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

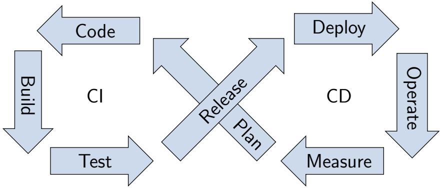
Commercial Products

CodeSonar	Static Analysis
CodeSentry	Binary N-day vulnerability detection

DevSecOps

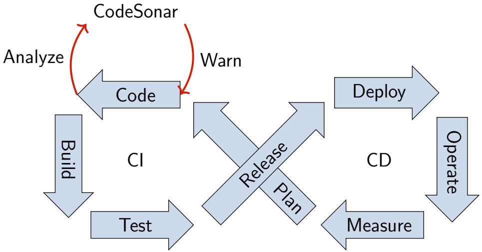
Secure software development

Tools



DevSecOps

Secure software development

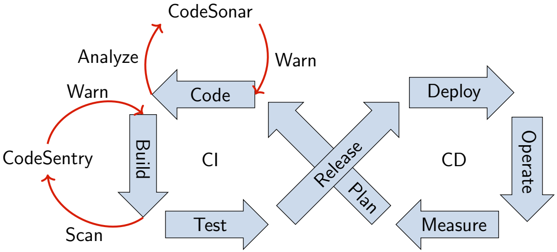


Tools

CodeSonar Static Analysis

DevSecOps

Secure software development

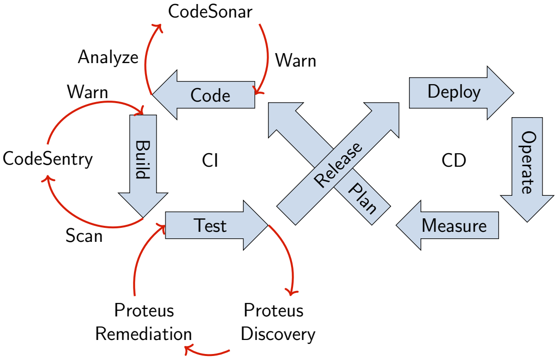


Tools

CodeSonar	Static Analysis
CodeSentry	Binary N-day detection

DevSecOps

Secure software development

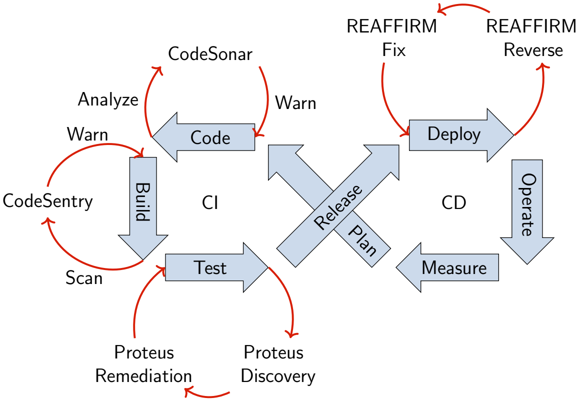


Tools

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

DevSecOps

Secure software development

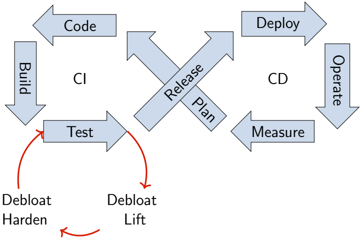


Tools

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

DevSecOps

Automated binary hardening



Post Compilation Software Protection

- Binary Consolidation
- Binary Debloating
- Binary Hardening

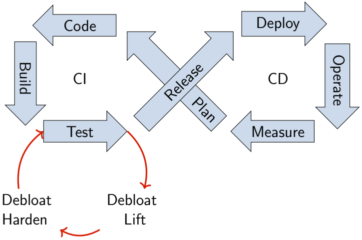
Environments

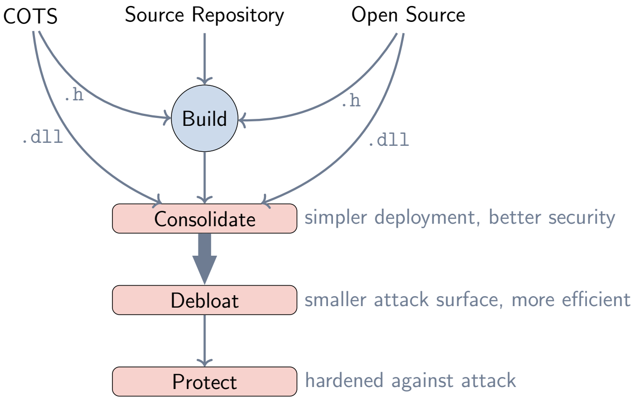
Systems
Linux, Windows, Embedded

ISAs Intel, ARM, MIPS

DevSecOps

Automated binary hardening





Environments

Systems

Linux, Windows, Embedded

ISAs Intel, ARM, MIPS

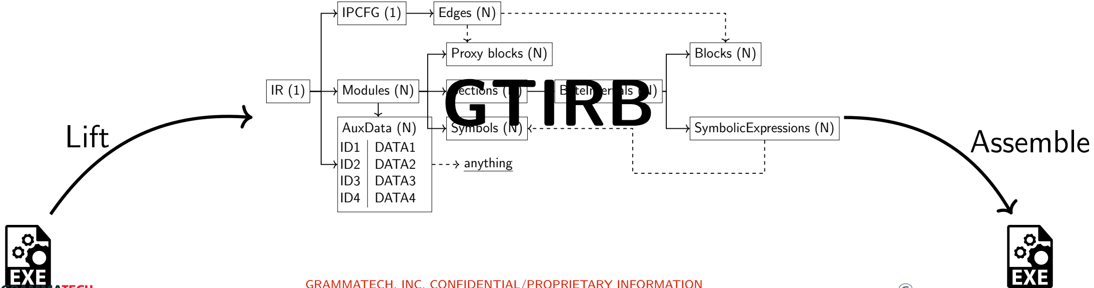
GrammaTech's Binary Rewriting

Capabilities

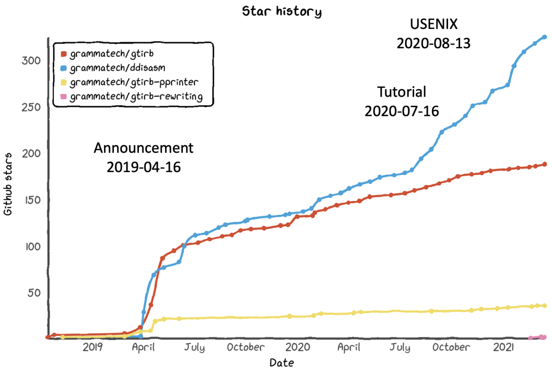
- Zero Overhead
- Impeccable disassembly
- Commercial-grade tooling

Use Cases

- Binary Consolidation
- Binary Debloating
- Binary Hardening
- Automated Diversity

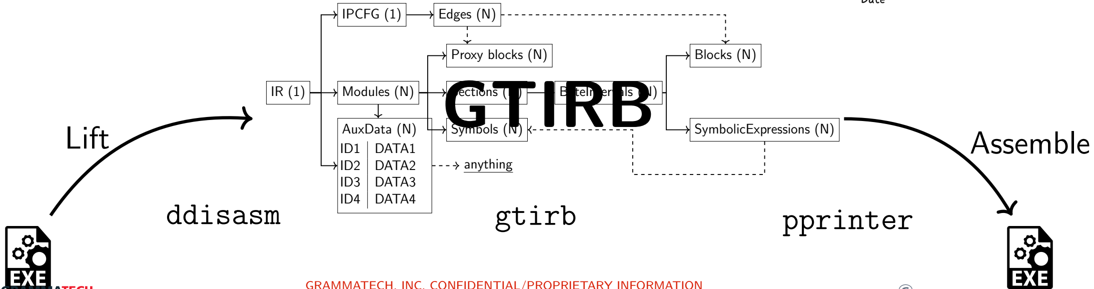


GrammaTech's Binary Rewriting



Open Source

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



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

ISAs

	Intel-64	Intel-32	ARM-64	ARM-32	MIPS-64	MIPS-32	PPC-64	PPC-32
ddisasm	✓	✓	✓*	✓	✗	✓*	✗	✗

Legend

Symbol	Meaning
✓	Complete
✓*	Complete (non-public)
✢	Partial
✗	Not yet covered

OSs

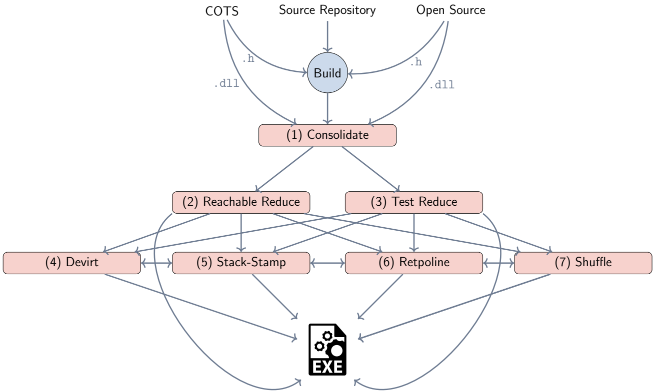
	Linux	Windows	Mac-OS	Firmware
ddisasm	✓	✓	✗	✓*

Features

	w/ & w/o Symbols	w/ & w/o Relocations	TLS (Multi-threaded)	Debug	Exceptions	PE Resources
ddisasm	✓	✓	✓	✢	✢	✓

Technical Details

Protective binary transformations



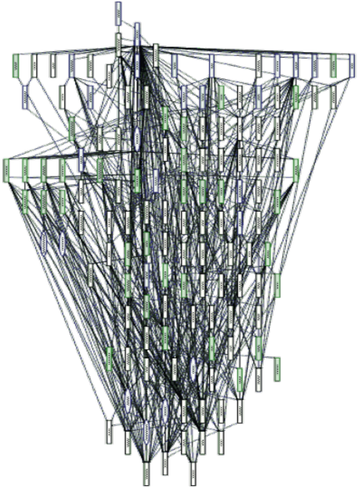
Requirements

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

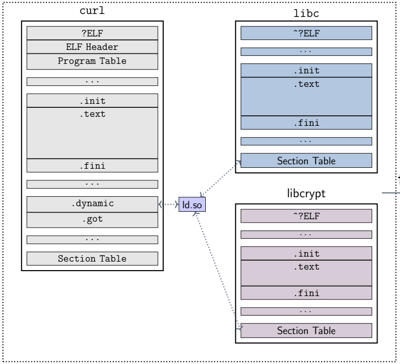
Benefits

- Deployment
- Administration
- Efficiency
- Security

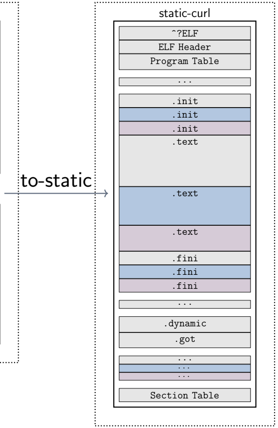
Modern software dependencies



Statically link dependencies



Before: curl executable dynamically linked with libc and libcrypt



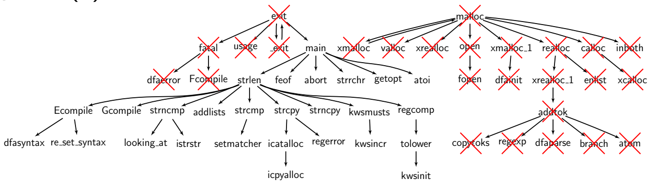
After: curl executable statically linked with libc and libcrypt

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).

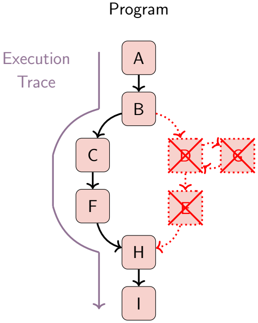


Benefits

- **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.

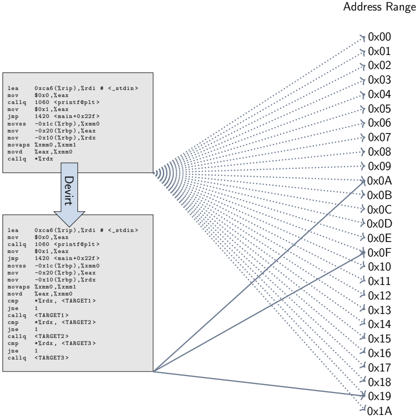


Benefits

- **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.

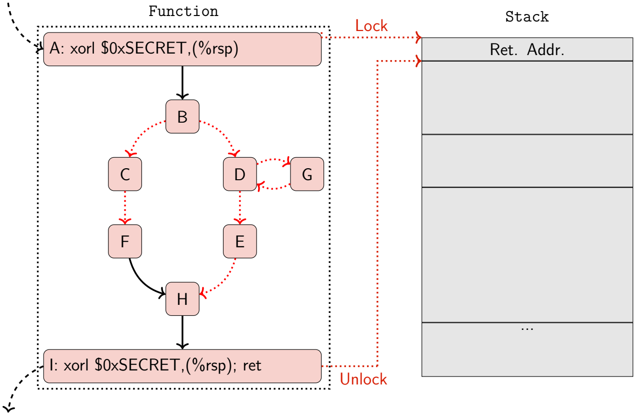


Benefits

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

Harden control flow (2/2)

Encrypt return values on the stack.



Benefits

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

Retpoline

```
// Before
call *%r11
```

→

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

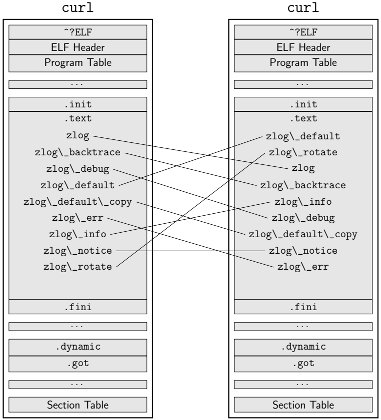
Benefits

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

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

Diversify

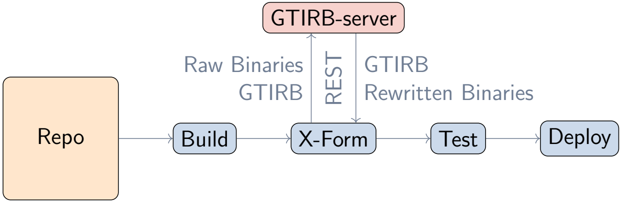
Shuffle layout of code inside the text section.
Similar to a fine-grained ASLR.



Benefits

- Diversified attack surface across instances.
- Payloads don't generalize across instances.
- Limit utility of captured binaries.

CI Integration




```
xform:
  services:
    - name: docker.grammatech.com/rewriting/gtirb-server:service-8080
      alias: gtirb-server
  script:
    - apt-get update -y
    - apt-get install -y gtirb-pprinter
    - for binary in $BINARIES ; do
        curl -F transform=ddisasm -F input=@$binary \
             --output $binary.gtirb http://gtirb-server:8080/simple
        curl -F transform=ss -F input=@$binary.gtirb \
             --output $binary.ss.gtirb http://gtirb-server:8080/simple
        gtirb-pprinter --ir $binary.ss.gtirb \
                       --binary $binary.shuffle.ss
      done
```

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/)

Real-World Examples

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

Internal Dogfooding

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

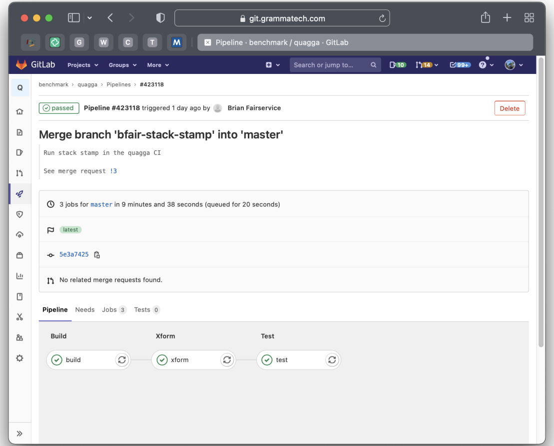
Examples

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

Demo

Demo

https://git.grammatech.com/benchmark/quagga/-/pipelines/422352

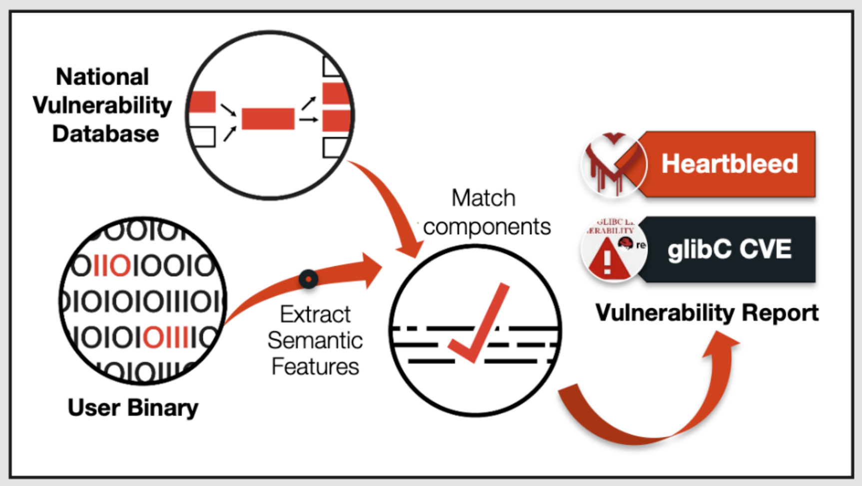


Hands On Exercises

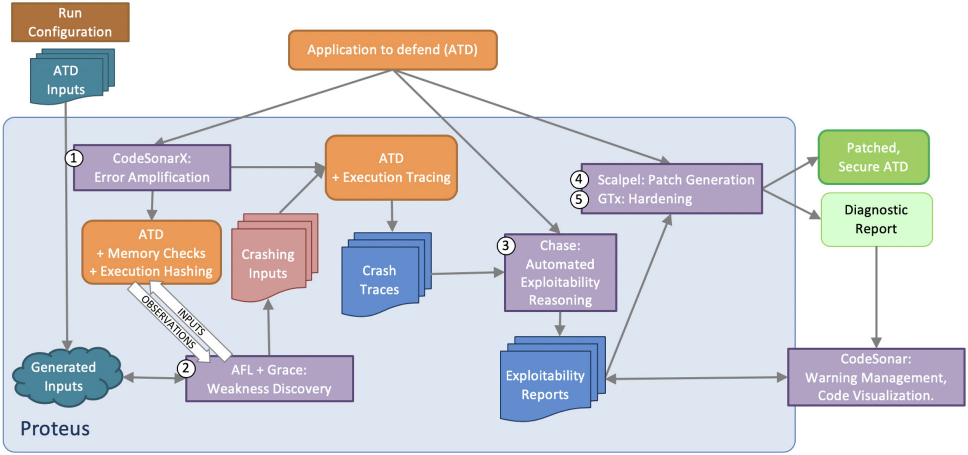
Thanks

eschulte@grammatech.com
research@grammatech.com

CodeSentry

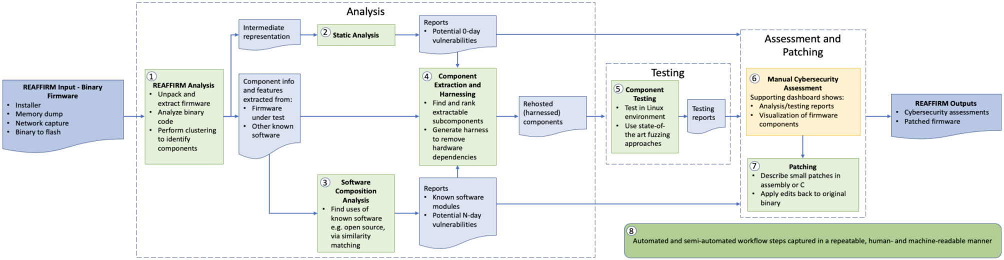

National
Vulnerability
Database
User Binary
Extract
Semantic
Features
Match
components
Heartbleed
glibC CVE
Vulnerability Report

Proteus



Run Configuration
ATD Inputs
Application to defend (ATD)
1 CodeSonarX: Error Amplification
ATD + Execution Tracing
4 Scalpel: Patch Generation
5 GTx: Hardening
Patched, Secure ATD
Diagnostic Report
ATD + Memory Checks + Execution Hashing
Crashing Inputs
Crash Traces
3 Chase: Automated Exploitability Reasoning
OBSERVATIONS
INPUTS
Generated Inputs
2 AFL + Grace: Weakness Discovery
Exploitability Reports
CodeSonar: Warning Management, Code Visualization.
Proteus

Reaffirm



Relevant technologies
(GrammaTech-developed underlined)

1) CodeSurfer®, Ghidra
2) CodeSonar®
3) DISCOVER
4) GTx
5) QEMU, AFL
6) PANORAMA
7) Scalpel
8) Jupyter, Papermill