Mitigating and Preventing Vulnerabilities with ELFbac

Ira Ray Jenkins, Dartmouth College
Code to Process

- **source code**
  - **hello.c**

- **Compiler (gcc)**
  - **hello.o**

- **Static linker (ld)**
  - **hello**

- **Runtime linker/loader (ld.so)**
  - **libc.so**

- **ELF relocatable objects**

- **ELF executable**

- **Running process**

- **Common object file formats**
  - *nix -> Executable and Linkable Format (ELF)
  - Windows -> Portable Executable (PE)
  - OSX/iOS -> MACH Object (MACH-O)
Sections & Segments

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

• Sections contain the code and data of a program.
  • Each section defines semantically distinct units of code and data

• 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

- Goal: 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
ELFbac Policy Creation

- Policy is as a Finite State Machine.
  - States define a particular abstract phase of program execution driven by a given section of code, e.g., input parsing, network code, or cryptographic code
  - Transitions between states are achieved via memory accesses ("data transitions") and function calls ("call transitions")

- ELFbac policy is defined via linker scripts in simple JSON.
  - Defining custom sections, their access controls, and any intersectional relationships
  - 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;`

```json
"name": "Parse",
"sections": [{
  "name": "inputs",
  "description": "*(.data.secret)",
  "flags": "rw"
}]

"call_transitions": [ {
  "from": "Parse",
  "to": "Calculate",
  "address": "GoToCalculate()"
}]
```
ELFbac Policy Enforcement

• Replaces 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.
OpenSSH is Ubiquitous

- Most popular implementation of the Secure Shell (SSH) network protocols
- Used to securely connect to and manage remote devices

"The company believes that its optional access to the Linux operating system through a secure shell (SSH) will be of particular interest to OEMs."
Roaming in OpenSSH

• In version 5.4, released 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

- Goal: Use ELFbac to isolate the memory regions used to store 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 Mitigation

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crypto heap</td>
<td>Establish connection.</td>
</tr>
<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>
Demo
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

Future Work

• Policy creation relies largely on codebase familiarity and intuition...
• Performance can be a problem...
• Multiple policies in a single executable...
• Where does ELFbac fit with the IoT and ICS...
• Mitigating Spectre...

Thanks!
References

- https://memegenerator.net/instance/81422724
- https://ics-cert.us-cert.gov/tips/CSAR-10-114-01
- https://ics-cert.us-cert.gov/alerts/ICS-ALERT-12-034-01
- https://thehackernews.com/2016/10/sshowdown-iot-security.html
- https://www.shodan.io/report/jaGB3De1
- https://commons.wikimedia.org/wiki/File:Document_text.svg
- https://openclipart.org/detail/275692/icon-book
- http://www.clker.com/cliparts/5/j/m/Z/s/z/runrunrun-md.png
- https://openclipart.org/detail/256083/gears
- https://en.wikipedia.org/wiki/Virtual_address_space#/media/File:Virtual_address_space_and_physical_address_space_relationship.svg