Blind Return Oriented Programming

Nathan Huckleberry

University of Texas at Austin

February 21, 2020

Table of Contents

Table of Contents

Attack Assumptions

- The target binary is a "fork server"
- It accepts one connection, forks a child to handle the connection
- Parent continues handling connections
- There is a stack-based buffer overflow we can exploit

Attack Assumptions

- The address space is not re-randomized after a fork
- Stack canaries are not re-randomized after a fork

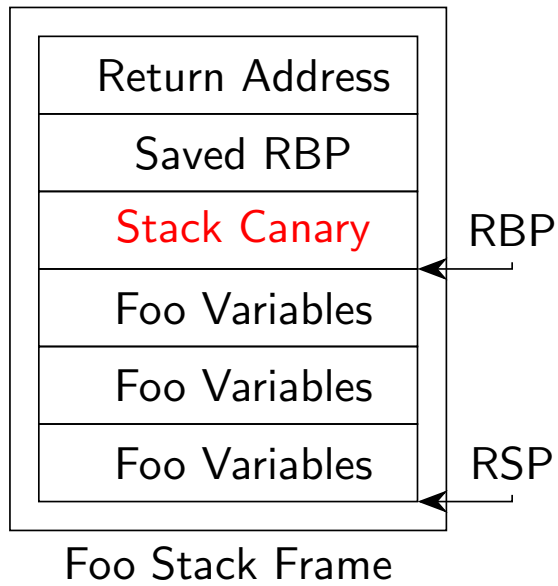
Table of Contents

Stack Canaries

- Stack canaries are a protection from buffer overflow attacks
- A random number is placed between the return address and function local variables
- The value of the canary is compared to a global copy in the function prologue
- If the canary value is different we know to crash

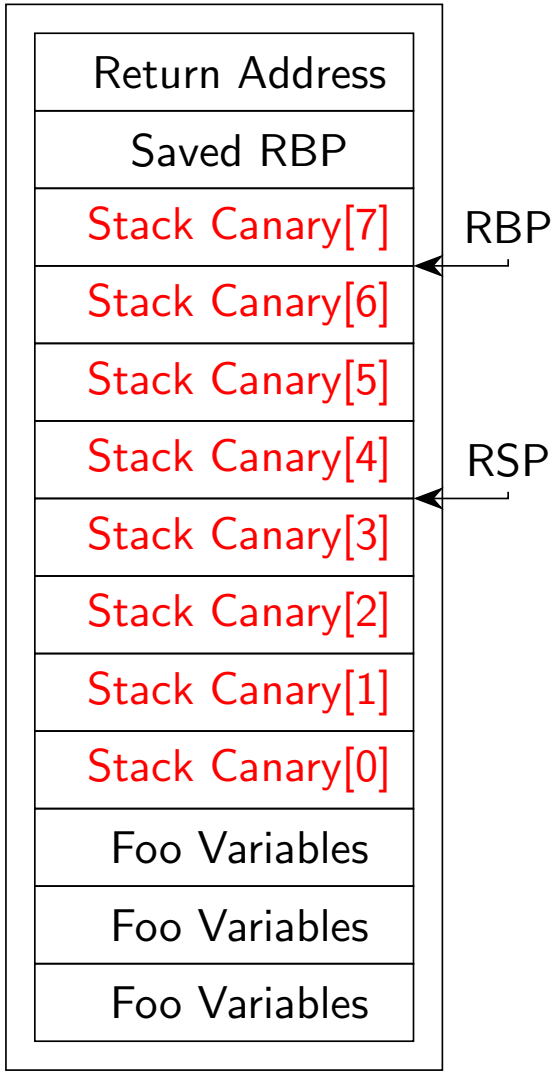
Stack Canaries

- Stack overflow overwrites canary
- Program crashes



Crash Oracle

- Guess the lowest byte of the stack canary.
- If the program crashes we guessed wrong.
- If the program continues we guessed correctly.



Crash Oracle

- Stack canary is not re-randomized.
- This allows us to bruteforce the entire canary.
- Bruteforcing beyond canary gives stack and code addresses, breaking ASLR.

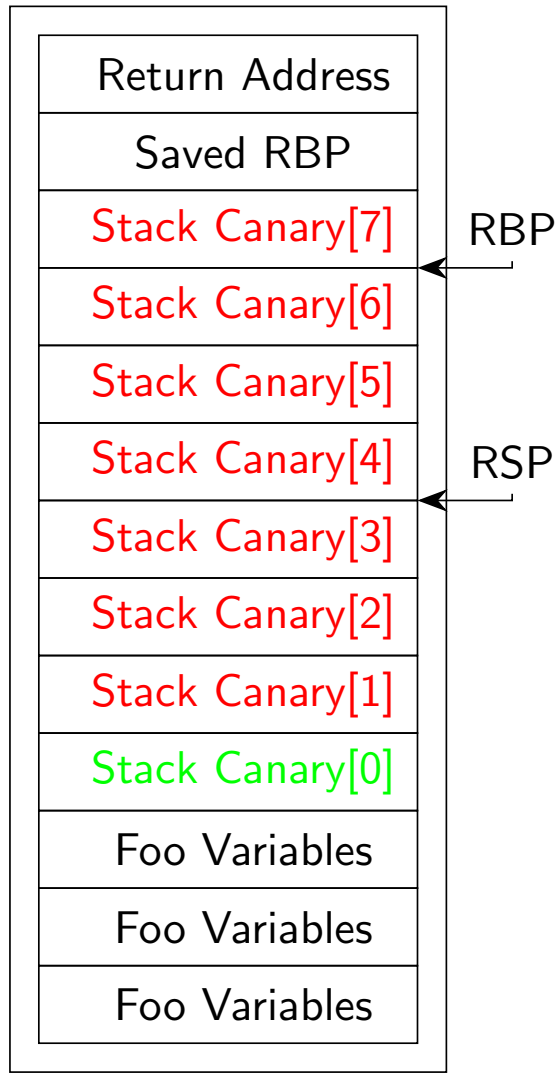


Table of Contents

Eventual Goals

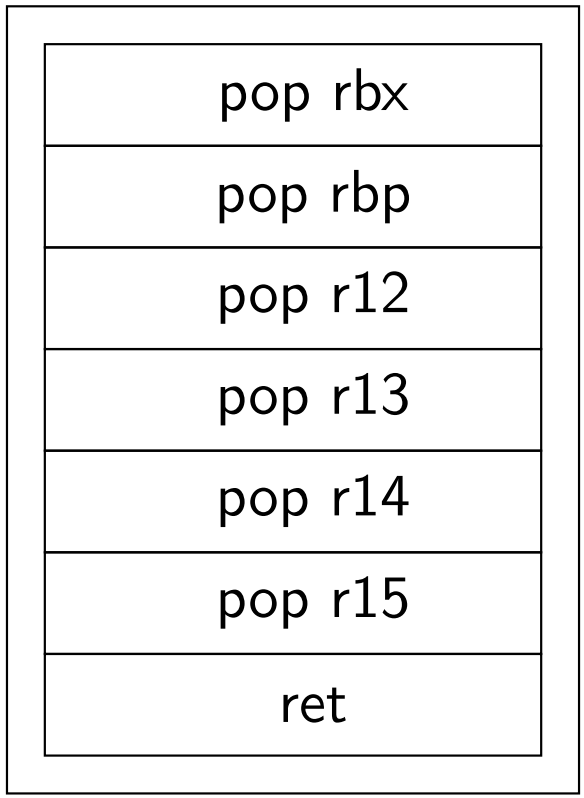
- Find the `write` function
- Call `write` to network socket
- Write binary code segment to attacker machine
- Do regular ROP

Controlling Argument Registers

- Write accepts 3 parameters passed in `rdi`, `rsi`, `rdx`
- To use write we must control these registers

BROP Gadget

- The BROP Gadget pops 6 times in a row then returns



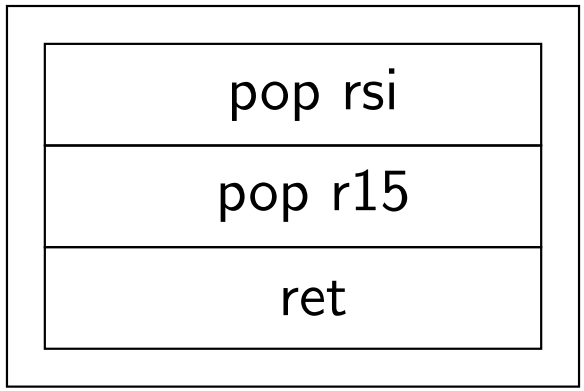
BROP Gadget

BROP Gadget

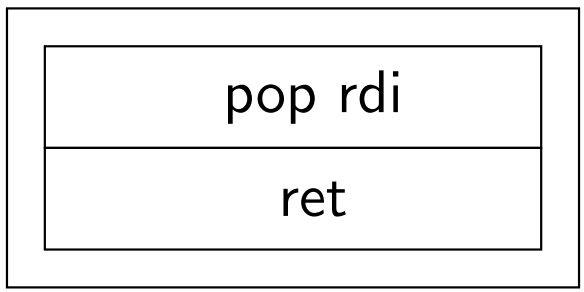
- The BROP Gadget appears in `__libc_csu_init` which is called before `main`
- A gadget that pops 6 times in a row is rare

BROP Gadget

- Jumping to the gadget plus an offset gives useful gadgets
- Finding a BROP gadget gives access to rdi and rsi



BROP Gadget +0x7



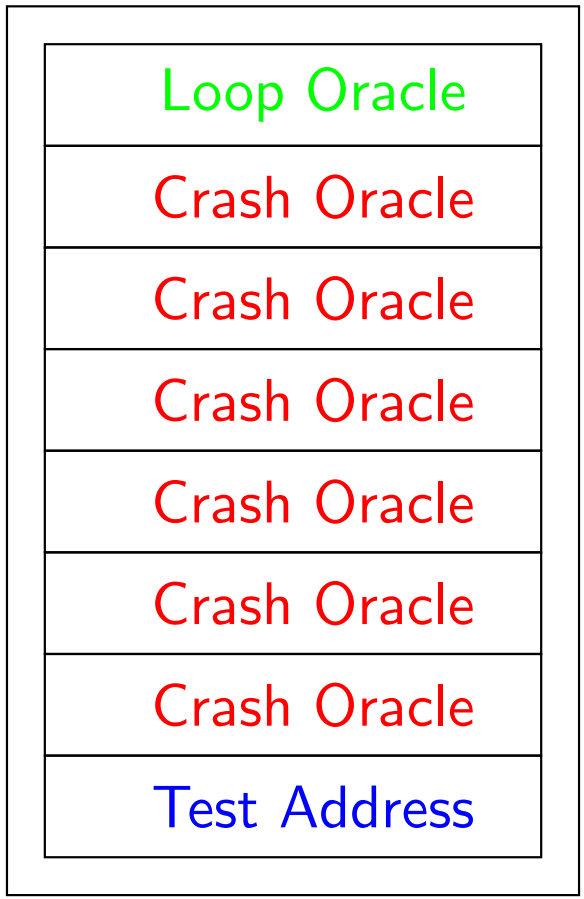
BROP Gadget +0x9

Crash and Loop Oracles

- Loop oracles are addresses that cause infinite loops
- The connection to the attacker will stay open
- Crash oracles are addresses that cause the program to crash
- The connection to the attacker will close

Finding the BROP Gadget

- We can construct a BROP Oracle with a ROP chain of crash and loop oracle addresses



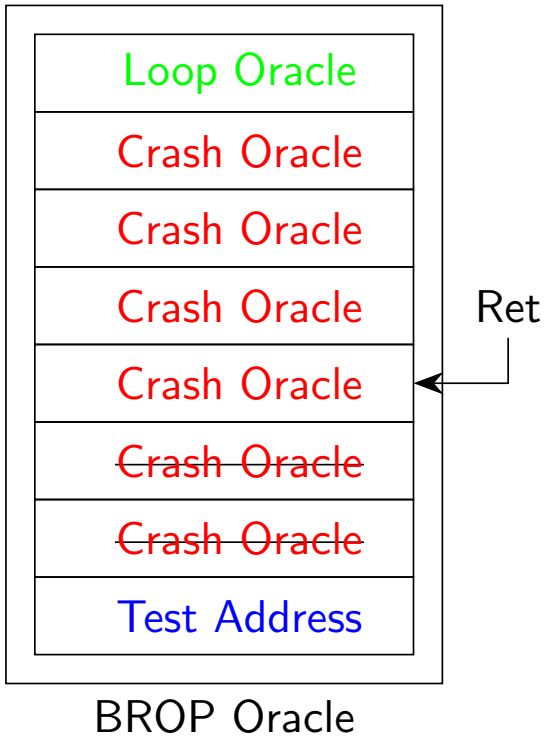
BROP Oracle

Finding the BROP Gadget

Listing 1: Test Address

```
1 pop rdi
2 pop rdx
3 add rdi, rdx
4 ret
```

- Crashes with 2 pops

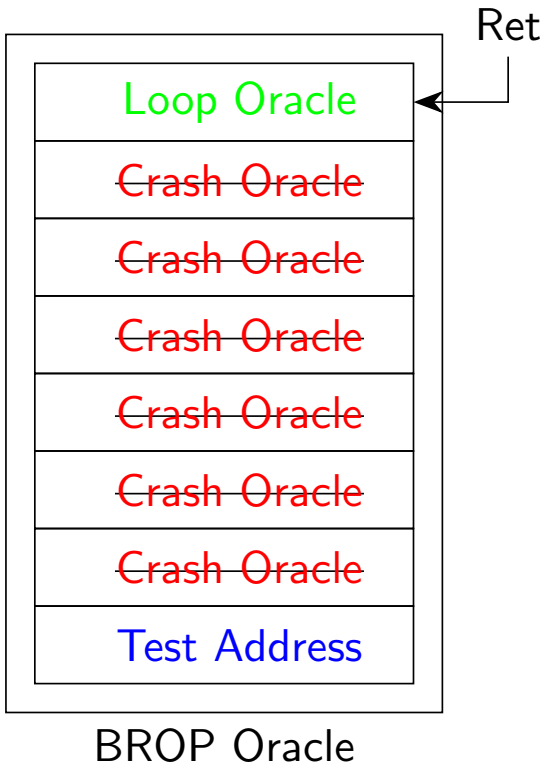


Finding the BROP Gadget

Listing 2: Test Address

```
1 pop rbx
2 pop rbp
3 pop r12
4 pop r13
5 pop r14
6 pop r15
7 ret
```

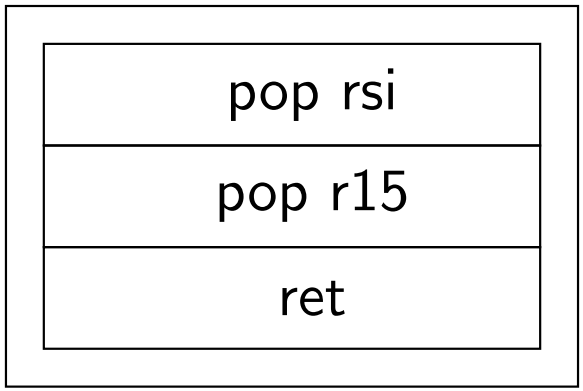
- Infinite loops with 6 pops



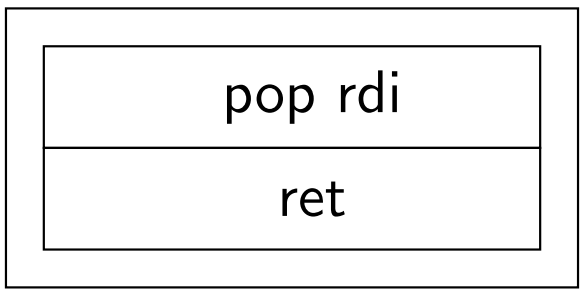
Finding the BROP Gadget

- Guess random addresses starting at `0x400000`
- If the connection stays open, we've found a BROP gadget
- We can use this gadget to control `rdi` and `rsi`

Using the BROP Gadget



BROP Gadget +0x7



BROP Gadget +0x9

Set `rdi` `rsi` Gadget

Table of Contents

How to control RDX

- The function `strcmp` happens to set `rdx` to the length of the string inserted.
- We can find `strcmp` in the PLT
- The PLT will be near `0x400000`

What is the PLT

- PLT is a table where each entry is three instructions that jump to libc
- Each is 0x10 bytes

```
0000000000401040 <puts@plt>:
401040: ff 25 da 2f 00 00 jmpq *0x2fda(%rip)
401046: 68 01 00 00 00 pushq $0x1
40104b: e9 d0 ff ff ff jmpq 401020 <.plt>
0000000000401050 <write@plt>:
401050: ff 25 d2 2f 00 00 jmpq *0x2fd2(%rip)
401056: 68 02 00 00 00 pushq $0x2
40105b: e9 c0 ff ff ff jmpq 401020 <.plt>
```

Looking for Strcmp

- Iterate every `0x10` bytes near `0x400000`
- Test if we are at `strcmp`

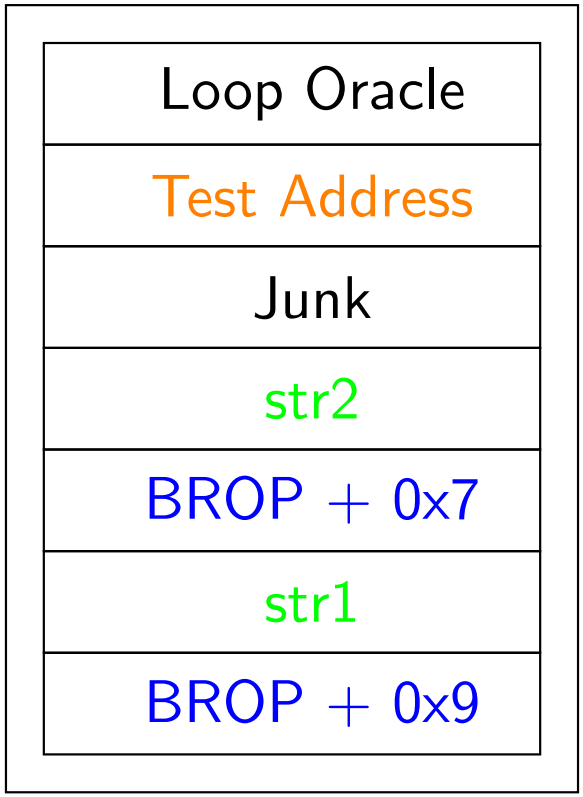
Strcmp Oracle

- `strcmp(bad, bad)` crashes
- `strcmp(good, bad)` crashes
- `strcmp(bad, good)` crashes
- `strcmp(good, good)` does not crash
- Where `good` is a readable address and `bad` is 0x0.

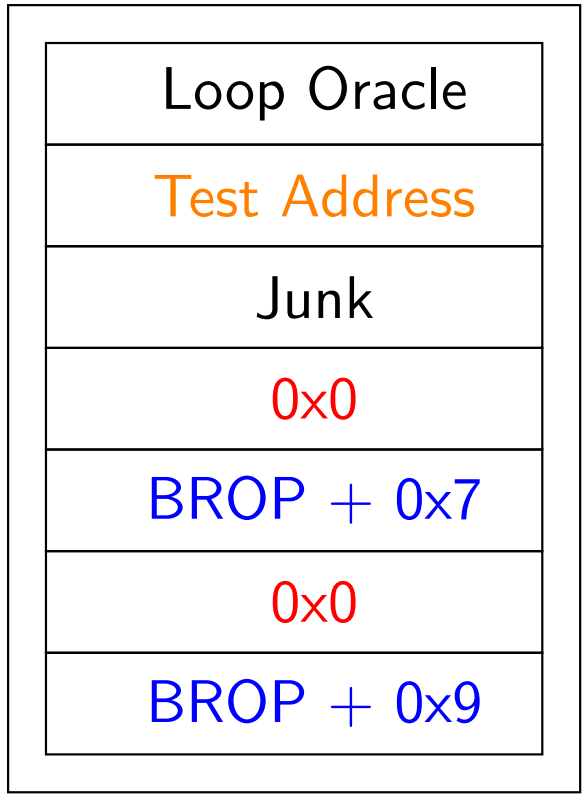
Strcmp Oracle

- We can construct an oracle for `strcmp` the same way we made the BROP Gadget oracle
- This gives us the ability to overwrite `rdx`

Strcmp Oracle



Loops if `strcmp`



Crashes if `strcmp`

Strcmp Oracle

- False positives for `strcmp` are possible, but most of them also set `rdx`
- False positives examples are `strncmp`, `strcasecmp`

Setting RDX

- We can now set `rdx` to a non-zero value by inputting two non-zero length strings.

Table of Contents

Finding Write

- There is some file descriptor that writes to the attacker
- File descriptors are numbers [0, 1024] by POSIX standards
- The lowest possible descriptor is assigned when a file descriptor is requested

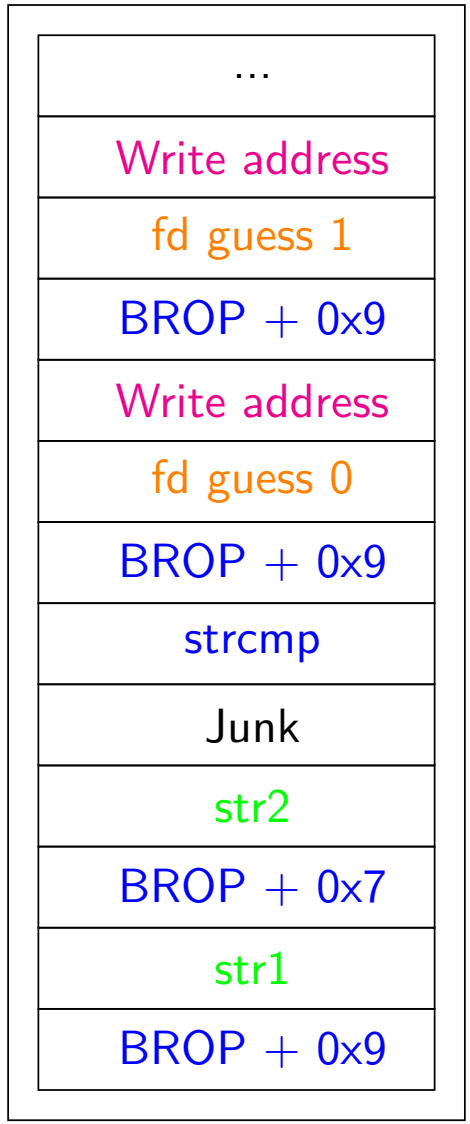
Finding Write

- We can guess the file descriptor trying all 0 to 1024
- If the attacker receives any bytes, we've found the correct file descriptor
- `write` is in the PLT
- PLT entries are `0x10` away from each other
- Iterating PLT entries will find `write`

Listing 3: Test Address

```
1 ssize_t write(int fildes, const void *buf, size_t nbyte);
```

Finding Write



Find `write`

Finding Write

- We can now use write to send the binary across the network to the attacker
- We can now do a regular ROP