Post Compilation Binary Protection

Dr. Eric Schulte
eschulte@grammatech.com
November 3, 2021
Agenda

Introduction / Context / Impact

Technical Details

Real-World Examples

Demo

Hands On Exercises
Introduction / Context / Impact
GrammaTech

Company Info

Since 1989
Employees 100+ total
20+ w/PhD.
Location Bethesda, MD
Ithaca, NY
Remote, USA
Sponsors DARPA, ONR, DHS, ARMY, AIRFORCE

Areas of Expertise

Source Analysis SAST, DAST
Source Generation/Adaptation
Binary Analysis
Binary Hardening/Transformation

Commercial Products

CodeSonar Static Analysis
CodeSentry Binary N-day vulnerability detection
DevSecOps

Secure software development

Tools

- CodeSonar Static Analysis
- CodeSentry Binary N-day detection
- Proteus Vulnerability Discovery and Remediation
- REAFFIRM Reverse Engineer, Analyze, and Fix Firmware
DevSecOps

Secure software development

Tools

CodeSonar  Static Analysis

CodeSonar Static Analysis

CodeSentry Binary N-day detection

Proteus Vulnerability Discovery and Remediation

REAFFIRM Reverse Engineer, Analyze, and Fix Firmware

© 2021 GrammaTech, Inc. All rights reserved.
DevSecOps

Secure software development

Tools
- **CodeSonar**  Static Analysis
- **CodeSentry**  Binary N-day detection

Diagram:
- CodeSonar
- CodeSentry
- Code
- Analyze
- Warn
- Build
- Scan
- CI
- Test
- Plan
- CD
- Deploy
- Operate
- Measure
- Release
DevSecOps

Secure software development

Tools
- **CodeSonar**: Static Analysis
- **CodeSentry**: Binary N-day detection
- **Proteus**: Vulnerability Discovery and Remediation

Tools:
- **CodeSonar**: Static Analysis
- **CodeSentry**: Binary N-day detection
- **Proteus**: Vulnerability Discovery and Remediation

DIAGRAM:
- CodeSonar
- CodeSentry
- Proteus
- CI
- CD
- Build
- Test
- Measure
- Plan
- Release
- Deploy
- Operate
- Analyze
- Warn
- Scan
- Proteus Remediation
- Proteus Discovery

© 2021 GrammaTech, Inc. All right reserved.
DevSecOps

Secure software development

Tools

**CodeSonar**
- Static Analysis

**CodeSentry**
- Binary N-day detection

**Proteus**
- Vulnerability Discovery and Remediation

**REAFFIRM**
- Reverse Engineer, Analyze, and Fix Firmware

Diagram:
- CI (Code) builds Code, which is analyzed by CodeSonar and CodeSentry.
- Test results are used to plan the release.
- Plan leads to deployment, and deployed Code is operated.
- Measure feedback is used to improve future releases.

- CodeSentry Scan and Warn cycle back to CI for further analysis.
- Proteus Discovery and Remediation cycle back to Operate for reverse engineering and analysis.
- REAFFIRM Fix and Reverse cycle back to Plan for improved risks and vulnerabilities.
DevSecOps

Automated binary hardening

Post Compilation Software Protection
- Binary Consolidation
- Binary Debloating
- Binary Hardening

Environments

Systems
- Linux, Windows, Embedded

ISAs
- Intel, ARM, MIPS
DevSecOps

Automated binary hardening

Environments

Systems
Linux, Windows, Embedded

ISAs
Intel, ARM, MIPS

GRAMMATECH, INC. CONFIDENTIAL/PROPRIETARY INFORMATION
Distribution to third parties is prohibited without the written confirmation of GrammaTech, Inc. 6/25
© 2021 GrammaTech, Inc. All right reserved.
GrammaTech’s Binary Rewriting

Capabilities
▶ Zero Overhead
▶ Impeccable disassembly
▶ Commercial-grade tooling

Use Cases
▶ Binary Consolidation
▶ Binary Debloating
▶ Binary Hardening
▶ Automated Diversity

Lift

Assemble

IR (1)

IPCFG (1)

AuxData (N)
ID1 DATA1
ID2 DATA2
ID3 DATA3
ID4 DATA4

Modules (N)

Symbols (N)

Sections (N)

Edges (N)

Blocks (N)

Proxy blocks (N)

GTIRB

SymbolicExpressions (N)

something
GrammaTech’s Binary Rewriting

Open Source
► Open R&D ecosystem
► Global contributors
► Improved usability, accuracy, security

IR (1)
AuxData (N)
DATA1 ID1
DATA2 ID2
DATA3 ID3
DATA4 ID4

IPCFG (1)
Edges (N)
Proxy blocks (N)
Blocks (N)
SymbolicExpressions (N)

Lift
Assemble

ddisasm
gtirb
pprinter

GRAMMATECH, INC. CONFIDENTIAL/PROPRIETARY INFORMATION
Distribution to third parties is prohibited without the written confirmation of GrammaTech, Inc. © 2021 GrammaTech, Inc. All right reserved.
GTIRB vs. LLVM

Why not use LLVM as a Binary IR.

**LLVM Strengths**
- Huge community
- C/C++ compilers
- Industry standard
- Many optimization passes
- Many hardening passes
- Many analysis passes

**LLVM Weakness**
- Representation
  - Typed data – too difficult to lift
  - SSA code – loses information
- Rewriting
  - Represent stack and memory as byte arrays
  - Emulated stack
    - Bulky binaries
    - Baroque binaries
    - Limits applicability of LLVM passes
## Binary Rewriting Supported Systems

<table>
<thead>
<tr>
<th>ISAs</th>
<th>Intel-64</th>
<th>Intel-32</th>
<th>ARM-64</th>
<th>ARM-32</th>
<th>MIPS-64</th>
<th>MIPS-32</th>
<th>PPC-64</th>
<th>PPC-32</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddisasm</td>
<td>✓</td>
<td>✓</td>
<td>✓*</td>
<td>✓</td>
<td>✗</td>
<td>✓*</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

### Legend
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>Complete</td>
</tr>
<tr>
<td>✓*</td>
<td>Complete (non-public)</td>
</tr>
<tr>
<td>✗</td>
<td>Partial</td>
</tr>
<tr>
<td>✗</td>
<td>Not yet covered</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OSs</th>
<th>Linux</th>
<th>Windows</th>
<th>Mac-OS</th>
<th>Firmware</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddisasm</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓*</td>
</tr>
</tbody>
</table>

### Features

<table>
<thead>
<tr>
<th>Features</th>
<th>w/ &amp; w/o Symbols</th>
<th>w/ &amp; w/o Relocations</th>
<th>TLS (Multi-threaded)</th>
<th>Debug</th>
<th>Exceptions</th>
<th>PE Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>ddisasm</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>
Protective binary transformations

Requirements
- Binaries
- CI/CD Integration
- Supported ISA
- Supported OS

Benefits
- Deployment
- Administration
- Efficiency
- Security
Modern software dependencies
Statically link dependencies

Before: curl executable dynamically linked with libc and libcrypt

After: curl executable statically linked with libc and libcrypt

Benefits

▶ Easier deployment
▶ Easier system administration
▶ No “dll hell”
▶ Fewer system requirements
▶ Control all libraries
▶ No dynamic code loading
▶ Downstream processing applies to all code
Reduce attack surface (1/2)

Remove code not transitively reachable from entry point(s).

Benefits

- **Conservatively** remove only unreachable code
- Reduced attack surface
  - fewer bugs
  - fewer 0-days
  - fewer N-days
- Reduced file size
  - smaller packages
  - smaller on disk
  - smaller in memory
Reduce attack surface (2/2)

Remove code not exercised in a block-level execution trace of the test suite.

Benefits

- Aggressively remove all code not needed by tests
- Reduced attack surface
  - fewer bugs
  - fewer 0-days
  - fewer N-days
- Reduced file size
  - smaller packages
  - smaller on disk
  - smaller in memory
Harden control flow (1/2)

Hard code indirect (open-ended) branches.

Benefits

- Harder for attackers to turn bugs into exploits
- Protect against:
  - control flow hijack attacks
  - code reuse attacks
Encrypt return values on the stack.

Benefits

- Harder for attackers to turn bugs into exploits
- Protect against:
  - control flow hijack attacks
  - code reuse attacks
Before

```
call *%r11
```

After

```
// After
call set_up_target;
capture_spec:
  pause;
  jmp capture_spec;
set_up_target:
  mov %r11, (%rsp);
ret;
```

Benefits

- Protect against Spectre (data leak)
- Apply modern compiler protections to legacy binaries

Adapted from https://support.google.com/faqs/answer/7625886.
Diversify

Shuffle layout of code inside the text section. Similar to a fine-grained ASLR.

Benefits

▶ Diversified attack surface across instances.
▶ Payloads don’t generalize across instances.
▶ Limit utility of captured binaries.
CI Integration

X-Forms

1. shuffle
2. stack-stamp
3. block-trace → profile-viz
4. reachable → reduce
5. to-static → *

CI Integration

- gtirb-server
- gtirb-client or curl (https://pypi.org/project/gtirb-client/)
CentOS

Experience:

1. Hardened /bin in CentOS:8 Docker image
   - 320 exes: gcc, git, make, python, ssh, vim, ...
2. In use as development environment
   - No noticeable slowdowns or instability

Take away:

- Reliable tooling
- Efficient results
- Portable tools
Experience:

1. Integrating X-forms into our own CI
2. Config is easier than expected
3. So far so good but very early...

Take away:

▶ CI usage works well
▶ Transform fits between build and test
▶ Deployment is simple

Examples

▶ https://git.grammatech.com/benchmark/quagga/-/pipelines/422352
▶ https://git.grammatech.com/rewriting/gtirb-pprinter/-/pipelines/422360
Demo

https://git.grammatech.com/benchmark/quagga/-/pipelines/422352
Hands On Exercises
Thanks

eschulte@grammatech.com
research@grammatech.com
CodeSentry

National Vulnerability Database

User Binary

Extract Semantic Features

Match components

Heartbleed

glibc CVE

Vulnerability Report
Reaffirm

Reaffirm Input - Binary Firmware
- Installer
- Memory dump
- Network capture
- Binary to flash

Analysis
1. Intermediate representation
2. Static Analysis
   - Reports
     - Potential 0-day vulnerabilities
3. Software Composition Analysis
   - Find uses of known software e.g. open source, via similarity matching
4. Component Extraction and Harnessing
   - Find and wrap extractable subcomponents
   - Generate harness to remove hardware dependencies
5. Rehosted (harnessed) components

Testing
6. Component Testing
   - Test in Linux environment
   - Use state-of-the-art fuzzing approaches

ReaFIRM Outputs
- Security assessments
- Patched firmware

Assessment and Patching
6. Manual Cybersecurity Assessment
   - Supporting dashboard shows:
     - Analysis/testing reports
     - Visualization of firmware components

7. Patching
   - Describe small patches in assembly or C
   - Apply edits back to original binary

Automated and semi-automated workflow steps captured in a repeatable, human- and machine-readable manner

Relevant technologies (GrammaTech-developed underlined)
1) CodeSurfer®, Ghidra
2) CodeSonar®
3) DISCOVER
4) GTx
5) QEMU, AFL
6) PANORAMA
7) Scalpel
8) Jupyter, Papermill