ROP Gadget Prevalence and Survival under Compiler-based Binary Diversification Schemes

Joel Coffman Daniel M. Kelly Christopher C. Wellons
Andrew S. Gearhart

Johns Hopkins University Applied Physics Laboratory

2nd International Workshop on Software Protection

Cybersecurity

Current Landscape

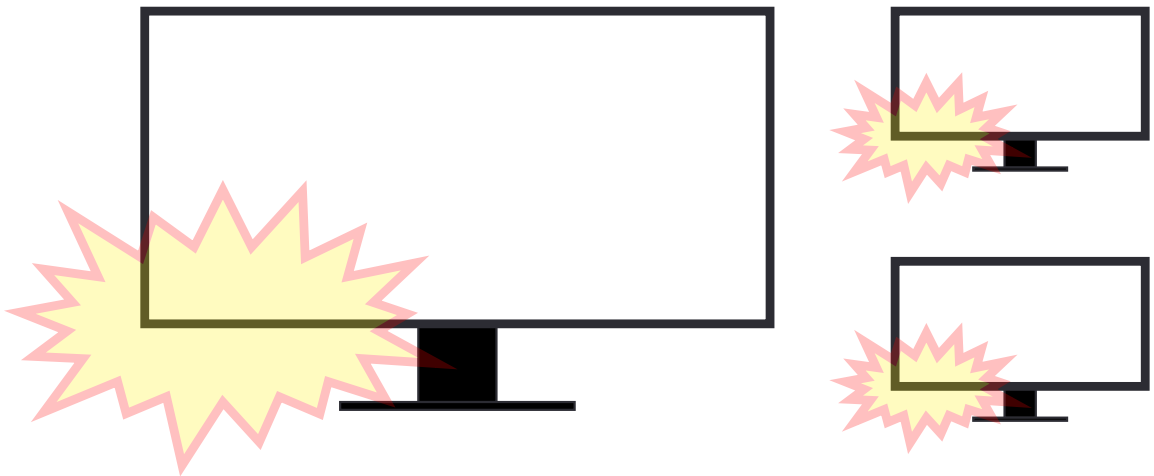
Compromise once, compromise everywhere

Cybersecurity

Current Landscape

Compromise once, compromise everywhere

- Systems are homogeneous and share vulnerabilities

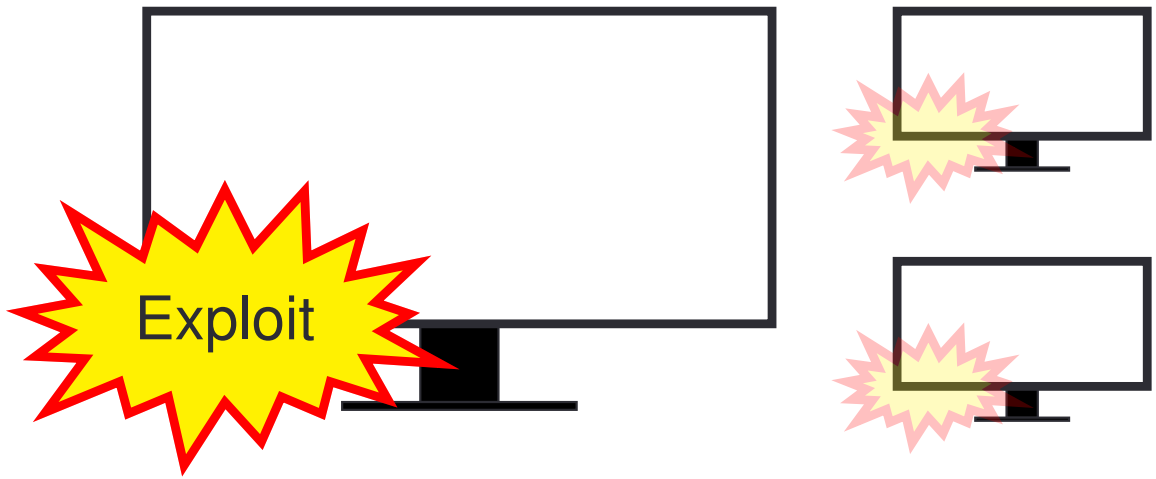


Cybersecurity

Current Landscape

Compromise once, compromise everywhere

- Systems are homogeneous and share vulnerabilities
- Single exploit reused to compromise all systems
  - e.g., Morris, Nimda, Conficker, and Heartbleed

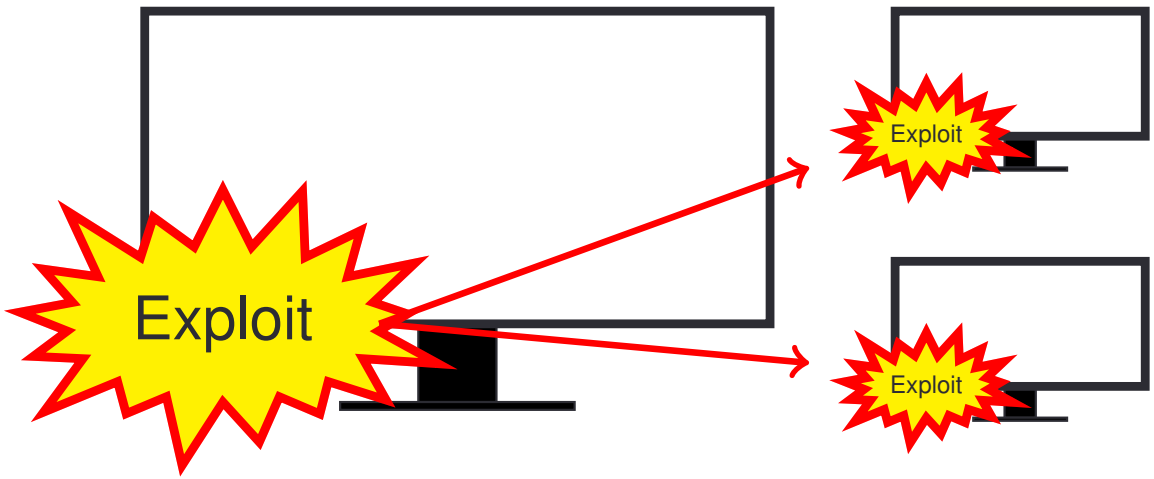


Cybersecurity

Current Landscape

Compromise once, compromise everywhere

- Systems are homogeneous and share vulnerabilities
- Single exploit reused to compromise all systems
  - e.g., Morris, Nimda, Conficker, and Heartbleed



Cybersecurity (continued)

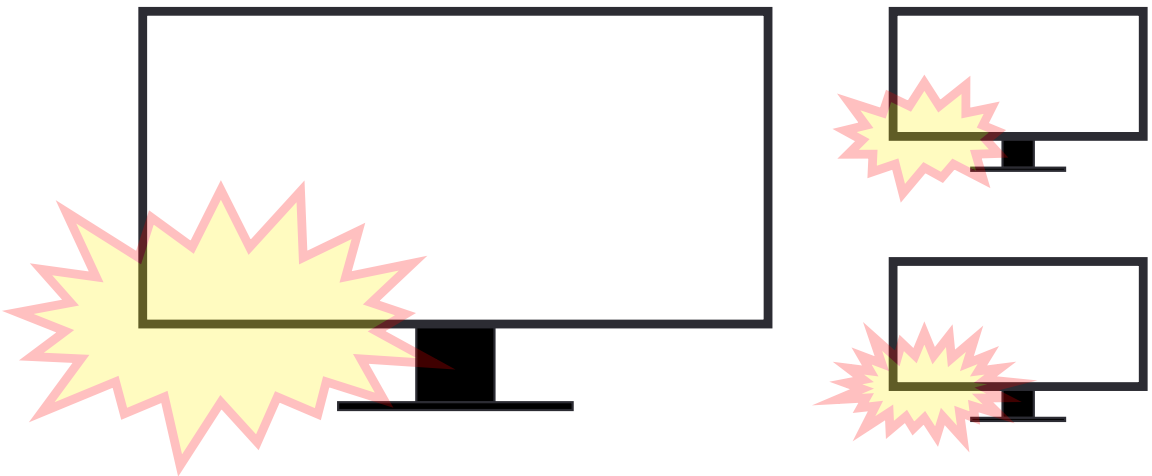
Diversity

Software diversity breaks the assumption of consistency in operational environments

Cybersecurity (continued)

Diversity

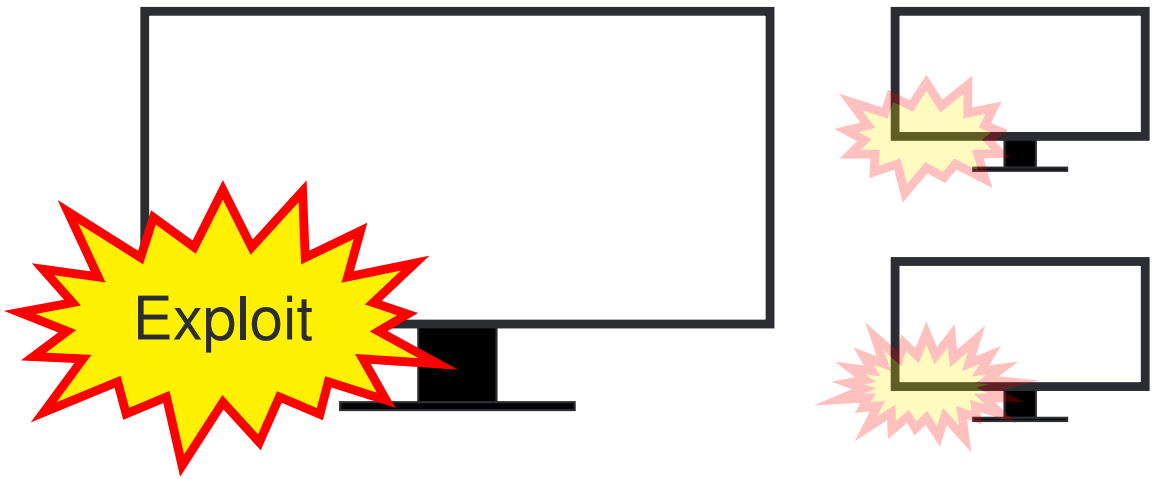
Software diversity breaks the assumption of consistency in operational environments



Cybersecurity (continued)

Diversity

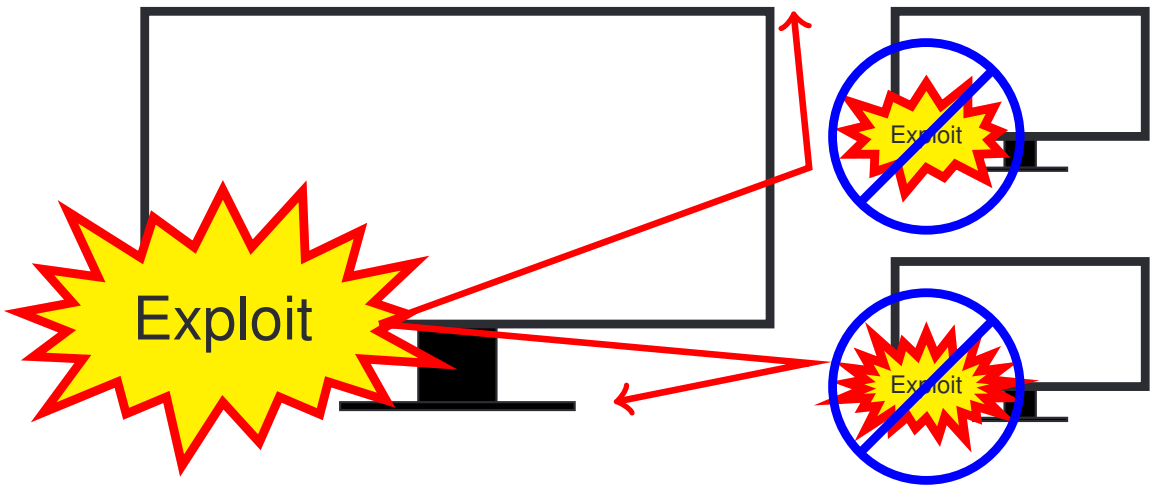
Software diversity breaks the assumption of consistency in operational environments



Cybersecurity (continued)

Diversity

Software diversity breaks the assumption of consistency in operational environments

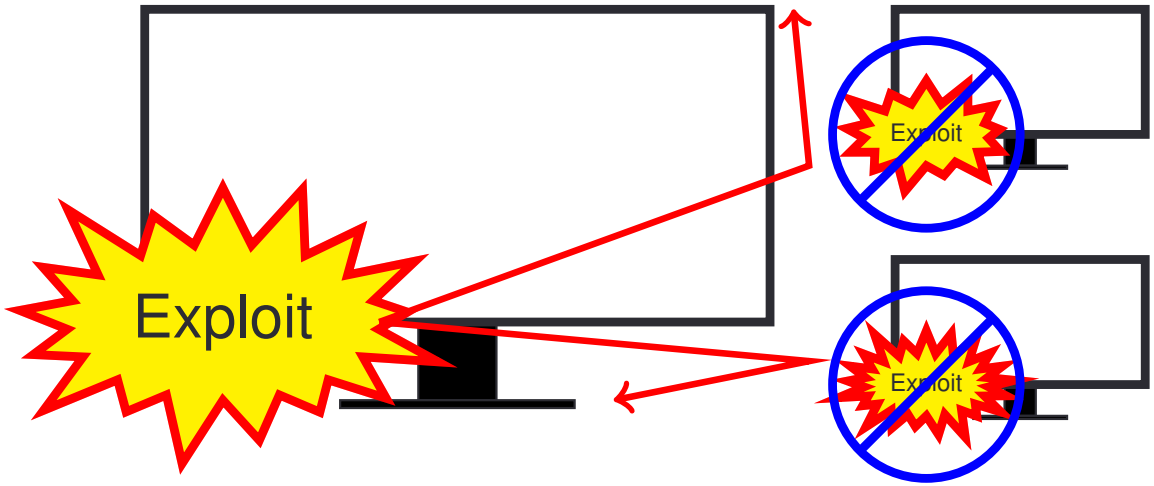


Cybersecurity (continued)

Diversity

Software diversity breaks the assumption of consistency in operational environments

- Increases attacker cost by reducing exploit reuse

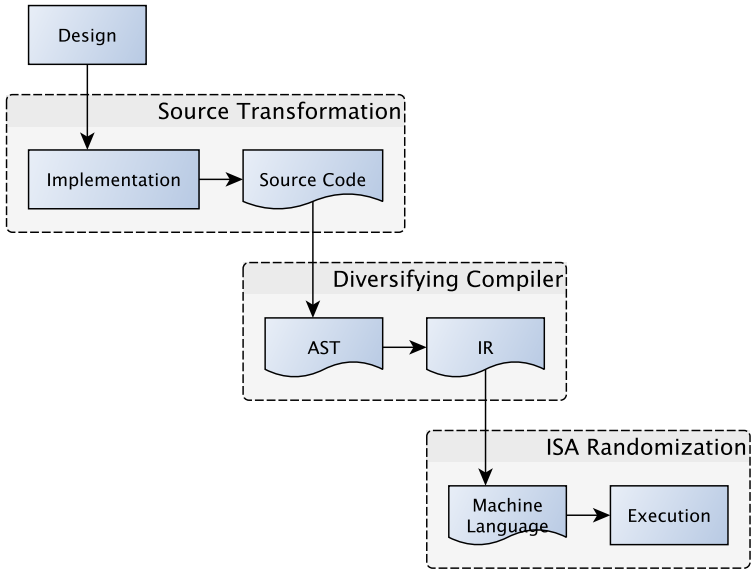


Software Diversity

Opportunities

Techniques exist to introduce diversity throughout the software development process

- Design diversity
- *N*-version programming
- Diversifying compilers
- Instruction set architecture (ISA) randomization

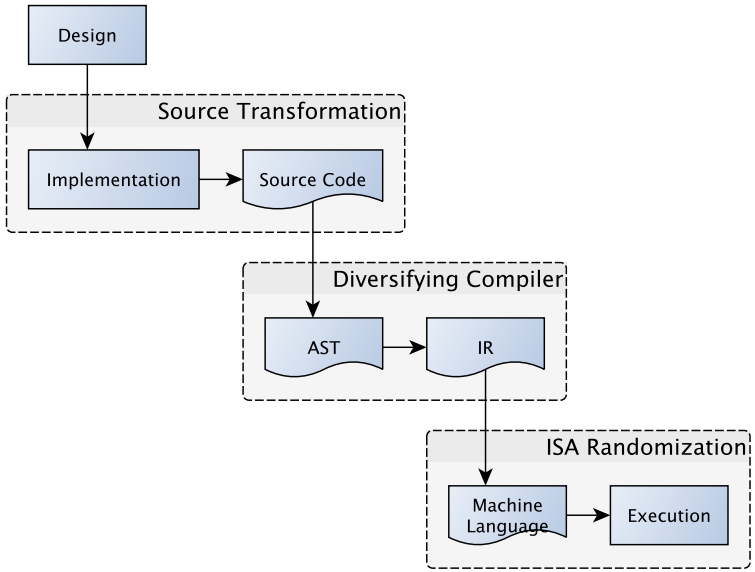


Software Diversity

Opportunities

Techniques exist to introduce diversity throughout the software development process

- Design diversity
- *N*-version programming
- Diversifying compilers
- Instruction set architecture (ISA) randomization



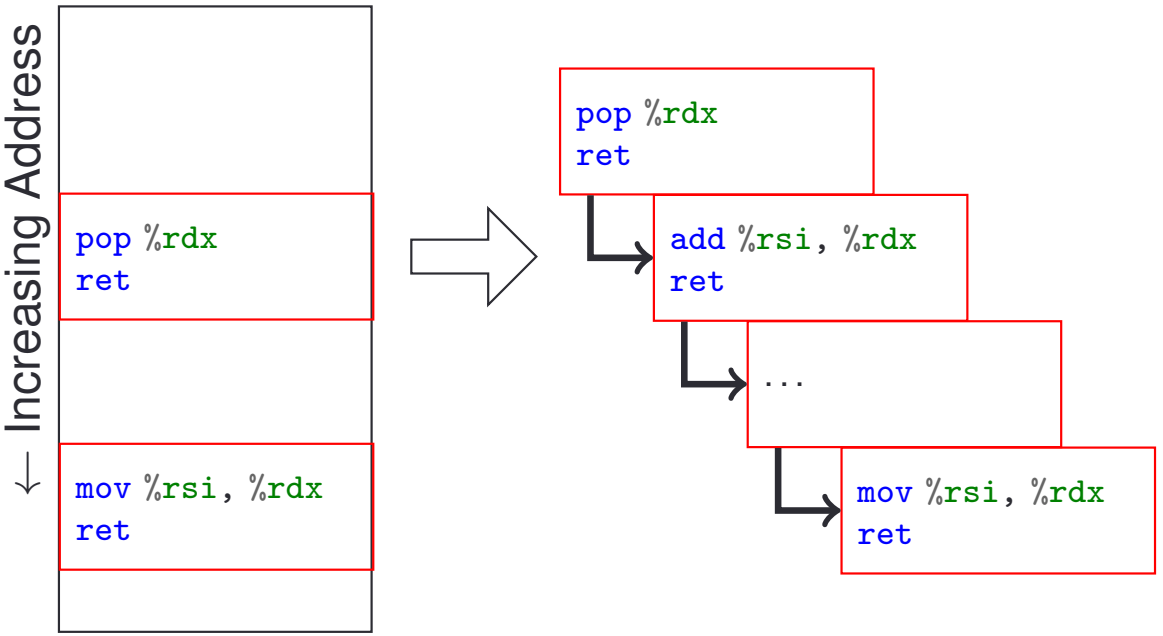
Our focus: diversifying compilers

- Allows transformation and optimization using existing tools
- Several open source projects exist

Reducing Exploit Reuse

Code reuse attacks are increasingly common

- Response to preventing execution of code in data segments
- Return-oriented programming (ROP) is a class of code reuse attacks



Prior Work

Little work evaluates the effectiveness of the proposed techniques

- Many security evaluations are based on logical arguments or concrete attacks

> *The study of how diversity affects the adversary's effort is in its infancy. [. . .] Numerous papers have been published on how to perform sound performance evaluations; [. . .] a similar effort should be undertaken with respect to efficacy metrics for diversified software. [Larsen et al., 2014]*

Prior Work (continued)

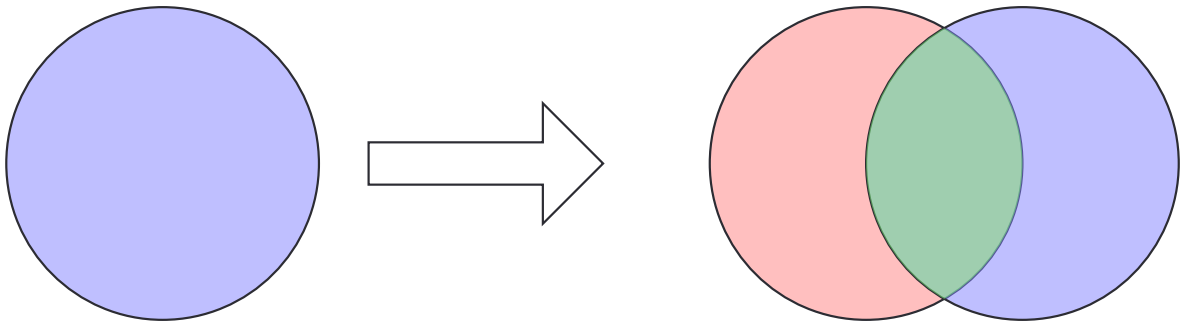
Few studies consider how diversity interferes with exploit reuse

- Testing against concrete attacks does not demonstrate effectiveness against alternative tactics
  - e.g., the transition from code injection to code reuse attacks
- Attack-specific analyses should consider an attacker's learning
  - e.g., invariance among diversified variants

Prior Work (continued)

Few studies consider how diversity interferes with exploit reuse

- Testing against concrete attacks does not demonstrate effectiveness against alternative tactics
  - e.g., the transition from code injection to code reuse attacks
- Attack-specific analyses should consider an attacker's learning
  - e.g., invariance among diversified variants



Today
Software Monoculture

Tomorrow
Diversified Software

Gadget Survival

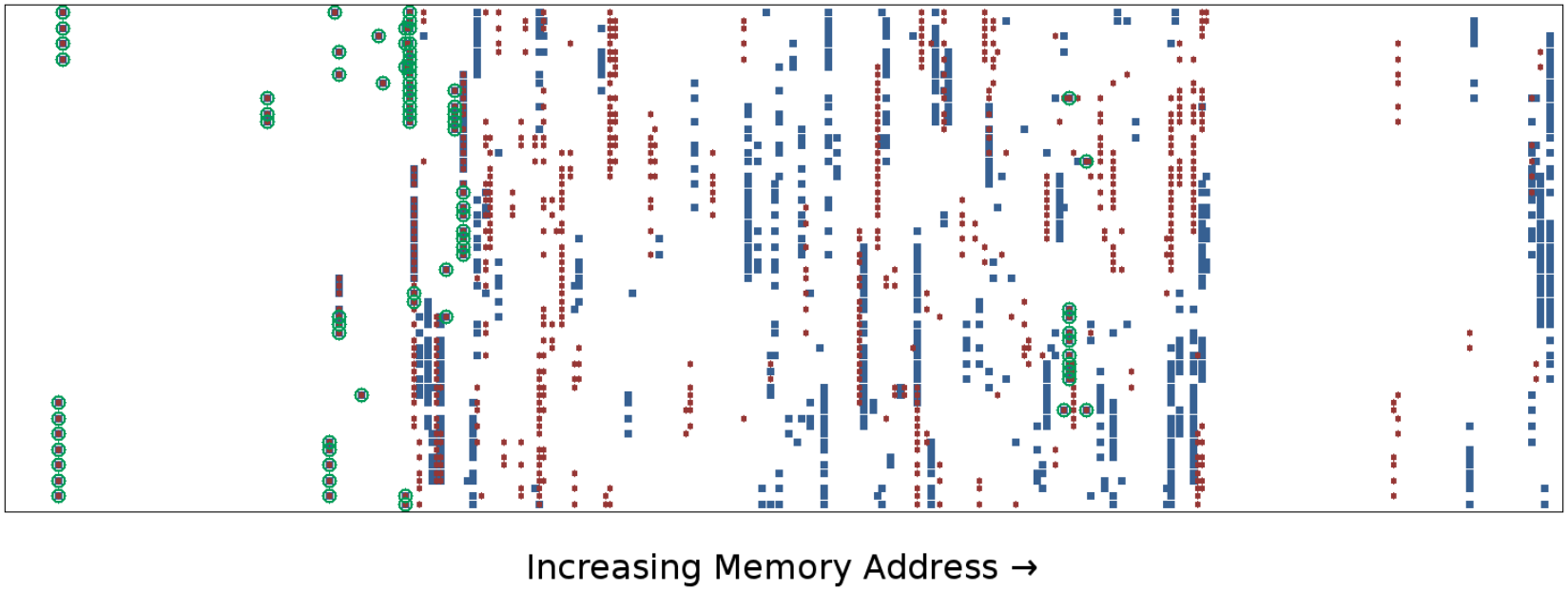


Figure: Gadget locations in two variants (red, blue) of `dirname` with common gadgets circled in green.

Outline

Diversity Techniques

Techniques implemented by the multicompiler [Homescu et al., 2013] and Obfuscator-LLVM [Junod et al., 2015]

NOP insertion Changes address of ROP gadgets

Instruction substitution Replaces instructions with arithmetic identities

- e.g., $b + c = b - (-c) = -(-b + (-c))$

Schedule randomization Reorders independent instructions

Bogus control flow Inserts a basic block with an opaque predicate to hinder reverse engineering

Control flow flattening Obfuscates the control flow graph via indirect jumps using "jump tables"

Function shuffling Reorders functions in the executable

Data Sets

GNU core utilities

- 103 different binaries ($\approx$ 60 KLOC)
  - Many binaries limits the impact of outliers on analysis
- Open source for reproducibility and amenable to compiler-based diversity schemes

Data Sets

GNU core utilities

- 103 different binaries ($\approx$ 60 KLOC)
  - Many binaries limits the impact of outliers on analysis
- Open source for reproducibility and amenable to compiler-based diversity schemes

Variants

- Generate 100 unique variants for each diversity technique
- Select 4000 unique combinations from the $\binom{100}{k}$ possibilities
  - $4000 \approx \max_{k \in \{2,\ldots,16\}} \binom{100}{k}$

Metrics

Statically identify all gadgets in binaries

- Disassemble a sliding window of 25 bytes looking for a valid sequence that terminates in a return instruction

Metrics

Statically identify all gadgets in binaries

- Disassemble a sliding window of 25 bytes looking for a valid sequence that terminates in a return instruction

Survivor [Homescu et al., 2013] Identical gadgets have the same sequence of bytes and same offset in binary

Metrics

Statically identify all gadgets in binaries

- Disassemble a sliding window of 25 bytes looking for a valid sequence that terminates in a return instruction

Survivor [Homescu et al., 2013] Identical gadgets have the same sequence of bytes and same offset in binary

Bag of Gadgets Identical gadgets have the same sequence of bytes

Metrics

Statically identify all gadgets in binaries

- Disassemble a sliding window of 25 bytes looking for a valid sequence that terminates in a return instruction

Survivor [Homescu et al., 2013] Identical gadgets have the same sequence of bytes and same offset in binary

- Represents no prior knowledge available to attacker

Bag of Gadgets Identical gadgets have the same sequence of bytes

- Represents attacker with knowledge of application but not the specific variant

Number of Gadgets

Most diversity techniques increase the number of gadgets

- Modifying binary introduces (different) gadgets not present in original

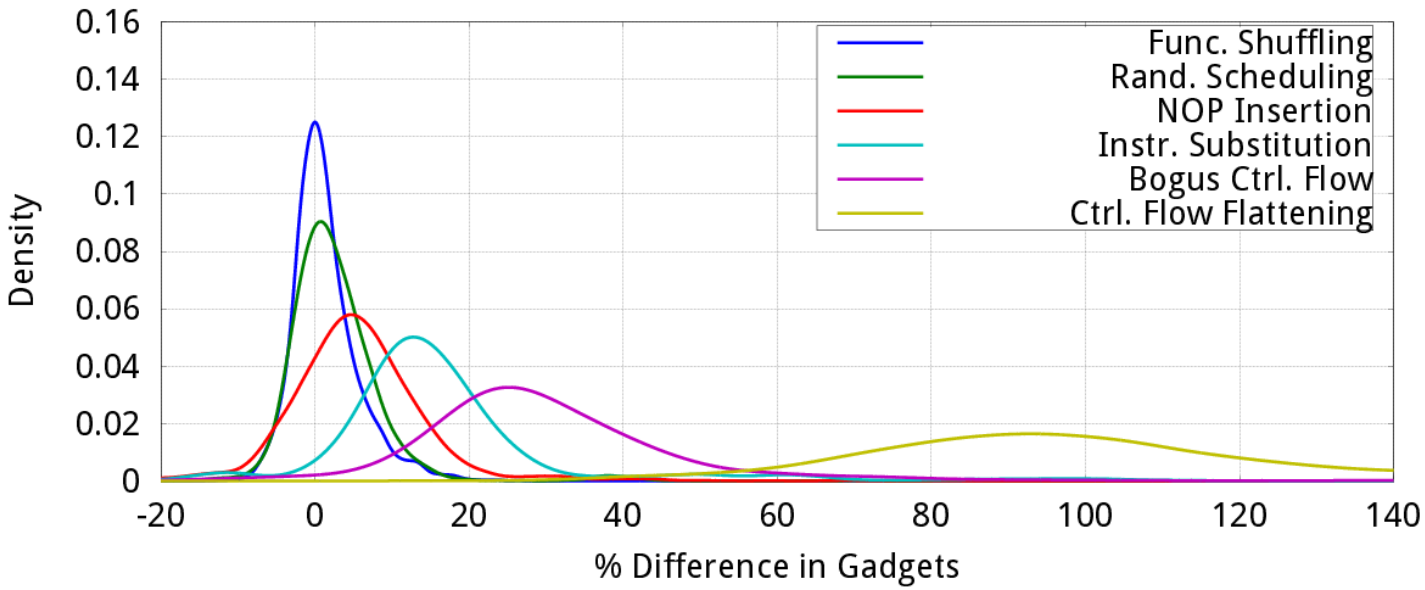


Figure: Kernel density estimate of the probability density function of the change in the number of gadgets

Gadget Survival (Survivor)

1–8% of gadgets are common among variants

- Population size has little impact on survival rate

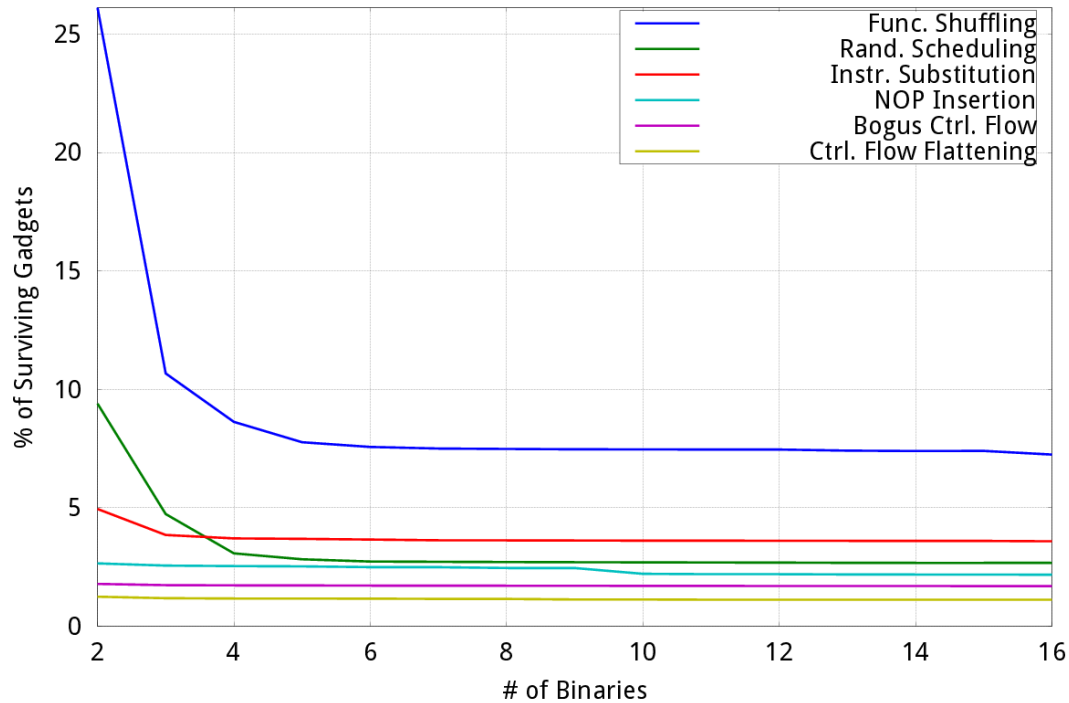


Figure: Median gadget survival across the GNU core utilities

Gadget Survival (Bag of Gadgets)

Significantly worse performance than the Survivor metric

- Risk if information disclosure vulnerabilities exist

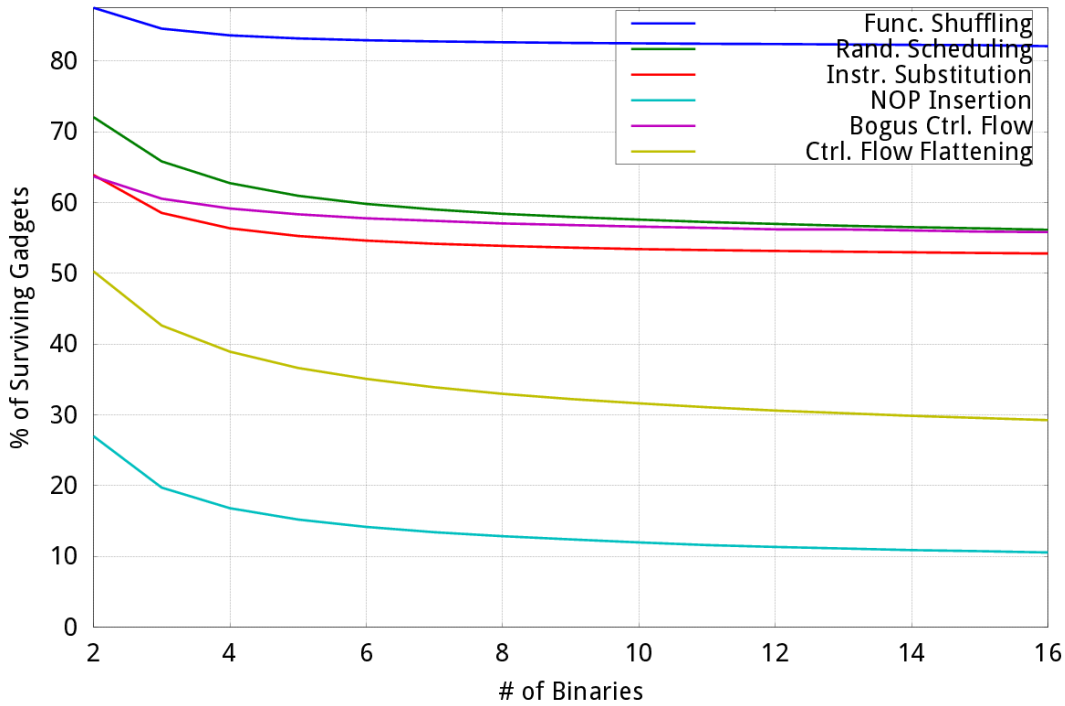


Figure: Median gadget survival across the GNU core utilities

Gadgets Remaining

Very few gadgets are common to all variants

- Many attacks require only a few gadgets ($\approx$ 10–20 [Pappas et al., 2012])

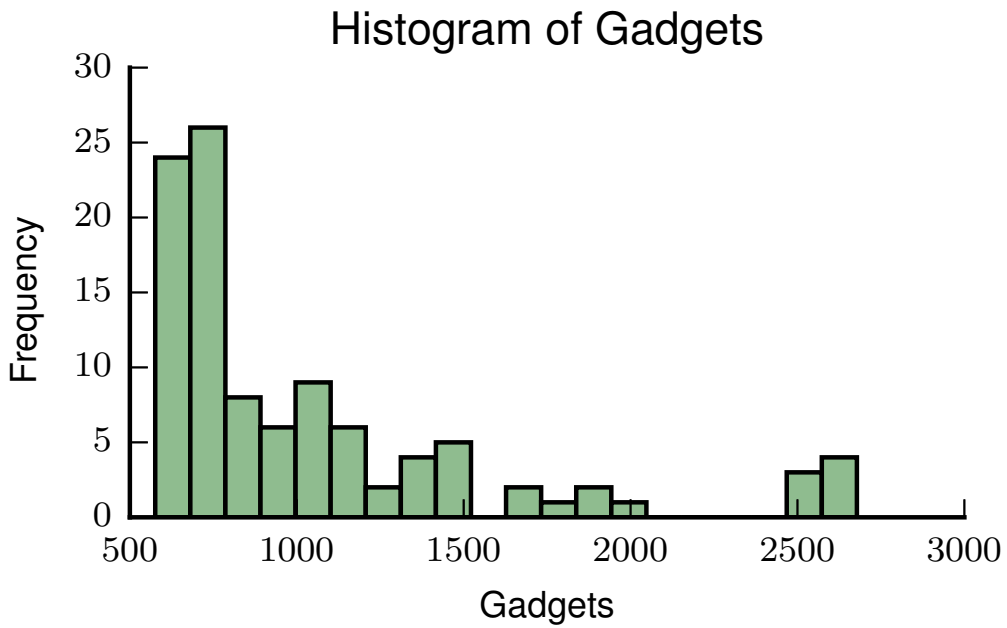


Figure: Histogram of the raw number of gadgets in the GNU core utilities

Gadget Locations

Is it possible to identify surviving gadgets *a priori*?

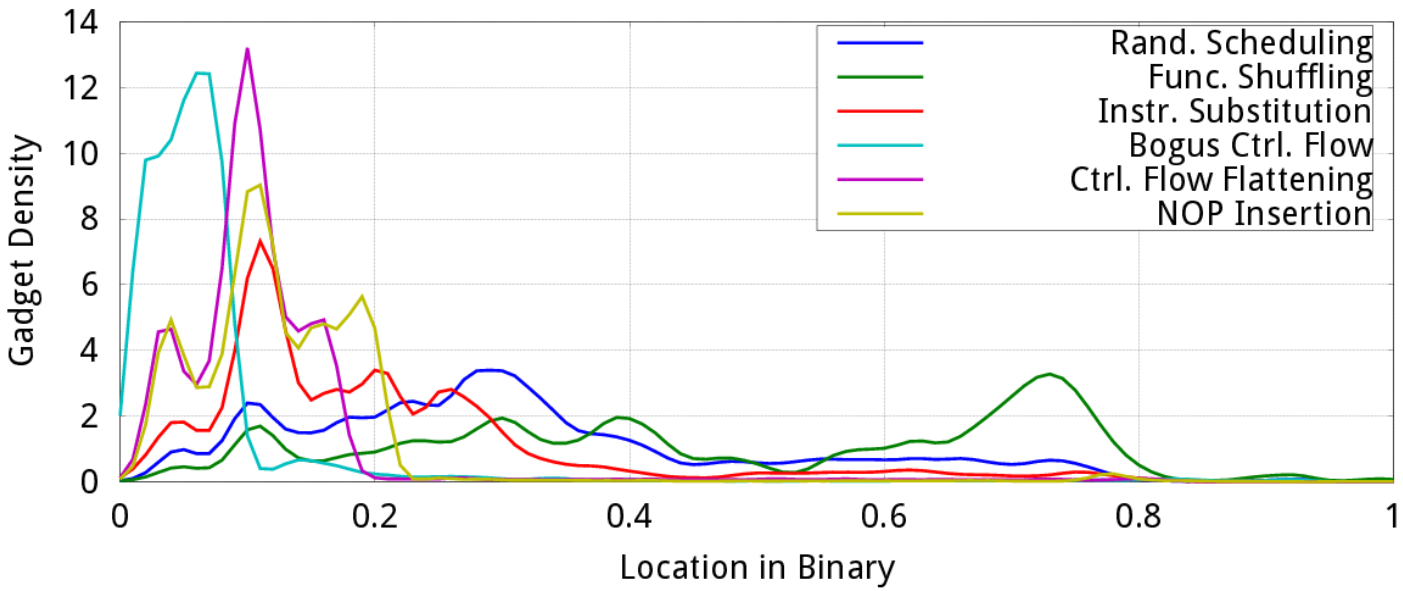


Figure: Location distribution of gadgets that survive diversification

Most surviving gadgets appear at the start of the binary

- Fewer opportunities for diversification

Outline

Conclusions

Empirical evaluation of the effectiveness of existing software diversity techniques

Conclusions

Empirical evaluation of the effectiveness of existing software diversity techniques

Survivor The number of ROP gadgets surviving diversification is close to the threshold required for a successful attack

- The number of surviving gadgets is essentially constant across the GNU core utilities

Conclusions

Empirical evaluation of the effectiveness of existing software diversity techniques

Survivor The number of ROP gadgets surviving diversification is close to the threshold required for a successful attack

- The number of surviving gadgets is essentially constant across the GNU core utilities

Bag of Gadgets Minimal effort expected to adapt an existing exploit to a different variant

- Challenge only for attackers without access to target variant

Ideal Diversity Schemes

What defines a “good” diversity scheme?

- Resistance to attacks on a single variant
- Reducing exploit reuse among diversified variants
  - Diversity improves the security of the *population*
- Resistance to reverse engineering for vulnerability detection and detection of the diversity details
  - Obfuscation is a related area of study (as is anti-attribution)
  - Diversity schemes ideally follow Kerckhoffs’s Principle

Future Work

Open research questions

Future Work

Open research questions

- Theoretical bounds on the effectiveness of diversity techniques

Future Work

Open research questions

- Theoretical bounds on the effectiveness of diversity techniques
- Interaction among techniques being composed

Future Work

Open research questions

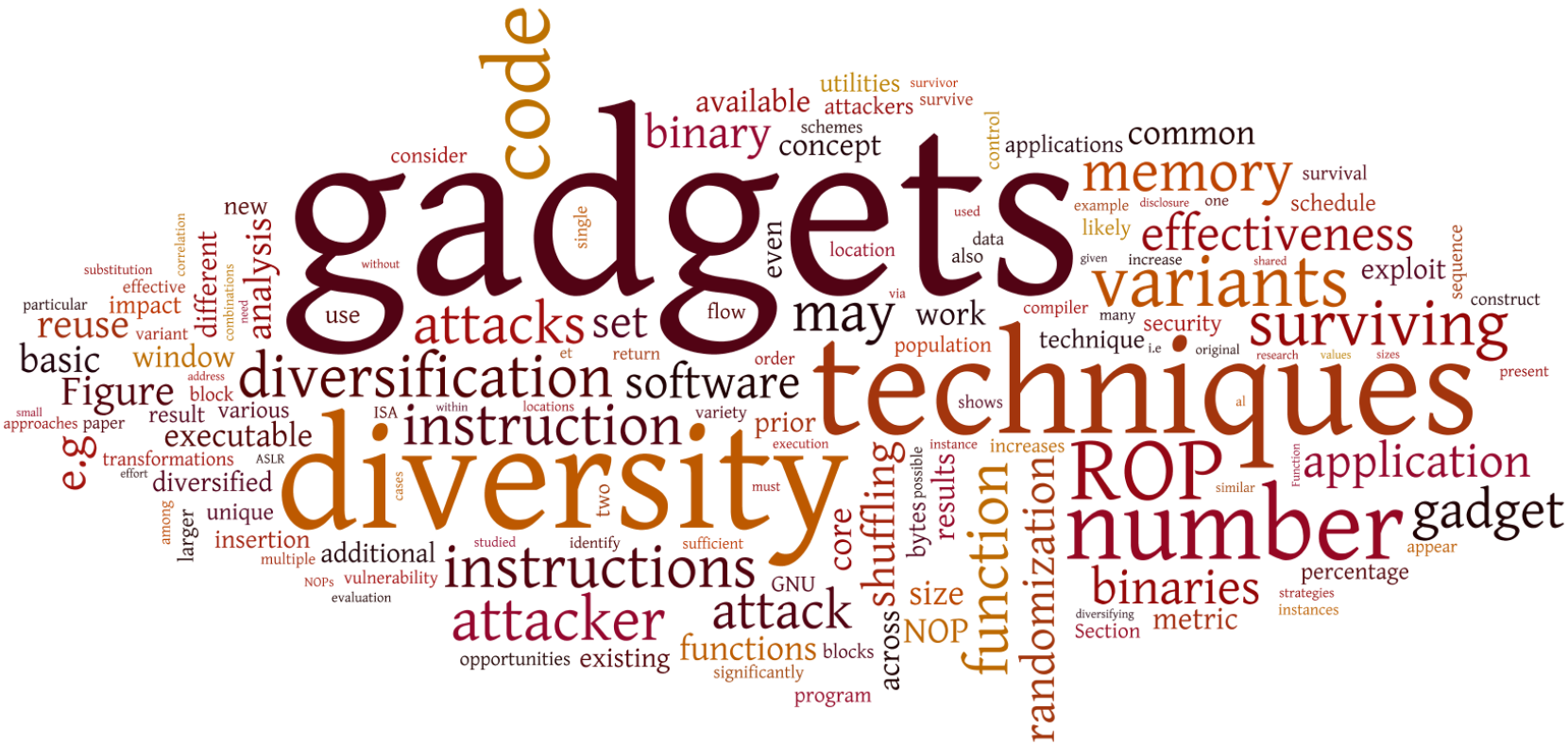
- Theoretical bounds on the effectiveness of diversity techniques
- Interaction among techniques being composed
- Feasibility of identifying surviving gadgets without analyzing the entire population

Future Work

Open research questions

- Theoretical bounds on the effectiveness of diversity techniques
- Interaction among techniques being composed
- Feasibility of identifying surviving gadgets without analyzing the entire population
- New metrics that quantify the security impact of diversity techniques

Questions?


gadgets
code
diversity
techniques
instruction
diversification
software
instructions
attacks
attacker
attack
ROP
number
gadget
application
binaries
memory
effectiveness
variants
surviving
shuffling
randomization
function
binary
available
utilities
attackers
survive
survivor
schemes
concept
consider
applications
common
survival
schedule
example
disclosure
one
likely
data
also
location
used
single
even
flow
may
work
compiler
many
security
technique
population
order
return
et
set
use
without
new
analysis
different
combinations
correlation
substitution
effective
particular
impact
reuse
variant
window
basic
Figure
block
address
small
approaches
paper
result
various
executable
e.g
transformations
ASLR
effort
diversified
unique
insertion
multiple
additional
vulnerability
evaluation
NOPs
among
larger
studied
identify
sufficient
two
cases
must
core
GNU
across
size
NOP
functions
blocks
significantly
program
opportunities
existing
ISA
within
locations
variety
prior
execution
shows
al
instance
increases
bytes possible
results
Function
similar
appear
percentage
strategies
instances
metric
Section
diversifying
present
construct
sequence
exploit
shared
given
increase
control
compiler
security
technique
i.e
original
research
values
sizes

Appendix

References I

[Homescu et al., 2013] Homescu, A., Neisius, S., Larsen, P., Brunthaler, S., and Franz, M. (2013). Profile-guided Automated Software Diversity. In *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, CGO '13, pages 1–11.

[Junod et al., 2015] Junod, P., Rinaldini, J., Wehrli, J., and Michielin, J. (2015). Obfuscator-LLVM: Software Protection for the Masses. In *Proceedings of the 1st International Workshop on Software Protection*, SPRO '15, pages 3–9.

[Larsen et al., 2014] Larsen, P., Homescu, A., Brunthaler, S., and Franz, M. (2014). SoK: Automated Software Diversity. In *2014 IEEE Symposium on Security and Privacy*, pages 276–291.

References II

[Pappas et al., 2012] Pappas, V., Polychronakis, M., and Keromytis, A. D. (2012). Smashing the Gadgets: Hindering Return-Oriented Programming Using In-place Code Randomization. In *2012 IEEE Symposium on Security and Privacy*, pages 601–615.

Glossary I

ISA instruction set architecture. 11, 12

KLOC thousands of lines of code. 20, 21

ROP return-oriented programming. 13, 19, 32–34