GRAMMATECH

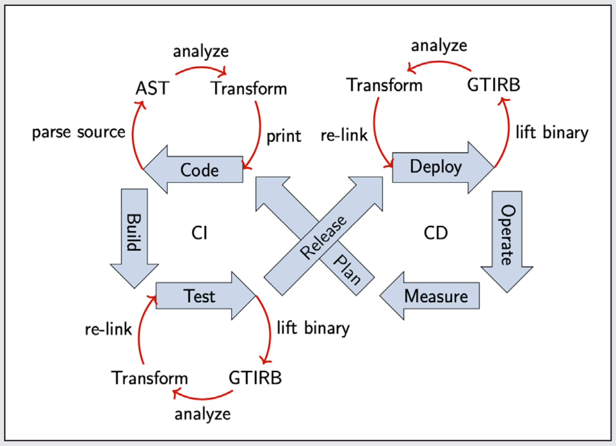
PROGRAM PROTECTION THROUGH REDUCTION, HARDENING, AND DIVERSIFICATION

Binary Transformation
Monitoring and Hardening
Runtime Application Self-Protection

Design
Development
✓ **Pre-Deployment**
✓ **Deployment**
✓ **Sustainment**

Under ONR's TPCP program GrammaTech has developed revolutionary tools for automated software protection through reduction, hardening, and diversification. In addition to these mature commercial tools for end-users GrammaTech has released the supporting infrastructure as open-source software. By sharing this technology with the research community we hope to inspire radical change in capabilities for both developers and end users to modify and protect new, legacy, and third-party software.

GTIRB	GrammaTech Intermediate Representation for Binaries.
DDisasm	A fast and accurate reassemblable disassembler.
To-static	Converts a dynamically linked binary executable to one that is statically linked.
Reduce	Removes specified features from a binary.
Stack-stamp	Applies 'stack stamping' ROP protections to a binary.
Diversify	Evolves diverse binary implementations with unique signatures and attack surfaces.

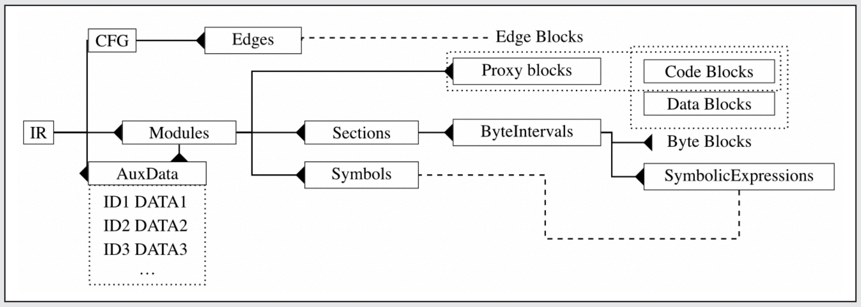


GTIRB

GrammaTech Intermediate Representation for Binaries

The GrammaTech Intermediate Representation for Binaries (GTIRB) is a machine code analysis and rewriting data structure. It is intended to facilitate the communication of binary IR between programs performing binary disassembly, analysis, transformation, and pretty printing. GTIRB is modeled on LLVM-IR, and seeks to serve a similar functionality of encouraging communication and interoperability between tools.

- `https://github.com/grammatech/gtirb`
- `https://grammatech.github.io/gtirb/`
- `https://arxiv.org/abs/1907.02859`

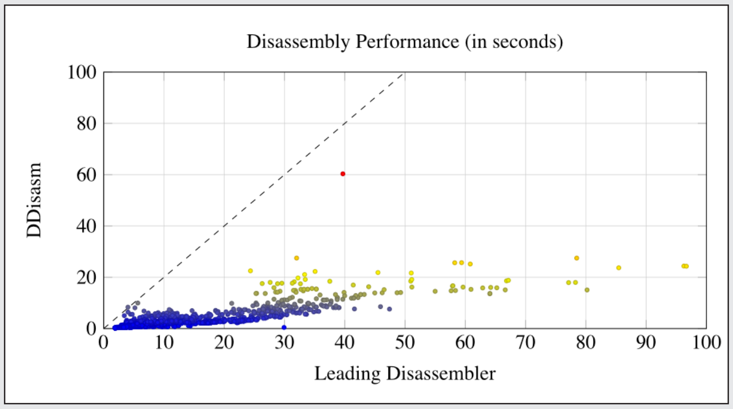


DDisasm

A fast and accurate reassemblable disassembler

DDisasm is fast disassembler which is accurate enough for the resulting assembly code to be reassembled. The disassembler is implemented using the datalog (souffle[1]) declarative logic programming language to compile disassembly rules and heuristics. The disassembler first parses binary file information and decodes a superset of possible instructions to create an initial set of datalog facts. These facts are analyzed to identify code location, symbolization, and function boundaries. The results of this analysis, a refined set of datalog facts, are then translated to the GTIRB intermediate representation for binary analysis and reverse engineering. The GTIRB pretty printer[2] may then be used to pretty print the GTIRB to reassemblable assembly code.

- `https://github.com/GrammaTech/ddisasm`
- `https://arxiv.org/abs/1906.03969`



[1] souffle `https://github.com/souffle-lang/souffle`
[2] GTIRB pretty printer `https://github.com/grammatech/gtirb-pprinter`

GRAMMATECH

BINARY ANALYSIS AND REWRITING

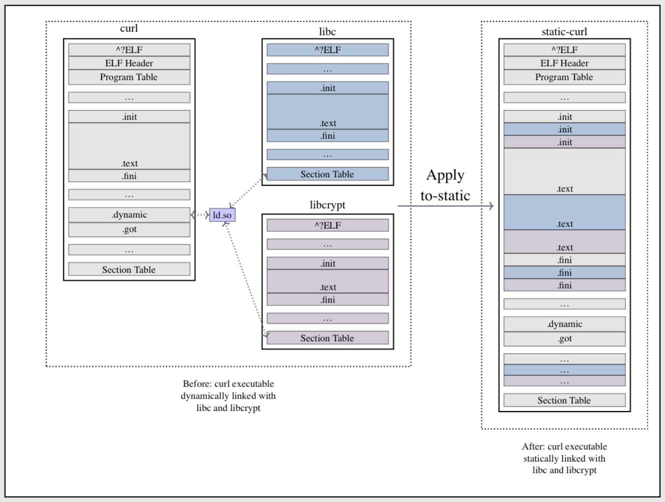
Binary Transformation
Monitoring and Hardening
Runtime Application Self-Protection

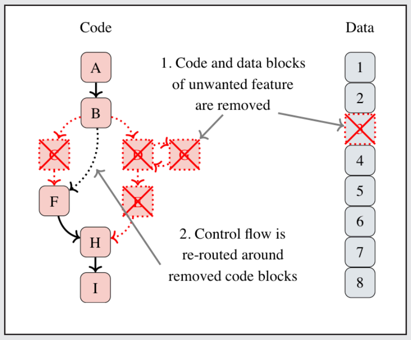
Design
Development
✓ **Pre-Deployment**
✓ **Deployment**
✓ **Sustainment**

To-static

Transform to convert a dynamically linked binary executable to one that is statically linked

To-static is a binary rewriting software transformation that takes a COTS binary executable along with the dynamic libraries it would load at runtime and consolidates them all into a single statically linked binary executable. This process is useful in cases where the equivalent statically linked executable cannot be built from source: for example, because the original source code or build system are unavailable, or are available but cannot be modified to accommodate static linking. The to-static transform confers all the benefits of static linking—simplified distribution, reduced runtime requirements, streamlined cross-library function calls—while also ensuring that any subsequent binary transformations (e.g., control flow integrity (CFI), hardening, debloating, optimization) will automatically apply to library code as well as to the main executable code.


curl
^?ELF
ELF Header
Program Table
.init
.text
.fini
.dynamic
.got
Section Table
ld.so
libc
libcrypt
Apply to-static
static-curl
Before: curl executable dynamically linked with libc and libcrypt
After: curl executable statically linked with libc and libcrypt


Code
Data
A
B
F
H
I
1. Code and data blocks of unwanted feature are removed
2. Control flow is re-routed around removed code blocks
1
2
4
5
6
7
8

Reduce

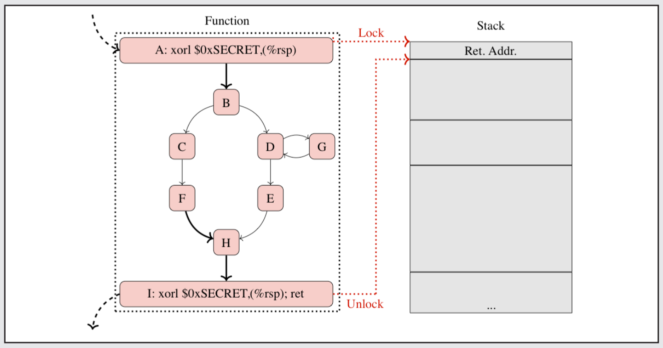
Transform to remove specified features from a binary

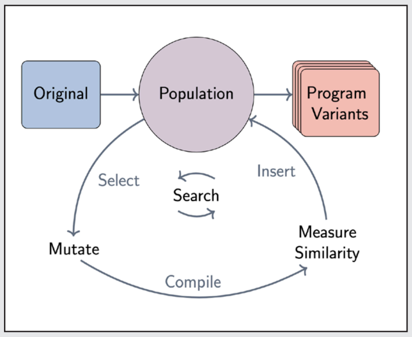
Reduce is a software transformation that takes a COTS binary executable and a set of features and rewrites the executable to remove all code which is not required by the specified features. It uses a combination of dynamic tracing and static analysis to collect a mapping from features to portions of the binary and thus identify the code and data blocks that are to be removed. It then removes these blocks, rerouting control flow as needed, and compacts the resulting binary image. This can be useful when software includes undesirable or unused features, especially when those features have a negative impact on software performance or security.

Stack-stamp

Transform to apply ‘stack stamping’ ROP protections to a binary

Stack stamping is a technique to help mitigate ROP style attacks. This is done by ‘stamping’ the return address on the stack, thus encrypting it. Before it is popped off the stack and used, it is decrypted by ‘unstamping’ it. This can be an efficient protection, as no registers are needed, and while flags are affected, they are only affected at function entry and exits where they do not need to be preserved. An attacker who wishes to hijack control flow by overwriting the return address now has a more difficult task: not only must they find an opportunity to write to the stack, but they must write a value that **decodes** to their replacement address instead of simply writing the address itself.





Diversify

Evolve diverse binary implementations with unique signatures and attack surfaces

Diversify is a source-to-source software transformation which takes in an original program and a test suite and produces a set of program variants that retain functionality on the test suite while achieving binary diversity. Binary diversity is measured by compiling the software variants to binary executables and then comparing the results against the compiled binary of the original program. Diversity is achieved by repeatedly selecting a mutation from a large set of candidates and applying it to the source code of the original program. Mutations range from simple (e.g., deletion) to complex (e.g., structured software refactorings such as variable or function inlining or extraction). Repeated application of mutations in a search-based algorithm that maximizes diversity can evolve a population of functionally equivalent variants with significant differences from the original binary and from one another.