat subdomains.txt | ./subs.sh example.net
www.example.net
tom@scan:~▶
```

Dangling CNAMEs

lol.sh (~) - VIM

```
tom@scan:~▶ host invalid.sbtuk.net
Host invalid.sbtuk.net not found: 3(NXDOMAIN)
tom@scan:~▶ host -t CNAME invalid.sbtuk.net
invalid.sbtuk.net is an alias for lolifyouregisteredthisyouwastedyourmoney.com.
tom@scan:~▶ host lolifyouregisteredthisyouwastedyourmoney.com
Host lolifyouregisteredthisyouwastedyourmoney.com not found: 3(NXDOMAIN)
tom@scan:~▶
```

The Plan

- Check subdomains for CNAME records
- Check if those CNAMEs resolve
- ...profit?
- Demo time :)

Getting the CNAMEs

lol.sh (~) - VIM

```
tom@scan:~▶ host -t CNAME invalid.sbtuk.net | grep 'alias for'
invalid.sbtuk.net is an alias for lolifyouregisteredthisyouwastedyourmoney.com.
tom@scan:~▶ host -t CNAME invalid.sbtuk.net | grep 'is an al' | awk '{print $NF}'
lolifyouregisteredthisyouwastedyourmoney.com.
tom@scan:~▶
```

Incase That Demo Went Badly...

check-cnames.sh (~) - VIM

check-cnames.sh buffers

```
1  #!/bin/bash
2
3  domain=$1
4  while read sub; do
5      host -t CNAME "$sub.$domain" | grep 'alias for' | awk '{print $NF}' |
6      while read cname; do
7          if ! host "$cname" &> /dev/null; then
8              echo "$cname doesn't resolve ($sub.$domain)"
9          fi
10     done
11 done
12
~
~
~
```

NORMAL check-cnames.sh sh 100% ☰ 12/12 ㏑ : 1

"check-cnames.sh" 12L, 270C written

Fetch All The Things

- Having lots of targets to look at can be overwhelming
- Ddddddddemo time

A Thing To Fetch All The Things

fetch.sh (~/bsides) - VIM

fetch.sh index buffers

```
1  #!/bin/bash
2
3  mkdir -p out
4
5  while read url; do
6      filename=$(echo "$url" | md5sum | awk '{print $1}')
7      filename="out/$filename"
8      curl -sk "$url" -o "$filename" &> /dev/null
9      echo "$filename $url" >> index
10 done
11
~
~
~
~
```

NORMAL fetch.sh sh 100% ☰ 11/11 ㏑ : 1

"fetch.sh" 11L, 220C written

Finding Things In The Output

fetch.sh (~/bsides) - VIM

```
tom@scan:~/bsides▶ ./fetch.sh < urls
tom@scan:~/bsides▶ grep -HnroiE '<title>(.*)</title>'
out/56a6e4a8b88694e855ec457024babb4e:306:<title>BBC - Home</title>
out/639c2c4f448073d571a5135fbc1a0339:1:<title>Google</title>
out/cec0c034699dabe9891744f12fd63379:4:<title>Example Domain</title>
out/d3397772b65f89f729c434637946caf8:4:<title>Example Domain</title>
tom@scan:~/bsides▶ cat index
out/d3397772b65f89f729c434637946caf8 http://example.com
out/cec0c034699dabe9891744f12fd63379 https://example.net
out/639c2c4f448073d571a5135fbc1a0339 https://www.google.com
out/56a6e4a8b88694e855ec457024babb4e https://bbc.co.uk
tom@scan:~/bsides▶
```

Some Things To Grep For

- Titles
- Server headers
- Known ‘subdomain takeover’ strings
- URLs (and then go and fetch the URLs!)
  - JavaScript files are nice (:
- Secrets
- Error messages
- File upload forms
- Interesting Base64 encoded strings ;)
  - (eyJ|YTo|Tzo|PD[89])
- Demo time, obv.

When Your Outputs Are Your Inputs

- Let’s look for some s3 buckets…
- D
- E
- M
- O

When In Doubt: Use Your Eyes

- Deeeeeeeemo time
- It’s demo time
- Time for a demo
- I like demos :)

Speeding Things Up

- Pipes give you *some* parallelisation for free
  - It’s not enough though, is it?
- xargs can run things in parallel…
- Let’s speed up our subdomain brute-forcer
- What time is it?
  - It’s demo time.

A Bit Of A Mess

parsub.sh + (~/bsides) - VIM

parsub.sh+ buffers

```
1 #!/bin/bash
2
3 domain=$1
4 xargs -P1 -n1 -I{} bash -c "
5     if host \"{}.$domain\" &> /dev/null; then
6         echo \"{}.$domain\"
7     fi
8 "
9
```

~
~
~
~
~
~

NORMAL parsub.sh[+] sh 100% ☰ 9/9 ㏑ : 1

A Little Cleaner

parsub.sh (~/bsides) - VIM

```
parsub.sh  sub.sh                                                    buffers
  1 #!/bin/bash
  2 domain=$1
  3 if host "$domain" &> /dev/null; then
  4     echo "$domain"
  5 fi
~
~
 sub.sh                                                sh     20% ☰    1/5 ℓ :  1
  1 #!/bin/bash
  2 domain=$1
  3 xargs -P10 -n1 -I{} ./sub.sh "{}.$domain"
  4
~
~
~
NORMAL  parsub.sh                                      sh  100% ☰    4/4 ℓ :  1
"sub.sh" 5L, 81C
```

Bits And Bobs

- Use dtach for long-running tasks
- vim is a major part of my workflow
- When things get complex, consider a different language…
  - I like Go :)
  - Check out meg, comb, unfurl, waybackurls, gf, httprobe, concurl...

Any Requests?

- This is risky isn’t it?
- Questions? I love questions