Conceal ROP gadgets for AArch64 COTS binary

Dongli Zhang
Oracle Asia Research and Development Centers (Beijing)
dongli.zhang@oracle.com

December 11, 2017
Plan

- ROP Attack: Return Oriented Programming Attack
- ELF and AArch64
- NORAX: eXecute-Only-Memory (XOM) on AArch64
Code Injection Attack

- Stack Smashing: to inject and run shellcode in stack
- Linux x86_64 Calling Convention: RDI, RSI, RDX, RCX, R8, R9, XMM07

```c
/* victim.c */
int main()
{
    char name[64];
    puts("What's your name? ");
    gets(name);
    printf("Hello, %s!\n", name);
    return 0;
}
```

Address of `name[64]`: 0x0
Input of `gets(name)`:
Overwrite

Dongli Zhang (Oracle)
Stack Smashing Mitigations

- Stack Canary

- DEP (Data Execution Prevention): \( W \oplus X \) (NX bit)
  - Attackers are not able to execute any injected code!
  - To disable via:
    ```
gcc -fno-stack-protector -o victim victim.c
```
Stack Canary

- To disable via: `gcc -fno-stack-protector -o victim victim.c`

DEP (Data Execution Prevention): $W \oplus X$ (NX bit)

Attackers are not able to execute any injected code!

To disable via: `execstack -s victim`

ASLR (Address Space Layout Randomization)

Transparent runtime randomization for security. SRDS 2003.

To disable via:
- `setarch 'arch' -R ./victim`
- `echo 0 > /proc/sys/kernel/randomize_va空间`
Stack Smashing Mitigations

- Stack Canary
  - To disable via: `gcc -fno-stack-protector -o victim victim.c`

- DEP (Data Execution Prevention): $W \oplus X$ (NX bit)
  - Attackers are not able to execute any injected code!
  - To disable via: `execstack -s victim`
Stack Smashing Mitigations

- Stack Canary
  - To disable via: `gcc -fno-stack-protector -o victim victim.c`

- DEP (Data Execution Prevention): $W \oplus X$ (NX bit)
  - Attackers are not able to execute any injected code!
  - To disable via: `execstack -s victim`

- ASLR (Address Space Layout Randomization)
  - Transparent runtime randomization for security. SRDS 2003.
Stack Smashing Mitigations

- **Stack Canary**
  - To disable via: `gcc -fno-stack-protector -o victim victim.c`

- **DEP (Data Execution Prevention)**: $W \oplus X$ (NX bit)
  - Attackers are not able to execute any injected code!
  - To disable via: `execstack -s victim`

- **ASLR (Address Space Layout Randomization)**
  - Transparent runtime randomization for security. SRDS 2003.
  - To disable via: `setarch 'arch' -R ./victim`
  - To disable via: `echo 0 > /proc/sys/kernel/randomize_va_space`
Code Reuse Attack (1/2)

- Gadgets: instruction sequence ended with "ret" instruction within existing program or libraries already present in memory
- ROP (Return Oriented Programming): to perform arbitrary operations by chaining relevant gadgets to bypass DEP

```c
/* victim.c */
int main()
{
    char name[64];
    puts("What's your name?\n");
    gets(name);
    printf("Hello, %s
", name);
    return 0;
}
```

Dongli Zhang (Oracle)
Conceal ROP gadgets for AArch64 COTS binary
December 11, 2017
1. inject ROP payload

Diagram:
- Adversary
- Stack
- Return Address 3
  - Return Address 2
  - Return Address 1
  - Heap Vulnerability
  - Code (Executable)
  - Libraries
    - ADD Gadget
    - LOAD Gadget
    - STORE Gadget
  - Stack Pivot
  - Program Memory
Code Reuse Attack (2/2)

1. inject ROP payload
2. hijack control flow
Code Reuse Attack (2/2)

1. inject ROP payload
2. hijack control flow
3. stack pivot sequences (e.g., mov %eax, %esp; ret)
Code Reuse Attack (2/2)

1. inject ROP payload
2. hijack control flow
3. stack pivot sequences (e.g., mov %eax, %esp; ret)
4. "ret" redirects to ROP payload
Code Reuse Attack (2/2)

1. inject ROP payload
2. hijack control flow
3. stack pivot sequences (e.g., mov %eax, %esp; ret)
4. "ret" redirects to ROP payload
5. ROP gadget and ret
6. ROP gadget and ret
7. ROP gadget and ret

Dongli Zhang (Oracle) Conceal ROP gadgets for AArch64 COTS binary December 11, 2017 6 / 35
- function order permutation
- basic block order permutation
- swap registers and replace instructions
- instruction location randomization
Thread Model Assumption

- Exercise a vulnerable entry point
- Execute arbitrary malicious computations
Memory Disclosure

Direct Memory Disclosure
- read instructions in code page
- return address
- function pointer
- dynamic linking information
- c++ vtable & exception handler

Indirect Memory Disclosure
- return address
- function pointer
- dynamic linking information
- c++ vtable & exception handler

return address
- on stack or code pointer in data section

Control flow transfer
- cross module call

...

- Fine-grained code diversification via LLVM
- Code and data separation via Intel EPT and LLVM
- Code-pointer hiding via LLVM
- Does not support COTS binary
New and Hard Problem

- Enable XOM on Android AArch64 COTS binaries (NORAX)
- Hide code pointers in data section (future work)

COTS

AArch64

No Compiler Change
# aarch64-linux-gnu-strip <binary>

- without symbol information

```assembly
0000000000002460 <main>:
2460: d111c3ff    sub sp, sp, #0x470
2464: a9ba7bfd    stp x29, x30, [sp,#-96]!
2468: 910003fd    mov x29, sp
...
3484: 8b3f63e0    add x0, sp, xzr
3488: 17ffffea    b   3430 <do_arm64_start>
348c: a9be7bfd    stp x29, x30, [sp,#-32]!
3490: 910003fd    mov x29, sp
...
34b4: a9be7bfd    stp x29, x30, [sp,#-32]!
34b8: 910003fd    mov x29, sp
```

Original Binary

COTS Binary

Dongli Zhang (Oracle)

Conceal ROP gadgets for AArch64 COTS binary

December 11, 2017 13 / 35
ELF - Linking vs. Execution

- segments sample
  - INTERP
  - LOAD
  - DYNAMIC

- sections sample
  - .interp
  - .dynsym, .dynamic
  - .rela.dyn, .rela.plt, .got.plt, .got
  - .plt, .text
  - .data, .rodata, .bss

- manuals
  - Executable and Linkable Format (ELF)
  - ELF for the ARM Architecture
  - ELF for the ARM 64-bit Architecture (AArch64)

<table>
<thead>
<tr>
<th>Linking View</th>
<th>Execution View</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELF header</td>
<td>ELF header</td>
</tr>
<tr>
<td>Program header table</td>
<td>Program header table</td>
</tr>
<tr>
<td>Section 1</td>
<td>Segment 1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Section n</td>
<td>Segment 2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Section header table</td>
<td>Section header table</td>
</tr>
</tbody>
</table>

(optional)
user space loads executable binary via exec system call
1. user space loads executable binary via exec system call
2. kernel loads executable binary and dynamic linker into memory
1. User space loads executable binary via exec system call
2. Kernel loads executable binary and dynamic linker into memory
3. Dynamic linker performs linking jobs while loading all prerequisite libraries (Android is without lazy address resolution)
ELF - load executable ELF

1. user space loads executable binary via exec system call
2. kernel loads executable binary and dynamic linker into memory
3. dynamic linker performs linking jobs while loading all prerequisite libraries (Android is without lazy address resolution)
4. start the executable binary
1. user space loads executable binary via exec system call
2. kernel loads executable binary and dynamic linker into memory
3. dynamic linker performs linking jobs while loading all prerequisite libraries (Android is without lazy address resolution)
4. start the executable binary
5. resolve dynamic symbol on-demand by linker
Suppose ./test calls function puts() belong to libc (lazy address resolution):

1. ./test calls puts@plt belong to plt section
Suppose ./test calls function puts() belong to libc (lazy address resolution):

1. ./test calls puts@plt belong to plt section
2. puts@plt redirects to puts in got.plt which points to corresponding handler in ld
Suppose ./test calls function puts() belong to libc (lazy address resolution):

1. ./test calls puts@plt belong to plt section
2. puts@plt redirects to puts in got.plt which points to corresponding handler in ld
3. ld calculates the hash of symbol name (puts), traverses each libraries and searches in buckets of gnu.hash with the hash value to identify the index of puts() in dynsym section
Suppose ./test calls function puts() belong to libc (lazy address resolution):

1. ./test calls puts@plt belong to plt section
2. puts@plt redirects to puts in got.plt which points to corresponding handler in ld
3. ld calculates the hash of symbol name (puts), traverses each libraries and searches in buckets of gnu.hash with the hash value to identify the index of puts() in dynsym section
4. Once entry of puts in dynsym is identified, the address of puts would be written to got.plt with the help of binary’s rela.plt
• instructions: 4-byte aligned and fixed size

AArch64 CPU Exception Level

- EL0: User Mode (Trustzone Normal World)
- EL1: Kernel Mode (Trustzone Normal World)
- EL2: Hypervisor Mode (Trustzone Normal World)
- EL3: Secure Monitor (Trustzone Secure World)
- N/A

Dongli Zhang (Oracle)
- instructions: 4-byte aligned and fixed size
- mode: user (EL0), kernel (EL1), hypervisor (EL2) and secure monitor (EL3)

Embedded image:

AArch64 CPU Exception Level:
- EL0: User Mode (Trustzone Normal World)
- EL1: Kernel Mode (Trustzone Normal World)
- EL2: Hypervisor Mode (Trustzone Normal World)
- EL3: Secure Monitor (Trustzone Secure World)

N/A

Dongli Zhang (Oracle)
Conceal ROP gadgets for AArch64 COTS binary
December 11, 2017 17 / 35
- Instructions: 4-byte aligned and fixed size
- Mode: user (EL0), kernel (EL1), hypervisor (EL2) and secure monitor (EL3)
- Registers: X0-X30 (X29 is FP, X30 is LR), SP (PC is not accessible)

**AArch64 CPU Exception Level**

- **EL0:** User Mode (Trustzone Normal World)
- **EL1:** Kernel Mode (Trustzone Normal World)
- **EL2:** Hypervisor Mode (Trustzone Normal World)
- **EL3:** Secure Monitor (Trustzone Secure World)
- N/A

Dongli Zhang (Oracle)
AArch64

- instructions: 4-byte aligned and fixed size
- mode: user (EL0), kernel (EL1), hypervisor (EL2) and secure monitor (EL3)
- registers: X0-X30 (X29 is FP, X30 is LR), SP (PC is not accessible)
- branch: B, BL (Branch with Link), BLR (Branch with Link to Register), BR (Branch to Register), B.cond

AArch64 CPU Exception Level

- EL0: User Mode (Trustzone Normal World)
- EL1: Kernel Mode (Trustzone Normal World)
- EL2: Hypervisor Mode (Trustzone Normal World)
- EL3: Secure Monitor (Trustzone Secure World)

Dongli Zhang (Oracle)  Conceal ROP gadgets for AArch64 COTS binary  December 11, 2017  17 / 35
• instructions: 4-byte aligned and fixed size
• mode: user (EL0), kernel (EL1), hypervisor (EL2) and secure monitor (EL3)
• registers: X0-X30 (X29 is FP, X30 is LR), SP (PC is not accessible)
• branch: B, BL (Branch with Link), BLR (Branch with Link to Register), BR (Branch to Register), B.cond
• memory load: ADR, ADRP, LDR
AArch64

- instructions: 4-byte aligned and fixed size
- mode: user (EL0), kernel (EL1), hypervisor (EL2) and secure monitor (EL3)
- registers: X0-X30 (X29 is FP, X30 is LR), SP (PC is not accessible)
- branch: B, BL (Branch with Link), BLR (Branch with Link to Register), BR (Branch to Register), B.cond
- memory load: ADR, ADRP, LDR
- other: ARM Architecture Reference Manual: ARMv8, for ARMv8-A architecture profile
instructions: 4-byte aligned and fixed size
mode: user (EL0), kernel (EL1), hypervisor (EL2) and secure monitor (EL3)
registers: X0-X30 (X29 is FP, X30 is LR), SP (PC is not accessible)
branch: B, BL (Branch with Link), BLR (Branch with Link to Register), BR (Branch to Register), B.cond
memory load: ADR, ADRP, LDR
other: ARM Architecture Reference Manual: ARMv8, for ARMv8-A architecture profile
since Android 5.0 (Lolopop), non-PIE loading is no longer supported
NORAX: Enabling Execute-Only Memory for COTS Binaries on AArch64

*YaohuiChen,* DongliZhang,† RuowenWang,* RuiQiao,
†AhmedM.Azab,* LongLu,† HayawardhVijayakumar,† WenboShen

*Stony Brook University †Samsung Research America

IEEE Symposium on Security & Privacy (Oakland) 2017
## XOM on AArch64

- commit, revert and commit
  - 2016-08-25, arm64: Introduce execute-only page access permissions
  - 2014-05-16, Revert "arm64: Introduce execute-only page access permissions"
  - 2014-05-09, arm64: Introduce execute-only page access permissions
- last commit (2016-08-25): cab15ce604e550020bb7115b779013b91bcdbdc21
- gcc/llvm (AFAIK) does not support code-data sepearation

<table>
<thead>
<tr>
<th>AP[2:1]</th>
<th>EL0 Permission</th>
<th>EL1 Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Executable-only</td>
<td>Read/Write</td>
</tr>
<tr>
<td>01</td>
<td>Read/Write, Config-Executable</td>
<td>Read/Write</td>
</tr>
<tr>
<td>10</td>
<td>Executable-only</td>
<td>Read-only</td>
</tr>
<tr>
<td>11</td>
<td>Read, Executable</td>
<td>Read-only</td>
</tr>
</tbody>
</table>
1. separate data and code to different pages
2. properly update all references
Executable Data Relocation Sample

Read-only Data Relocation

Inline Data Relocation
NORAX Challenges

- rodata and executable inline data
  - Reference from code (.text)
  - Reference from symbol table (.dynsym)
  - Reference from relocation table (.rela.dyn)
  - Reference from global offset table (.got)
  - Reference from read-only global data (.data.rel.ro)
NORAX Challenges

- rodata and executable inline data
  - Reference from code (.text)
  - Reference from symbol table (.dynsym)
  - Reference from relocation table (.rela.dyn)
  - Reference from global offset table (.got)
  - Reference from read-only global data (.data.rel.ro)

- read-only ELF header
  - Reference from linker
NORAX Challenges

- rodata and executable inline data
  - Reference from code (.text)
  - Reference from symbol table (.dynsym)
  - Reference from relocation table (.rela.dyn)
  - Reference from global offset table (.got)
  - Reference from read-only global data (.data.rel.ro)
- read-only ELF header
  - Reference from linker
- .eh_frame_hdr/.eh_frame
  - Reference from C++ runtime
Design Goals

Code-Data Separation: precision vs. practical

- A complete set of executable data
- A subset of references
## Design Goals

### Code-Data Separation: precision vs. practical
- A complete set of executable data
- A subset of references

### Security
- Expose as less code as possible
- Enforce policy based security on missed references
Design Goals

Code-Data Separation: precision vs. practical
- A complete set of executable data
- A subset of references

Security
- Expose as less code as possible
- Enforce policy based security on missed references

Practicability
- Low runtime and memory overhead
- Non-exclusive binary hardening solution
- Backward compatibility
- Modularity support
NORAX Framework

- NDisassembler: collect executable data and references
- NPatcher: static binary transformation
- NLoader: update executable data references
- NMonitor: runtime policy check for false-positive
Algorithm 1 and Algorithm 2 in NORAX paper for details

1. Linear-sweep disassembly (objdump -d)
2. Identify executable data position (rodata or inline) and reference (adr(p) or ldr)
3. For unbounded data, collect a set of over-approximated date via Unbounded Data Expansion (Algorithm 2)
**NORAX: NPatcher**

- New memory layout
  - New location of the executable data
  - Take into consideration reference addressing range, and emit stub code if needed
- Append NORAX-related metadata to the end
  - Duplicated inline data
  - References locations and displacements
  - Stub code
  - NORAX header
- **Ld-1**: Setup NORAX book-keeping data and new mapping of read-only data and sections
- **Ld-2**: Redirect .dynamic access to new read-only sections
- **Ld-3**: Adjust all referencees and enable XOM
Missed reference to embedded data
- NDisassembler may miss some references
- Reference to .eh_frame_hdr and .eh_frame

1) Check if access allowed
2) Emulate load instruction if allowed
Evaluation - transformation correctness

- LG Nexus 5X (Qualcomm Snapdragon 808 MSM8992 (4 x ARM Cortex-A53 & 2 x ARM Cortex-A57), and 2GB RAM)
- Android OS v6.0.1 (Marshmallow) with Linux kernel v3.14 (64-bit)
- Changed bionic linker and linux kernel
- Tested for 20 core system binaries

<table>
<thead>
<tr>
<th>System Modifications</th>
<th>Norax Components</th>
<th>SLoC</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux Kernel</td>
<td>NLoader, NMonitor</td>
<td>1947</td>
<td>C</td>
</tr>
<tr>
<td>Bionic Linker</td>
<td>NLoader</td>
<td>289</td>
<td>C++</td>
</tr>
<tr>
<td>Analysis &amp; Rewriting Modules</td>
<td>NDisassembler, NPatch</td>
<td>3580</td>
<td>Python &amp; Bash Shell Script</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module</th>
<th>Size (Stock)</th>
<th>Size (NORAX)</th>
<th>File Size Overhead</th>
<th># of Rewrite Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>486,032</td>
<td>517,736</td>
<td>5.49%</td>
<td>0</td>
</tr>
<tr>
<td>toybox</td>
<td>310,800</td>
<td>322,888</td>
<td>3.89%</td>
<td>0</td>
</tr>
<tr>
<td>toolbox</td>
<td>148,184</td>
<td>154,632</td>
<td>4.35%</td>
<td>0</td>
</tr>
<tr>
<td>dhcpcd</td>
<td>112,736</td>
<td>116,120</td>
<td>3.00%</td>
<td>0</td>
</tr>
<tr>
<td>logd</td>
<td>83,904</td>
<td>86,256</td>
<td>2.80%</td>
<td>0</td>
</tr>
<tr>
<td>infalld</td>
<td>72,152</td>
<td>76,896</td>
<td>6.58%</td>
<td>0</td>
</tr>
<tr>
<td>app_process64 (zygote)</td>
<td>22,456</td>
<td>23,016</td>
<td>2.49%</td>
<td>0</td>
</tr>
<tr>
<td>gseecond</td>
<td>14,584</td>
<td>15,093</td>
<td>3.07%</td>
<td>0</td>
</tr>
<tr>
<td>surfaceflinger</td>
<td>14,208</td>
<td>14,448</td>
<td>1.69%</td>
<td>0</td>
</tr>
<tr>
<td>rld</td>
<td>14,216</td>
<td>14,784</td>
<td>4.00%</td>
<td>0</td>
</tr>
<tr>
<td>libart.so</td>
<td>7,512,272</td>
<td>7,772,520</td>
<td>3.46%</td>
<td>0</td>
</tr>
<tr>
<td>libstagefright.so</td>
<td>1,883,288</td>
<td>1,946,328</td>
<td>3.35%</td>
<td>0</td>
</tr>
<tr>
<td>libcrypt.so</td>
<td>1,137,280</td>
<td>1,157,816</td>
<td>1.81%</td>
<td>0</td>
</tr>
<tr>
<td>libmedia.so</td>
<td>1,058,616</td>
<td>1,071,712</td>
<td>1.24%</td>
<td>0</td>
</tr>
<tr>
<td>libc.so</td>
<td>1,032,392</td>
<td>1,051,312</td>
<td>1.83%</td>
<td>0</td>
</tr>
<tr>
<td>libc++.so</td>
<td>944,056</td>
<td>951,632</td>
<td>0.80%</td>
<td>0</td>
</tr>
<tr>
<td>libssl.so</td>
<td>791,176</td>
<td>805,784</td>
<td>1.85%</td>
<td>0</td>
</tr>
<tr>
<td>libbinder.so</td>
<td>325,416</td>
<td>327,072</td>
<td>0.51%</td>
<td>0</td>
</tr>
<tr>
<td>libm.so</td>
<td>235,544</td>
<td>293,744</td>
<td>24.71%</td>
<td>0</td>
</tr>
<tr>
<td>libandroid.so</td>
<td>96,032</td>
<td>97,208</td>
<td>1.22%</td>
<td>0</td>
</tr>
<tr>
<td>AVG.</td>
<td></td>
<td></td>
<td>3.91%</td>
<td>0</td>
</tr>
</tbody>
</table>
## Evaluation - functionality test

### Figure: Functionality Test Result

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
<th>Experiment</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vold</code></td>
<td>Volume daemon</td>
<td>mount SD Card; umount</td>
<td>Yes</td>
</tr>
<tr>
<td><code>toybox</code></td>
<td><strong>115</strong> *nix utilities</td>
<td>try all commands</td>
<td>Yes</td>
</tr>
<tr>
<td><code>toolbox</code></td>
<td><strong>22</strong> core *nix utilities</td>
<td>try all commands</td>
<td>Yes</td>
</tr>
<tr>
<td><code>dhcpd</code></td>
<td>DHCP daemon</td>
<td>obtain dynamic IP address</td>
<td>Yes</td>
</tr>
<tr>
<td><code>logd</code></td>
<td>Logging daemon</td>
<td>collect system log for 1 hour</td>
<td>Yes</td>
</tr>
<tr>
<td><code>install</code></td>
<td>APK install daemon</td>
<td>install 10 APKs</td>
<td>Yes</td>
</tr>
<tr>
<td><code>app_process64 (zygote)</code></td>
<td>Parent process for all applications</td>
<td>open 20 apps; close</td>
<td>Yes</td>
</tr>
<tr>
<td><code>qseecomd</code></td>
<td>Qualcomm's proprietary driver</td>
<td>boot up the phone</td>
<td>Yes</td>
</tr>
<tr>
<td><code>surfaceflinger</code></td>
<td>Compositing frame buffers for display</td>
<td>Take 5 photos; play 30 min movie</td>
<td>Yes</td>
</tr>
<tr>
<td><code>ril</code></td>
<td>Baseband service daemon</td>
<td>Have 10 min phone call</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Figure: Compatibility evaluation with Android Compatibility Test Suite (CTS)

<table>
<thead>
<tr>
<th>Plan Name</th>
<th>Pass</th>
<th>Fail</th>
<th>Not Executed</th>
<th>CTS normal</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>126,457</td>
<td>552</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Plan Name</th>
<th>Pass</th>
<th>Fail</th>
<th>Not Executed</th>
<th>CTS NORAX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>126,457</td>
<td>552</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation - embedded data identification

- ground truth: compiled with debugging sections (dwarf .debug_*)
- very few gadgets in extracted inline data

<table>
<thead>
<tr>
<th>Module</th>
<th>#. of Real Inline Data (Byte)</th>
<th>#. of Inline Data Flagged by Norax (Byte)</th>
<th>#. of Gadgets found in extracted inline Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>toybox</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>toolbox</td>
<td>20</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>dhcpcd</td>
<td>40</td>
<td>40</td>
<td>4</td>
</tr>
<tr>
<td>Logd</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>installd</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>app_process64</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(zygote)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>qseecomd</td>
<td>N/A</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>surfacetflinger</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>rild</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>libart.so</td>
<td>17716</td>
<td>17716</td>
<td>8</td>
</tr>
<tr>
<td>libstagefright.so</td>
<td>296</td>
<td>296</td>
<td>5</td>
</tr>
<tr>
<td>libcrypto.so</td>
<td>2472</td>
<td>2512</td>
<td>25</td>
</tr>
<tr>
<td>libmedia.so</td>
<td>3936</td>
<td>3936</td>
<td>0</td>
</tr>
<tr>
<td>libc.so</td>
<td>4836</td>
<td>4836</td>
<td>5</td>
</tr>
<tr>
<td>libc++.so</td>
<td>12</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>libsqlite.so</td>
<td>932</td>
<td>1004</td>
<td>13</td>
</tr>
<tr>
<td>libbinder.so</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>libm.so</td>
<td>20283</td>
<td>20291</td>
<td>48</td>
</tr>
<tr>
<td>libandroid.so</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>50551</strong></td>
<td><strong>50671</strong></td>
<td><strong>108</strong></td>
</tr>
</tbody>
</table>
**Evaluation - performance**

- **average performance overhead:** 1.18%
- **average memory overhead:** 2.21%
The address of next instruction after bl is stored on stack and visible to attacker
Function pointer or function address in .got are visible to attacker

#include <stdio.h>
void foo(void)
{
    printf("Hello World!\n");
}
int main(int argc, char **argv)
{
    foo();
    return 0;
}
Interesting Paper/Link

- ROPgadget: https://github.com/JonathanSalwan/ROPgadget
- Control Flow Integrity for COTS Binaries. USENIX Security 2013
- SoK: Eternal War in Memory. IEEE S & P 2013
- http://shell-storm.org
- Control-Flow Integrity. CCS 2005
Shellcode injection and execution are not prerequisite for ROP
Shellcode injection and execution are not prerequisite for ROP
Fine-grained ASLR cannot defend JIT-ROP attack
Take-Home Message

- Shellcode injection and execution are not prerequisite for ROP
- Fine-grained ASLR cannot defend JIT-ROP attack
- Direct memory disclosure and indirect memory disclosure
Take-Home Message

- Shellcode injection and execution are not prerequisite for ROP
- Fine-grained ASLR cannot defend JIT-ROP attack
- Direct memory disclosure and indirect memory disclosure
- XOM is supported by Intel EPT and AArch64 userspace
Take-Home Message

- Shellcode injection and execution are not prerequisite for ROP
- Fine-grained ASLR cannot defend JIT-ROP attack
- Direct memory disclosure and indirect memory disclosure
- XOM is supported by Intel EPT and AArch64 userspace
- Code-data separation is possible for AArch64 COTS binary