Reinventing the Privilege Drop


Ira Ray Jenkins¹, Sergey Bratus¹, Sean Smith¹, Maxwell Koo²

¹Dartmouth College
²Narf Industries
Overview

“Exploits are proofs-by-construction to the discrepancies present between a programmer’s intent and a program’s actual behavior.”

Problem

The standard process model lets all code touch all data in the address space.

Goal

Preserve programmer intent for intra-process relationships between code and data.
Code to Process

• Code compiled into standard file format:
  • *nix - Executable and Linkable Format (ELF)
  • Windows - Portable Executable (PE)
  • OSX/iOS - MACH Object (MACH-O)

• Executable and Linkable Format (ELF) files contain:
  • Code and data for a given executable, as well as metadata necessary for the creation of a process address space.
Sections & Segments

• Sections define the semantically distinct units of a program.
  • .text, .data, .bss, etc...

• Segments are groupings of sections.
  • Segments are loaded at runtime into the process address space.
  • Segments define the permissions of memory sections.

Programmer intent is discarded in the packing of sections into segments!
ELF-based Access Control

“Sections are types, linking is policy”

Reclaim the programmer intent discarded by a “forgetful” loader.

• Code is annotated, compiled, and linked with ELFbac policy.

• An “unforgettable”, ELFbac-aware, loader builds the process address space with the policy, creating the desired isolation.

• An ELFbac-aware kernel enforces the policy during runtime.
  • Intent is enforced via page tables & page faults.
ELFbac Policy Creation

• Policy as a Finite State Machine:
  • States define phases of program execution driven by a given section of code, e.g., input parsing.
  • Transitions between states are achieved via memory accesses and function calls.

• Policy is defined via linker scripts.
  • Semantic policies, e.g., “input data can only be read by parsing functions”

• Code is annotated to use the policy via compiler pragmas:
  • `__attribute__((section(". inputs")))` int debug_flag = 0;

```
"states": [
{"name": "Parse",
 "sections": [{
 "name": "inputs",
 "description": "*(. .data.parse)", "flags": "rw "}],
{"name": "Process",
 "sections": [{
 "description": "*main.o(.data.parse_result)", "flags": "rw ""]}
"call_transitions": [ 
 {"from": "Parse",
  "to": "Process",
  "address": "GoToProcess()" }]
```
ELFbac Policy Enforcement

• Replace the kernel’s view of a process’ virtual memory context with a diversified collection of “shadow” contexts, each representing a single policy state.
  • Each shadow context only maps those regions of memory that can be accessed in the current state according to the policy.
  • Achieved through Page Tables and Virtual Memory Mappings.

• Policy violations (unintended memory accesses or function calls) are trapped, leading to error handling code or ultimately a segmentation fault.
Process View vs. Kernel View

Virtual Address Space

Physical Address Space

Virtual Address Space

Physical Address Space

Page belonging to process
Page NOT belonging to process

Page belonging to State 1
Page belonging to State 2
Page NOT belonging to process
Case Study: OpenSSH

• Most popular implementation of the Secure Shell (SSH) network protocols.
  • Used to securely connect to and manage remote devices.
Roaming in OpenSSH

- In 2010, the OpenSSH client introduced an experimental and undocumented “roaming” feature.
  - The purpose of roaming was to allow the resumption of suspended sessions, e.g., in the case of unexpected network termination.
- In 2016, CVE-2016-0777 disclosed an information leak present in the implementation of OpenSSH’s roaming feature.
Mitigating the Roaming Bug

Use ELFbac to isolate the cryptographic keys and the roaming buffer.

In total, 27 annotations in 4 files were all that was necessary to achieve the critical isolation.
Execution with ELFbac Mitigation

Client

<table>
<thead>
<tr>
<th>Crypto heap</th>
<th>Establish connection.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet heap</td>
<td>key-exchange: <a href="mailto:resume@appgate.com">resume@appgate.com</a></td>
</tr>
<tr>
<td>Roaming heap</td>
<td>Disconnect.</td>
</tr>
<tr>
<td>Default heap</td>
<td>Reconnect.</td>
</tr>
<tr>
<td></td>
<td>“Send me everything.”</td>
</tr>
<tr>
<td></td>
<td>Roaming messages (no SSH keys).</td>
</tr>
</tbody>
</table>

Server
Conclusions

• Programmer intent is a crucial part of software security.
• ELFbac allows a programmer to codify intent into enforceable policy.
• Were ELFbac to have been used in OpenSSH, this bug would never have occurred.
• ELFbac is as flexible and robust as a software’s modularity.
  • More modular -> More easily isolated

“Vulnerabilities arise as a result of a mismatch between a programmer’s mental model of software and the reality that exists when computation is performed in the real world.”
Future Work

• Policy creation relies on codebase familiarity and intuition...
  • Best done at design time, but still requires a paradigm shift.
• Performance can be a problem...
  • More states -> More transitions -> More overhead
• Multiple policies in a single executable?
  • Can policies interact? What weird machines lurk here...?
• Mitigating Meltdown?
  • Maybe... But no intention of porting ELFbac to the kernel.
• Mitigating Spectre?
  • YES! Coming soon!
Thanks!

Ira Ray Jenkins
jenkins@cs.dartmouth.edu

Code available at:
https://github.com/sergeybratus/elfbac-arm

This material is based upon work supported by the Department of Energy under Award Number: DE-OE0000780. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
Images

- https://thehackernews.com/2016/10/sshowdown-iot-security.html
- https://www.shodan.io/report/jaGB3De1