Adelie: Continuous Address Space Layout Re-randomization for Linux Drivers

Ruslan Nikolaev, Hassan Nadeem, Cathlyn Stone, Binoy Ravindran
Security vulnerabilities in OSs continues to rise

Number of CVEs for device drivers

- Linux
- Windows
Attacks

• Control-flow attacks
• Return-Oriented Programming (ROP)
  − ASLR mitigates against traditional ROP
  − More elaborate ROP attacks are still possible
  − KASLR is limited
Contributions

- Extend KASLR support in Linux
- Implement stack re-randomization, address encryption, and continuous ASLR on Linux modules
Goals

**Generality:** Transform all modules to the 64-bit KASLR model

**Performance:** Avoid costs of copying code and data (re-randomization)

**Entropy:** Kernel modules can be any distance apart from each other

**Security:** Protect against code reuse attacks
Extending KASLR

- We use a preliminary PIE patch for the Linux kernel
- Cannot use PIE for kernel modules
- We use a more general PIC model for modules, which is similar to shared libraries (with GOT and PLT support)
- Extends KASLR to 64 bits
- Avoids costly absolute-address models such as mcmodel=large
Extending KASLR

- Compilers rely on procedure linkage tables (PLT) and global offset tables (GOT) to call external functions and retrieve external addresses
- We use these to support multiple mappings to code during ongoing re-randomization

```
Code
  call f@plt

PLT
  mov r, f@got
  jmp r

GOT
  0xXX..XX
```
Optimizations

- Spectre-V2
  - Affects indirect calls
  - Impacts the PIC model
- Optimizations are crucial

* The picture is taken from Wikipedia
Continuous Module Re-Randomization

64-bit Virtual Address Space

Kernel

Creating new mappings

Unmapping old regions

1

2

3
Continuous Module Re-Randomization

Use a zero-copying mechanism and organize modules into movable and immovable parts.

updated during rerandomization
Continuous Module Re-Randomization

- Use **delayed unmapping** to control address space lifetime
- Track pending calls in a scalable manner with as little overhead as possible
- Enclose operations that access potentially disappearing memory blocks with calls to `mr_start` and `mr_finish`
Continuous Module Re-Randomization

Wrap externally accessible functions in re-randomizable modules, continuously re-randomize stacks

```c
long func(long arg) { ... }

long func_real(long arg) { ... } // Renamed function

long func(long arg) {  
    mr_start();
    get_new_stack();
    long ret = func_real(arg);
    return_old_stack();
    mr_finish();
    return ret;
}

kernel_ref(&func);
```
Continuous Module Re-Randomization

Wrap externally accessible functions in re-randomizable modules, continuously re-randomize stacks

```
get_new_stack (wrapper):
%rbp = %rsp;  // save stack
stk = pop_stack_this_cpu();
if (!stk) stk = alloc_stack();
%rsp = stk;

return_old_stack (wrapper):
stk = %rsp;
%rsp = %rbp;  // restore stack
push_stack_this_cpu(stk);

prologue/epilogue (non-static):
mov key@GOTPCREL(%rip), %r11
xor %r11, (%rsp)  // en/decrypt
xor %r11, %r11  // %r11 = 0

prologue/epilogue (static):
push %rbp
mov key@GOTPCREL(%rip), %rbp
xor %rbp, 8(%rsp)  // en/decrypt
pop %rbp
```
# Evaluation

<table>
<thead>
<tr>
<th></th>
<th>Server (for Evaluation)</th>
<th>Load Generator</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUs</td>
<td>Xeon Silver 4114 2.20GHz</td>
<td>Core i7 4770 3.40GHz</td>
</tr>
<tr>
<td>Cores</td>
<td>2x10, no HyperThreading</td>
<td>1x4, no HyperThreading</td>
</tr>
<tr>
<td>L1/L2 cache</td>
<td>64 / 1024 KB per core</td>
<td>64 / 256 KB per core</td>
</tr>
<tr>
<td>L3 cache</td>
<td>14080 KB</td>
<td>8192 KB</td>
</tr>
<tr>
<td>Memory</td>
<td>96 GB</td>
<td>16 GB</td>
</tr>
<tr>
<td>Network</td>
<td>Intel E1000E 1GbE</td>
<td>Intel E1000E 1GbE</td>
</tr>
<tr>
<td>Storage</td>
<td>Samsung 970 EVO NVMe</td>
<td>Samsung 860 EVO SSD</td>
</tr>
<tr>
<td>USB 3.0</td>
<td>Intel C620 xHCI</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Evaluation

NVMe microbenchmark
**Evaluation**

MySQL performance evaluation under different concurrency levels and transaction speeds.

- **Transctions / s** vs **Concurrency**
  - Linux: 5 ms
  - 5 ms
  - 1 ms

- **CPU Usage (%)** vs **Concurrency**
  - Linux: 5 ms
  - 5 ms
  - 1 ms
Evaluation

Apache
Evaluation

Extreme ioctl test

Stack Re-randomization:
- False
- True

Throughput (M/s)

<table>
<thead>
<tr>
<th></th>
<th>Linux</th>
<th>NoRand</th>
<th>5 ms</th>
<th>1 ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>True</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CPU Usage (%)

<table>
<thead>
<tr>
<th></th>
<th>Linux</th>
<th>NoRand</th>
<th>5 ms</th>
<th>1 ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>True</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Thank you!

<table>
<thead>
<tr>
<th>Name</th>
<th>Email</th>
<th>Institution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ruslan Nikolaev</td>
<td><a href="mailto:nikola@psu.edu">nikola@psu.edu</a></td>
<td>The Pennsylvania State University</td>
</tr>
<tr>
<td>Hassan Nadeem</td>
<td><a href="mailto:hnadeem@vt.edu">hnadeem@vt.edu</a></td>
<td>Virginia Tech</td>
</tr>
<tr>
<td>Cathlyn Stone</td>
<td><a href="mailto:stonecat@vt.edu">stonecat@vt.edu</a></td>
<td>Virginia Tech</td>
</tr>
<tr>
<td>Binoy Ravindran</td>
<td><a href="mailto:binoy@vt.edu">binoy@vt.edu</a></td>
<td>Virginia Tech</td>
</tr>
</tbody>
</table>

Adelie’s source code is available at: [https://github.com/adelie-kaslr](https://github.com/adelie-kaslr)