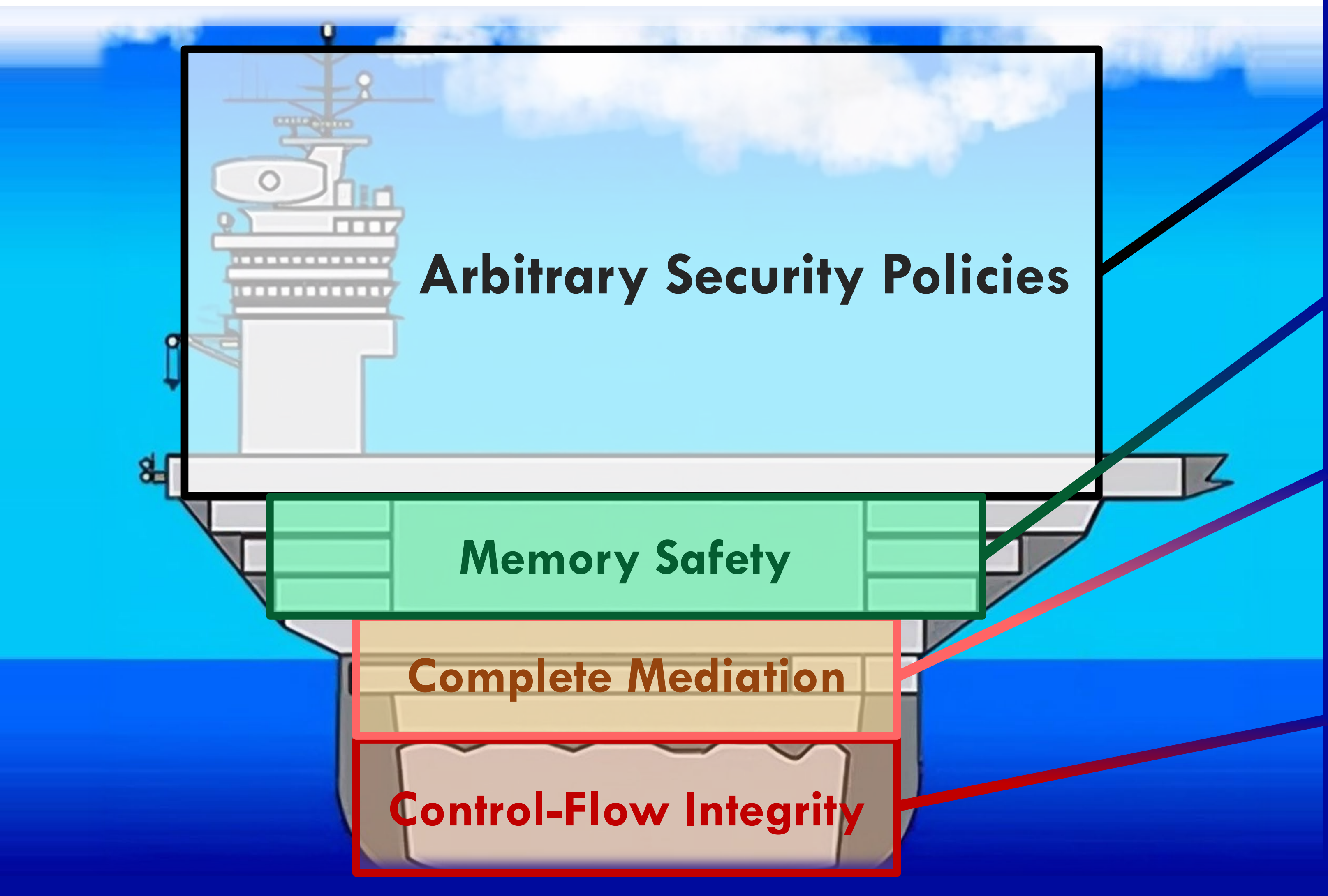
Dr. Kevin Hamlen
Louis A. Beecherl, Jr. Distinguished Professor
Computer Science Department
Cyber Security Research and Education Institute
The University of Texas at Dallas

Software Attack Surface Reduction On The Fly

Dr. Kevin Hamlen

ONR Award N00014-21-1-2654

Foundations of Software Security



- Safety + Liveness = Everything
  - Safety: “bad events” prohibited
  - Liveness: “good events” guaranteed

- Memory Safety
  - Secure security state storage

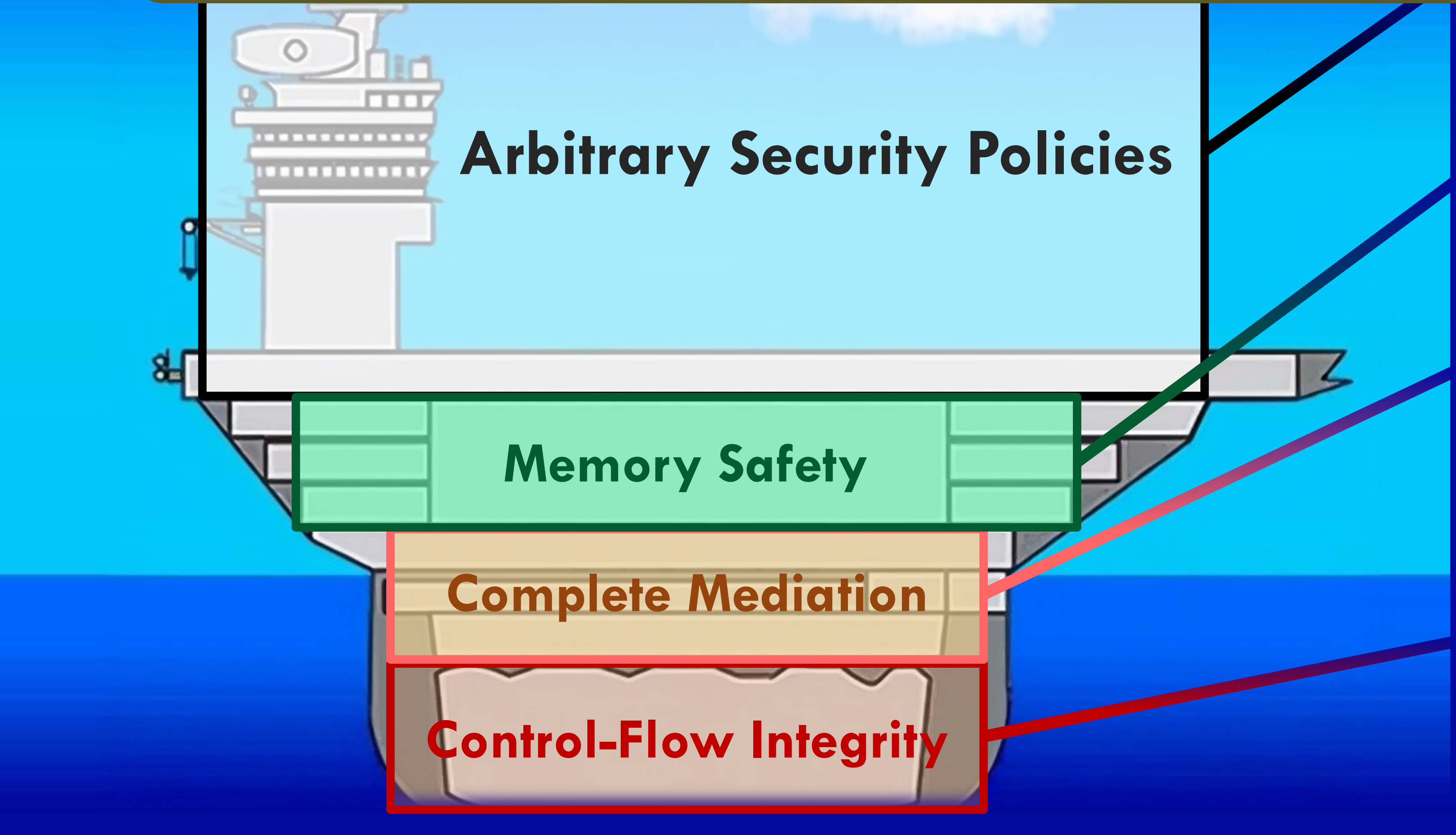
- Complete Mediation
  - Surround all security-relevant operations with uncircumventable guard code (security checks)

- Control-Flow Integrity (CFI)
  - Constrains execution order to a graph of allowed flows
  - Affords provably uncircumventable basic blocks of program instructions

Foundations of Software Security

Static Code Instrumentation

Dynamic Policy Enforcement

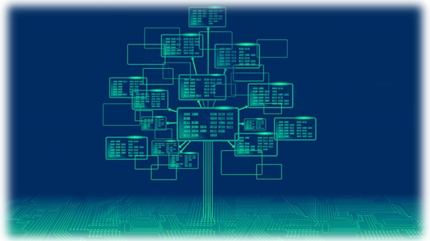



```
for (int i = 0; i < 32; ++i) {
  if (safe(&buf[i]))
    buf[i] = i;
  else
    abort(“memory corruption!”);
  if (++counter > max)
    abort(“deadlock alert”);
  if (check(&obj->method)
      && valid_args(buf, i)) {
    obj->method(buf, i);
  } else
    abort(“security alert!”);
}
```

Binary Code Debloating Architecture

1) Learn debloating policy from execution traces
   - consumer has no formal policy specification
   - consumer is not aware of all "undesired" program functionalities
2) Machine learning derives suitable policy from a whitelist of traces
3) Enforce learned policy with source-free, **context-sensitive** CFI
   - generalizes and subsumes non-contextual CFI and code byte erasure
4) Machine-validate binary hardening transforms for highest assurance
   - Picinæ: <u>P</u>latform <u>In</u> <u>C</u>oq for <u>IN</u>struction-level <u>A</u>nalysis of <u>E</u>xecutables

trace
whitelist

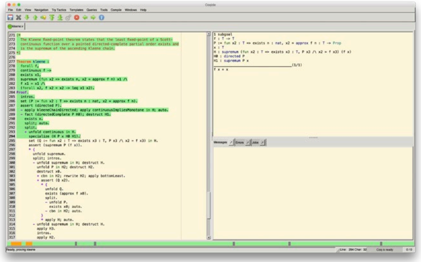


Machine Learning

policy

Binary, Context-sensitive CFI

debloated
binary

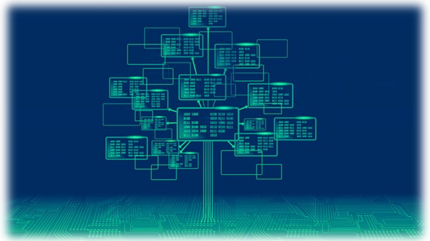


Formal Validation

Binary Code Debloating Architecture

1) Learn debloating policy from execution traces
   - consumer has no formal policy specification
   - consumer is not aware of all "undesired" program functionalities
2) Machine learning derives suitable policy from a whitelist of traces
3) **Enforce learned policy with source-free, context-sensitive CFI**
   - generalizes and subsumes non-contextual CFI and code byte erasure
4) Machine-validate binary hardening transforms for highest assurance
   - Picinæ: Platform In Coq for INstruction-level Analysis of Executables

trace whitelist

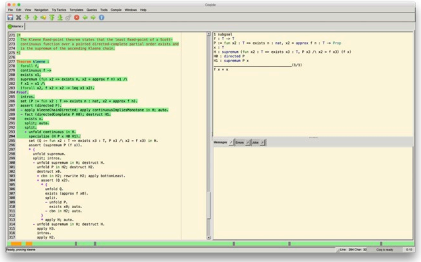


Machine Learning

policy

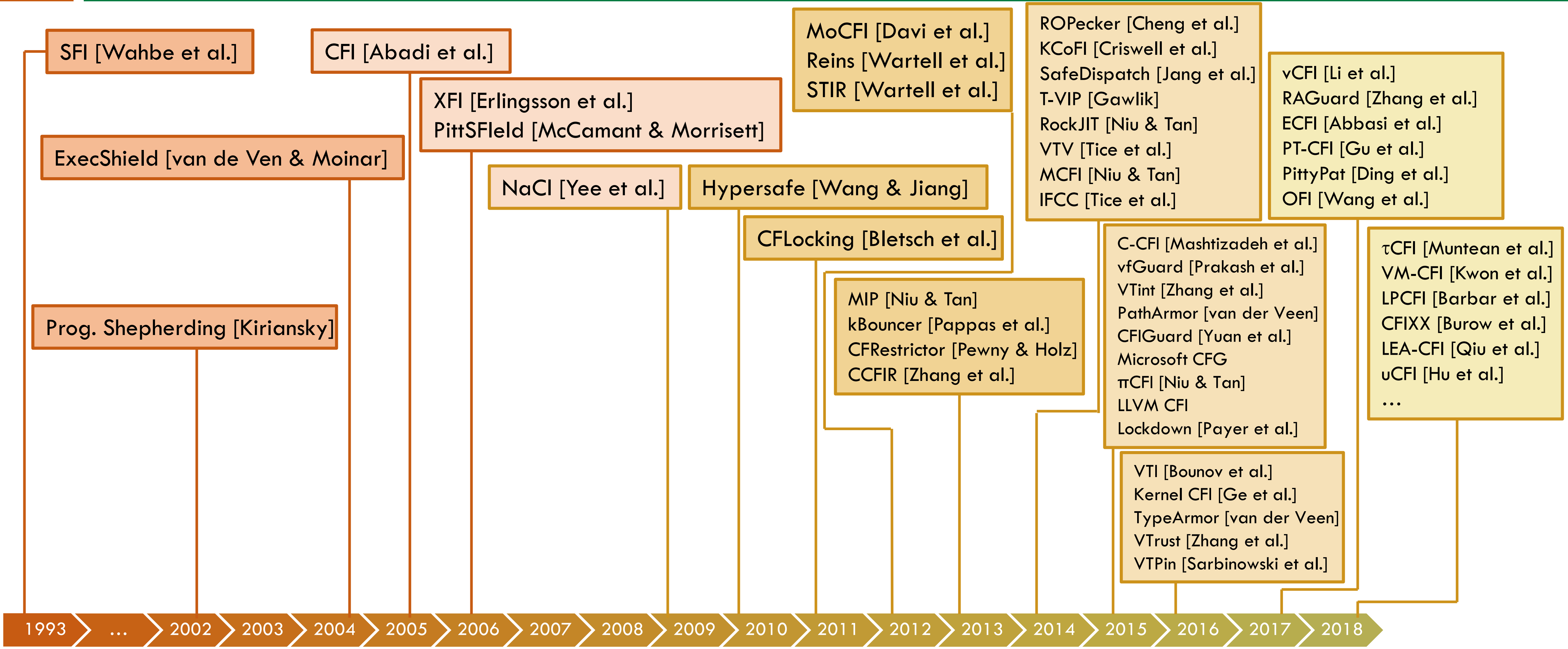
Binary, Context-sensitive CFI

debloated binary



Formal Validation

CFI Research Timeline



CFI Research Timeline

SFI [Wahbe et al.]

ExecShield [van de Ver

Prog. Shepherding [Kiri

1993 … 2002 200

SECURITY

- RIPE test suite

PERFORMANCE

- CPU SPEC benchmarks

COMPATIBILITY

-

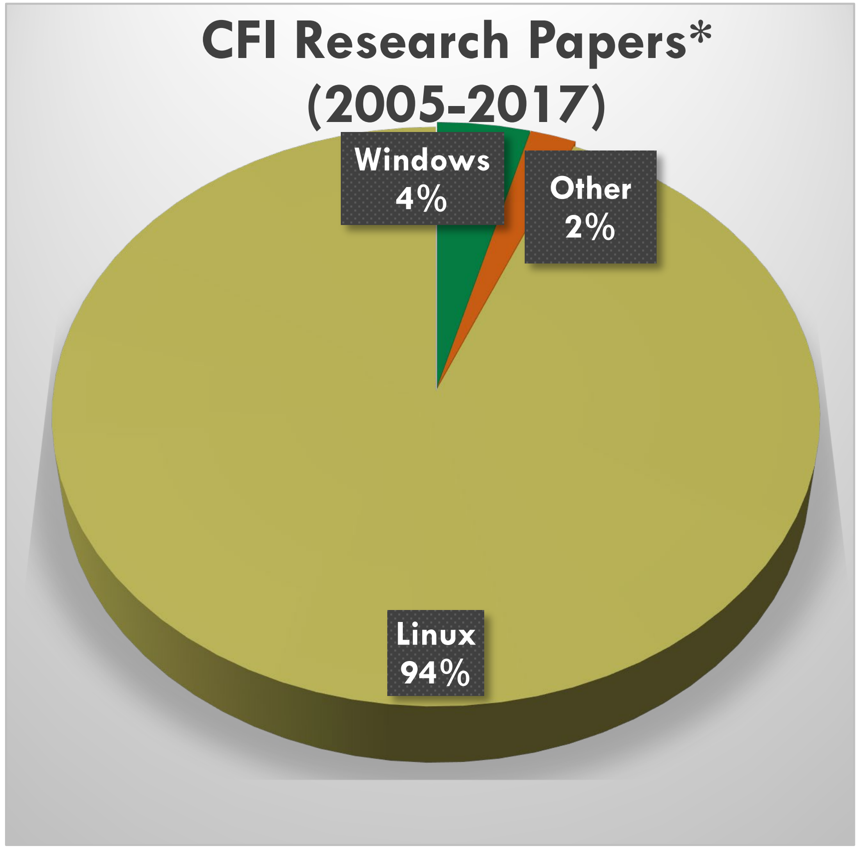
CFLocking [Bletsch et al.]

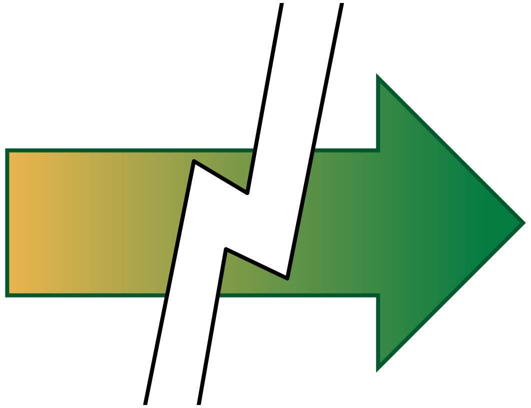
et al.]
rd [Zhang et al.]
bbasi et al.]
Gu et al.]
[Ding et al.]
ang et al.]

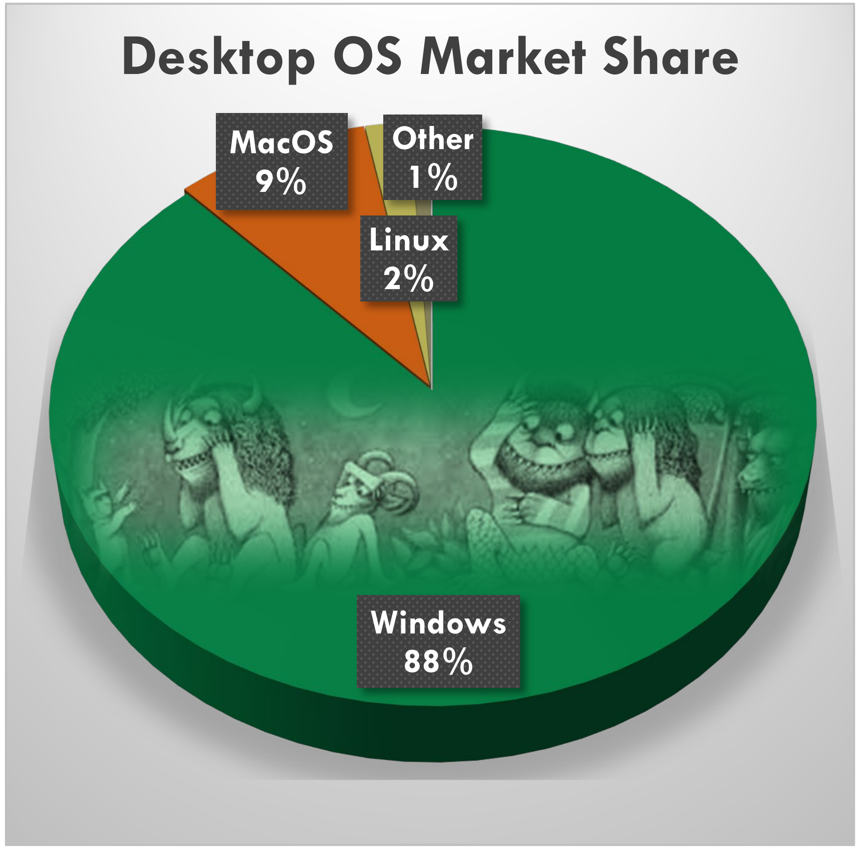
τCFI [Muntean et al.]
VM-CFI [Kwon et al.]
LPCFI [Barbar et al.]
CFIXX [Burow et al.]
LEA-CFI [Qiu et al.]
uCFI [Hu et al.]
…

18

Scalability Gap



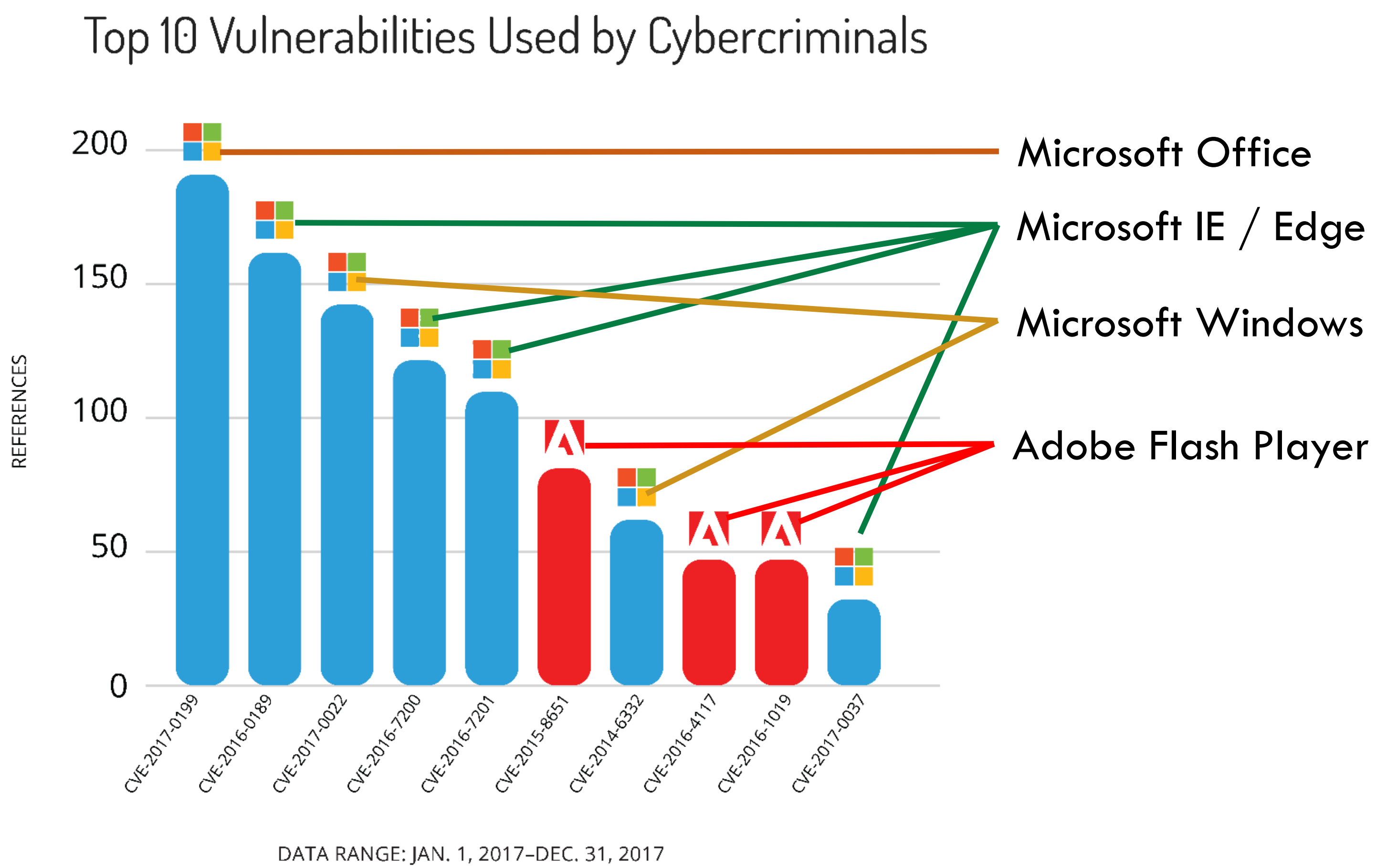


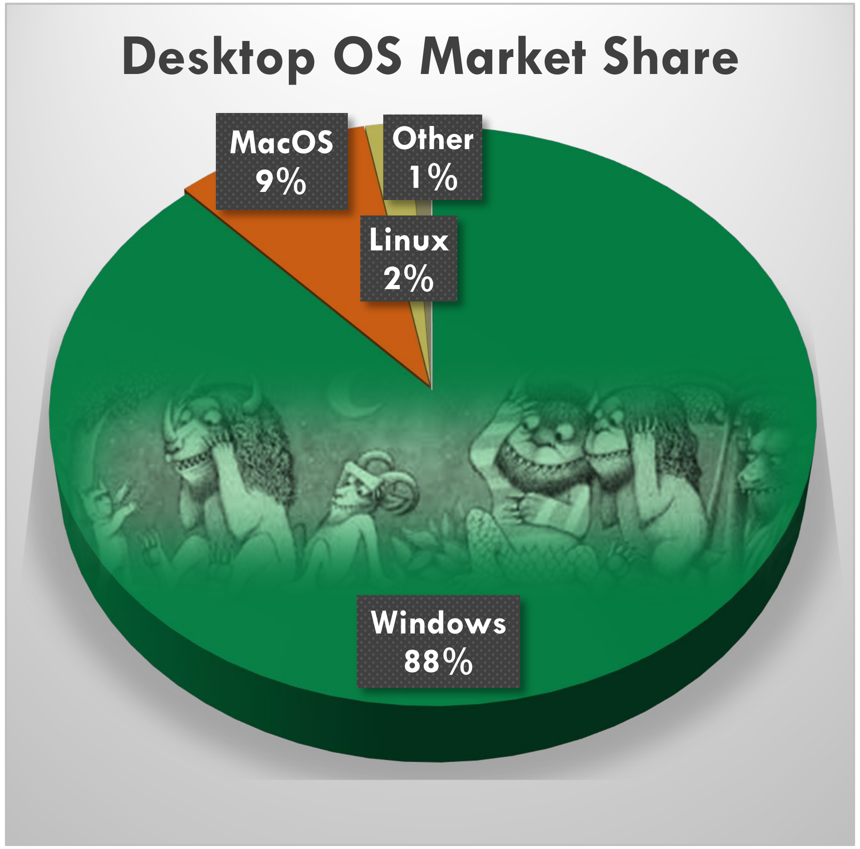


*Papers containing at least one experiment where at least <u>one</u> **<u>COMPLETE</u>** <u>non-benchmark</u> <u>application</u> for the indicated OS was rewritten & secured

Where the Wild Things Are

[RecordedFuture, CTA-2018-0327]





Soft(ware) Targets

- Windows/MacOS in mission-critical environments
  - "About 75% of control systems are on **Windows XP** or other **nonsupported** OSes." –Daryl Haegley, Office of Assistant Secretary of Defense for Energy, Installations and Environment
  - More than 25% of all government computers currently run an **outdated Windows** or **MacOS** operating system. [BitSight, 6/1/17]
  - DHS, Coast Guard, and Secret Service currently store top secret information on **outdated Windows 2003** servers. [OIG-18-56, 3/1/18]
  - Hundreds of *satellites* run **Windows 95** and/or are controlled by **Windows Mobile** devices.

20 Widespread Classes of CFI Compatibility Problems

Compatibility Metric	Real-world Software Examples
Function Pointers	7-Zip, Adobe Reader, Apache, Calculator, Chrome, Dropbox, Firefox, JVM, …
Callbacks	7-Zip, Adobe Reader, Apache, Calculator, Chrome, Dropbox, Firefox, JVM, …
Dynamic Linking	7-Zip, Adobe Reader, Apache, Calculator, Chrome, Dropbox, Firefox, JVM, …
Delay-Loading	Adobe Reader, Calculator, Chrome, Firefox, JVM, MS Paint, MS Powerpoint, …
Exporting/Importing Data Symbols	7-Zip, Apache, Calculator, Chrome, Dropbox, Firefox, MS Paint, MS Powerpoint, …
Virtual Functions	7-Zip, Adobe Reader, Calculator, Chrome, Dropbox, Firefox, JVM, Notepad, …
Writable Vtables	programs with UI's based on GTK+ (Linux) or COM (Windows)
Tail Calls	programs compiled with tail-call optimization (e.g., -O2 or /O2)
Switch-Case Statements	7-Zip, Adobe Reader, Apache, Calculator, Chrome, Dropbox, Firefox, JVM, …
Returns	almost every benign program
Unmatched Call/Return Pairs	Adobe Reader, Apache, Chrome, Firefox, JVM, MS PowerPoint, Visual Studio, …
Exceptions	7-Zip, Adobe Reader, Apache, Calculator, Chrome, Dropbox, Firefox, JVM, …
Calling Conventions	almost every program has functions
Multithreading	7-Zip, Adobe Reader, Apache, Calculator, Chrome, Dropbox, Firefox, JVM, …
TLS Callbacks	Adobe Reader, Chrome, Firefox, MS Paint, TeXstudio, UPX
Position-Independent Code	7-Zip, Adobe Reader, Apache, Calculator, Chrome, Dropbox, Firefox, JVM, …
Memory Management	7-Zip, Adobe Reader, Apache, Chrome, Dropbox, Firefox, MS PowerPoint, …
JIT Code	Adobe Flash, Chrome, Dropbox, Firefox, JVM, MS PowerPoint, PotPlayer, …
Self-Unpacking	programs decompressed by self-extractors (e.g., UPX, NSIS)
Runtime API Hooking	Microsoft Office, including MS Excel, MS PowerPoint, etc.

ConFIRM CFI Compatibility Benchmark Suite

Control-Flow Integrity Relevance Metrics (ConFIRM):

https://github.com/SoftwareLanguagesSecurityLab/Confirm

ConFIRM: Control-Flow Integrity Relevance Metrics

https://github.com/SoftwareLanguagesSecurityLab/Confirm

	LLVM (Windows)						LLVM (Linux)						
Test	CFI	ShadowStack	MCFG	OFI	Reins	GCC-VTV	CFI	ShadowStack	MCFI	πCFI	πCFI (nto)	PathArmor	Lockdown
fptr	6.35%	⚠	20.13%	4.35%	4.08%	⚠	6.97%	⚠	✗	-14.00%	-13.79%	⚠	174.92%
callback	⚠	⚠	⚠	128.39%	114.84%	⚠	⚠	⚠	✗	✗	✗	⚠	✗
load_time_dynlnk	2.74%	⚠	8.83%	3.36%	2.66%	⚠	1.33%	⚠	30.83%	31.52%	34.05%	74.54%	1.45%
run_time_dynlnk	⚠	⚠	17.63%	12.57%	11.48%	⚠	4.44%	⚠	✗	✗	✗	1,221.48%	✗
delay_load	N/A	N/A	8.16%	3.61%	✗	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
data_symbl	✓	⚠	✓	✓	✗	✓	✓	⚠	✓	✓	✓	✓	✓
vtbl_call	5.62%	⚠	27.71%	35.94%	31.17%	33.56%	5.94%	⚠	✗	-8.19%	-9.31%	⚠	227.82%
code_coop	⚠	⚠	⚠	✓	✗	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠
tail_call	6.17%	⚠	9.51%	0.05%	0.05%	⚠	6.82%	⚠	✗	-17.69%	-17.37%	⚠	178.06%
switch	-5.80%	⚠	3.51%	22.82%	17.69%	⚠	-6.93%	⚠	-29.01%	-27.19%	-28.46%	⚠	85.85%
ret	⚠	18.04%	⚠	49.34%	48.49%	⚠	⚠	20.88%	70.72%	72.40%	71.52%	⚠	106.71%
unmatched_pair	⚠	⚠	⚠	✓	✓	⚠	⚠	⚠	✓	✓	✓	⚠	⚠
signal	✓	⚠	✓	✗	✗	✓	✓	⚠	✓	✓	✓	✗	✓
cppeh	✓	⚠	✓	✓	✗	✓	✓	⚠	✓	✓	✓	✗	✓
seh	✓	⚠	✓	✓	✗	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
veh	⚠	⚠	⚠	✓	✗	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
convention	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓
multithreading	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠
tls_callback	N/A	N/A	N/A	✓	✗	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
pic	✓	✓	✓	⚠	⚠	✓	✓	✓	✓	✓	✓	✓	✓
mem	⚠	⚠	⚠	⚠	⚠	⚠	⚠	⚠	✗	✗	✗	✓	✗
jit	⚠	⚠	⚠	✗	✗	⚠	⚠	⚠	✗	✗	✗	⚠	✗
unpacking	N/A	N/A	N/A	✗	✗	N/A	N/A	N/A	N/A	N/A	N/A	✗	✗
api_hook	⚠	⚠	⚠	✗	✗	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Major Findings

- Multithreading + Unmatched call/return pairs = Trouble
  - unmatched call/returns arise from: exceptions, tail-call optimization
  - cross-thread stack smashing beats all CFI defenses we tested
  - seems hard to fix without huge performance overheads
- Runtime Code Generation
  - more prevalent than generally expected
    - rise of JIT-compiled languages, runtime hooking, self-extracting components
  - most RCG is beyond the reach of all CFI algorithms
- Questionable whether SPEC CPU adequately tests CFI performance
  - SPEC CPU benchmarks chosen/designed to test <u>CPU speeds</u>
  - Operation profiles prioritize opcodes that bottleneck <u>non-CFI software</u>
  - Mostly simple control-flow graphs

CFI Performance Measurement Problems

SPEC CPU Benchmark	CFI Solution									**Benchmark Correlation**
	MCFG	**Reins**	**GCC-VTV**	**LLVM-CFI**	**MCFI**	**πCFI**	**πCFI** (nto)	**PathArmor**	**Lockdown**	
perlbench				2.4	5.0	5.0	5.3	15.0	150.0	**0.09**
bzip2	−0.3	9.2		−0.7	1.0	1.0	0.8	0.0	8.0	**−0.12**
gcc					4.5	4.5	10.5	9.0	50.0	**0.02**
mcf	0.5	9.1		3.6	4.5	4.5	1.8	1.0	2.0	**−0.39**
gobmk	−0.2			0.2	7.0	7.5	11.8	0.0	43.0	**−0.09**
hmmer	0.7			0.1	0.0	0.0	−0.1	1.0	3.0	**0.33**
sjeng	3.4			1.6	5.0	5.0	11.9	0.0	80.0	**−0.03**
h264ref	5.4			5.3	6.0	6.0	8.3	1.0	43.0	**−0.09**
libquantum				−6.9	0.0	−0.3	−1.0	3.0	5.0	**0.51**
omnetpp	3.8		5.8		5.0	5.0	18.8			**−0.52**
astar	0.1		3.6	0.9	3.5	4.0	2.9		17.0	**0.92**
xalancbmk	5.5		24.0	7.2	7.0	7.0	17.6		118.0	**0.94**
milc	2.0			0.2	2.0	2.0	1.4	4.0	8.0	**0.40**
namd	0.1		−0.1	0.1	−0.5	−0.5	−0.5	3.0		**0.98**
dealII	−0.1		0.7	7.9	4.5	4.5	4.4			**−0.36**
soplex	2.3		0.5	−0.3	−4.0	−4.0	0.9	12.0		**0.89**
povray	10.8		−0.6	8.9	10.0	10.5	17.4		90.0	**0.88**
lbm	4.2			−0.2	1.0	1.0	−0.5	0.0	2.0	**−0.22**
sphinx3	−0.1			−0.8	1.5	1.5	2.4	3.0	8.0	**0.31**
ConFIRM median	9.51	4.59	33.56	5.19	30.83	−11.10	−11.60	648.01	140.82	**0.36**

CFI vs. Runtime Code Generation

- CFI Fundamental Assumptions (Abadi et al., 2005)
  - Non-Writable Code (NWC)
  - Non-Executable Data (NXD)
- Most Modern Software Violates Both
  - Rise of Just-In-Time (JIT) Languages since 2005
    - Lua, JavaScript, Python, Java, Ruby, PHP, Erlang, Wasm, Lisp, Etherium, …
    - Everything on .NET, all Microsoft COM software, …
  - All self-unpacking components (e.g., cloud), installers (e.g., UPX), …
  - Many forms of dynamic loading & hooking (example: Microsoft Office)

Existing Solutions

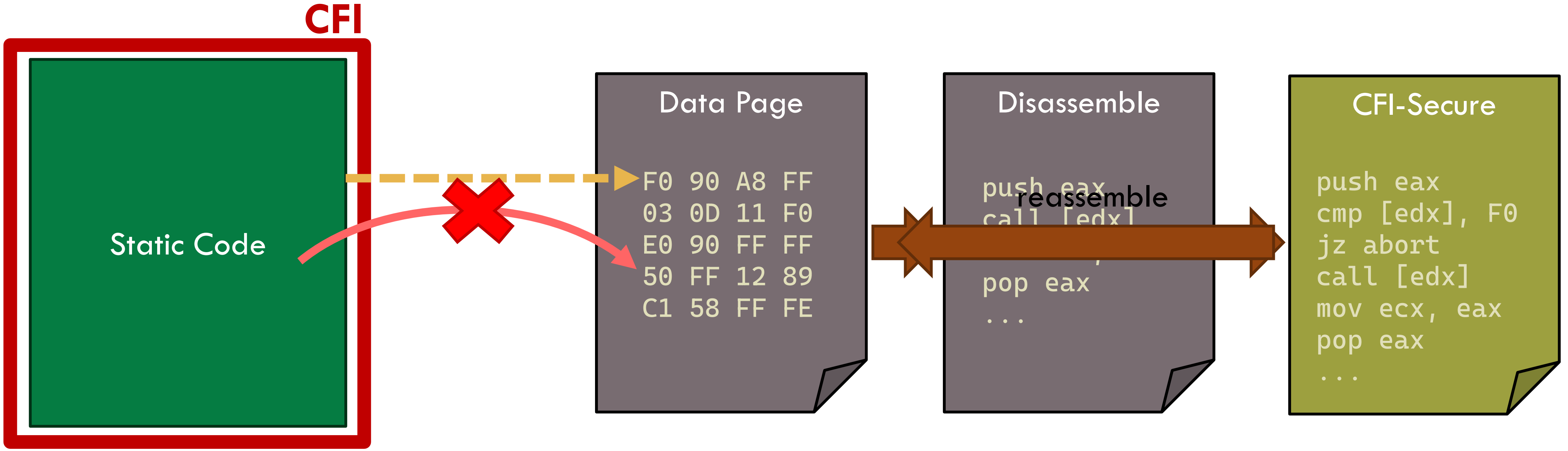
- Manually customize the code generator (RockJIT [Niu & Tan, CCS'14])
  - Extremely high maintenance burden
  - Only works for certain very specific code generation patterns (old JITs)
  - Incompatible with all modern software **(~9 years of 100% incompatibility)**
- Turn off all dynamic code generation
  - Massive performance hit **(e.g., 1600% overhead on JS Octane)**
  - Impossible for many products (.NET)
  - Introduces new compatibility & security problems

Real-world Example: Edge Browser

- Browser devs begin to realize that JIT is a huge security risk:
  - Over half of in-the-wild Chrome exploits from 2018-2021 abuse JIT vulnerabilities. [Mozilla Research, 2021]
  - More than 70% of the top programming languages are JIT-compiled, including JavaScript in almost all browsers.
- Microsoft turns off the JIT completely to enforce CFI in "secure mode"
  - *"As of Microsoft Edge 98, Control-flow Enforcement Technology (CET) and Arbitrary Code Guard (ACG) will be enabled in the renderer process when a site is in enhanced security mode. These additional mitigations* ***prevent dynamic code generation*** *in the renderer processes …"* [Microsoft Browser Vulnerability Research Lab, 2022]
- But JIT compilers are dynamic code generators (critical for performance)…

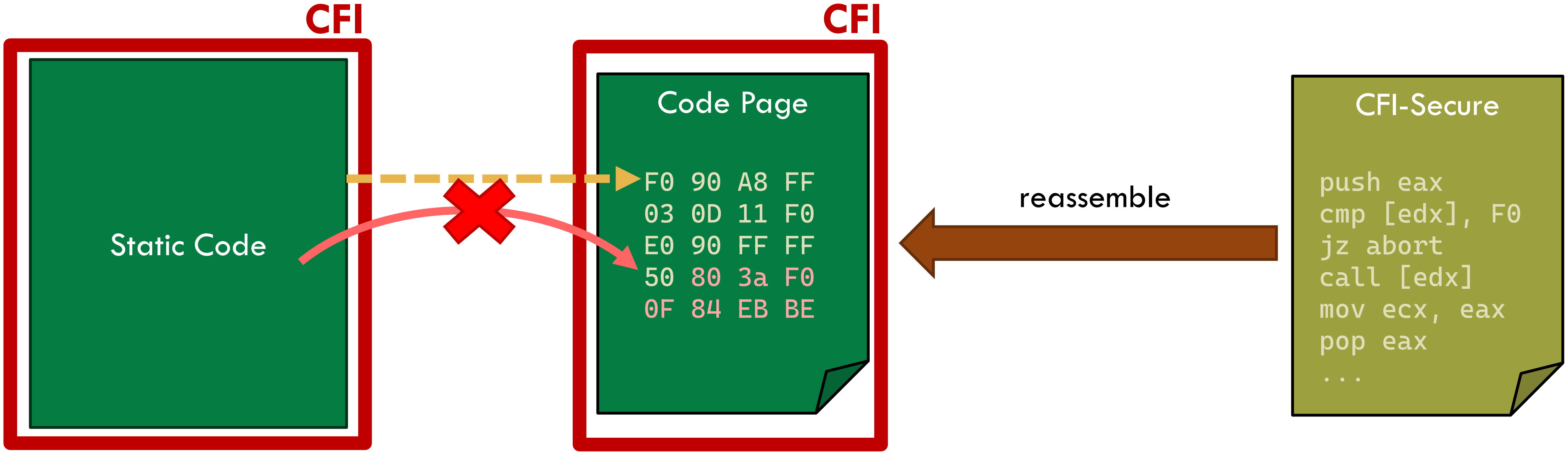
RENEW: Rewriting Newly Executable pages after Writes

E. Bauman, J. Duan, K.W. Hamlen, and Z. Lin, **"Renewable Just-In-Time Control-Flow Integrity,"** In *Proc. 26th Int. Sym. on Research in Attacks, Intrusions and Defenses (RAID)*, October 2023.



RENEW: Rewriting Newly Executable pages after Writes

Static+Dynamic Code Instrumentation
Dynamic Policy Enforcement



Challenges

- Disassembly + Reassembly must be *fast* and *secure*
  - Disassembly alone is provably undecidable in general!
- Must support *recursive (generational) dynamic code generation*
  - dynamic code may write new dynamic code
  - static code may edit dynamic code pages mid-execution
- Must support calls from dynamic code to static code
  - Static code pointer passed to dynamic code
  - Pointer might not target a CFI-sanctioned entry point!
- Real-world apps sometimes *read generated code as data* (ugh!)

Disassembly Undecidability


```
FF E0 5B 5D C3 0F
88 52 0F 84 EC 8B
```

- Disassemble this hex sequence
  - CISC disassembly is undecidable! [Fred Cohen, '86]

Valid Disassembly	
FF E0	jmp eax
5B	pop ebx
5D	pop ebp
C3	retn
0F 88 52 0F 84 EC	jcc
8B …	mov

Valid Disassembly	
FF E0	jmp eax
5B	pop ebx
5D	pop ebp
C3	retn
0F	db (1)
88 52 0F 84 EC	mov
8B …	mov

Valid Disassembly	
FF E0	jmp eax
5B	pop ebx
5D	pop ebp
C3	retn
0F 88	db (2)
52	push edx
0F 84 EC 8B …	jcc

Innovation: Superset Disassembly

Byte Sequence: FF E0 5B 5D C3 0F 88 B0 50 FF FF 8B

Disassembled **Invalid**

Hex
FF
E0
5B
5D
C3
0F
88
B0
50
FF
FF
8B

Included Disassembly
jmp eax
pop
L1: pop
retn
jcc
L2: mov
loopne
jmp L1
mov
jmp L2

Machine learning-based Disassembly Pruning

[Wartell, Zhou, Hamlen, Kantarcioglu, PAKDD'14]

- Insight: Distinguishing real code bytes from data bytes is a "noisy word segmentation problem".
  - Word segmentation: Given a stream of symbols, partition them into words that are contextually sensible. [Teahan, 2000]
  - Noisy word segmentation: Some symbols are noise (data).
- Machine Learning based disassembler
  - based on *k*th-order Markov model
  - Estimate the probability of the sequence B:

$$p(B|M_\alpha) = -\log \prod_{i=1}^{|B|} p(b_i|b_{i-k}^{i-1}, M_\alpha)$$

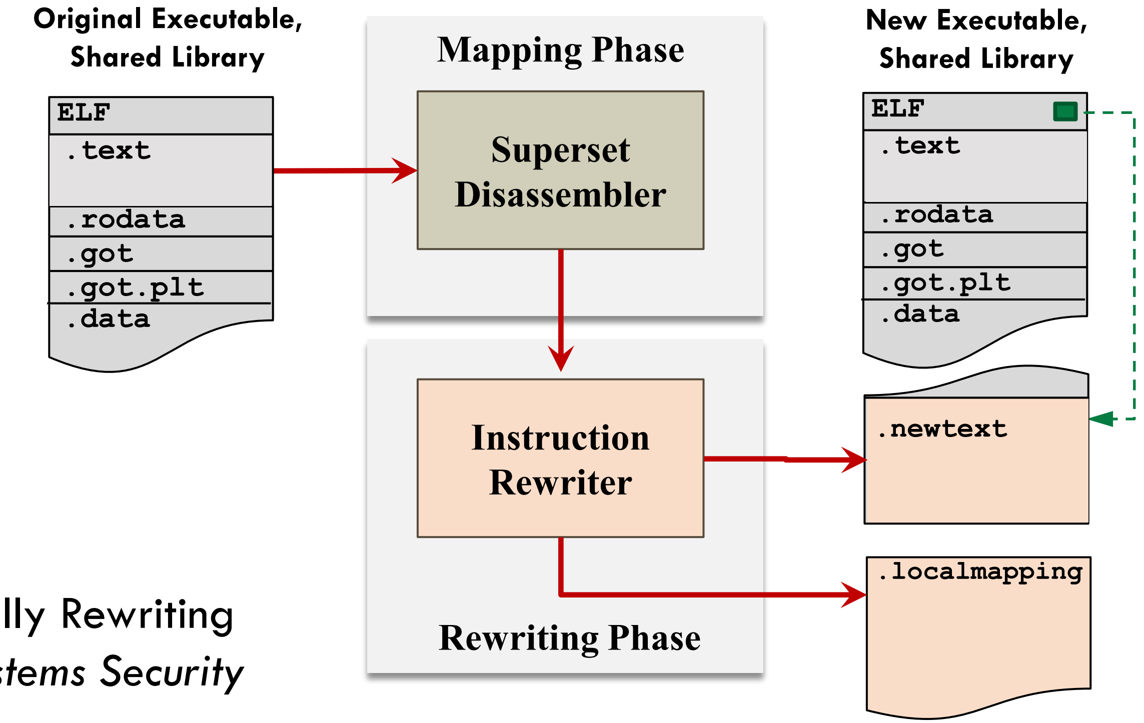
Wartell, Zhou, Hamlen, Kantarcioglu. "Shingled Graph Disassembly: Finding the Undecidable Path." PAKDD 2014.

Wartell, Zhou, Hamlen, Kantarcioglu, and Thuraisingham. "Differentiating code from data in x86 binaries." *ECML/PKDD* 2011.

Multiverse “Superset” Disassembler

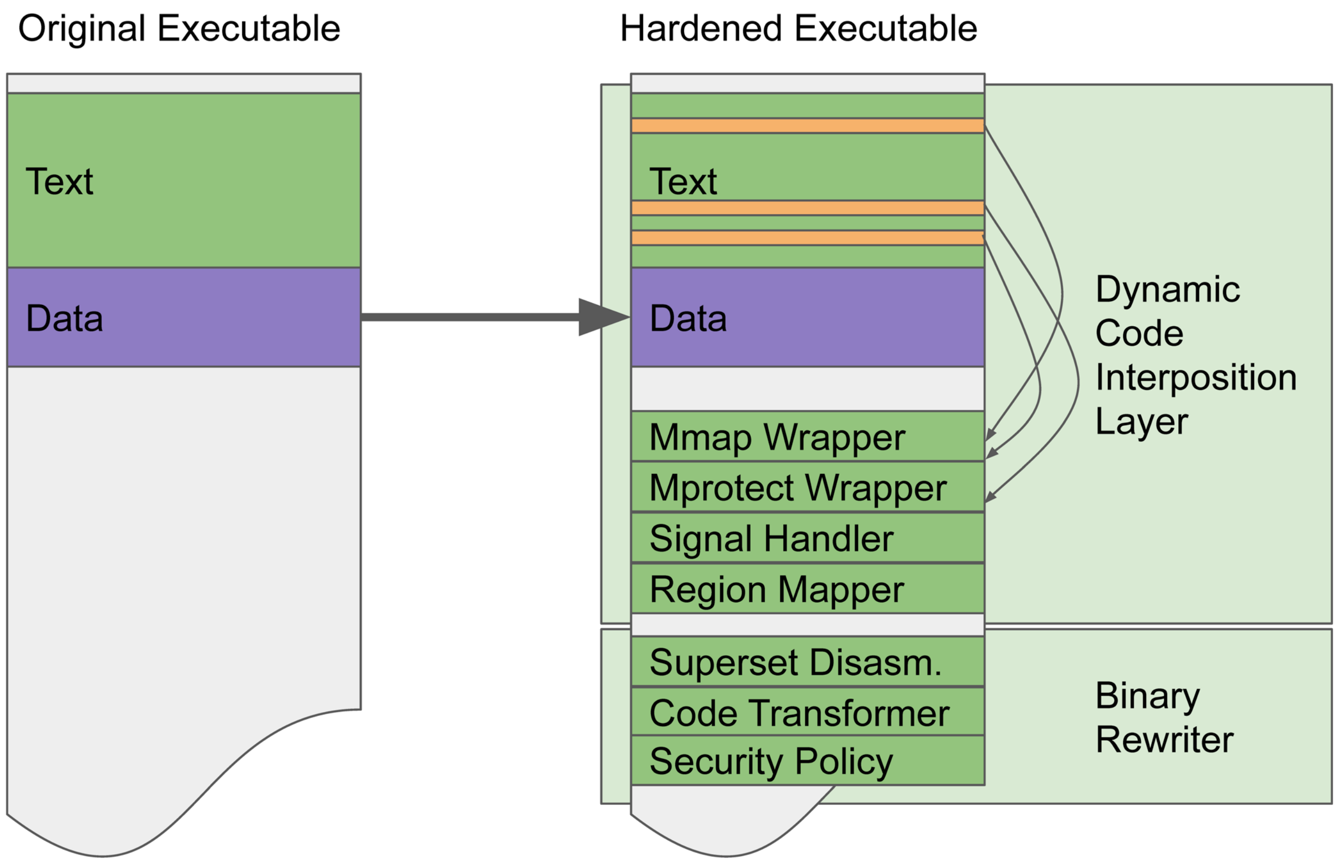
[Bauman, Lin & Hamlen, NDSS’18]

- Conservatively include *every possible* disassembly.
- Include and secure all of them.
- tested on 126 apps + 77 libs (all source-free)
- all application functionalities preserved
- 4-5x size increase of code segments (much smaller impact on overall file size)



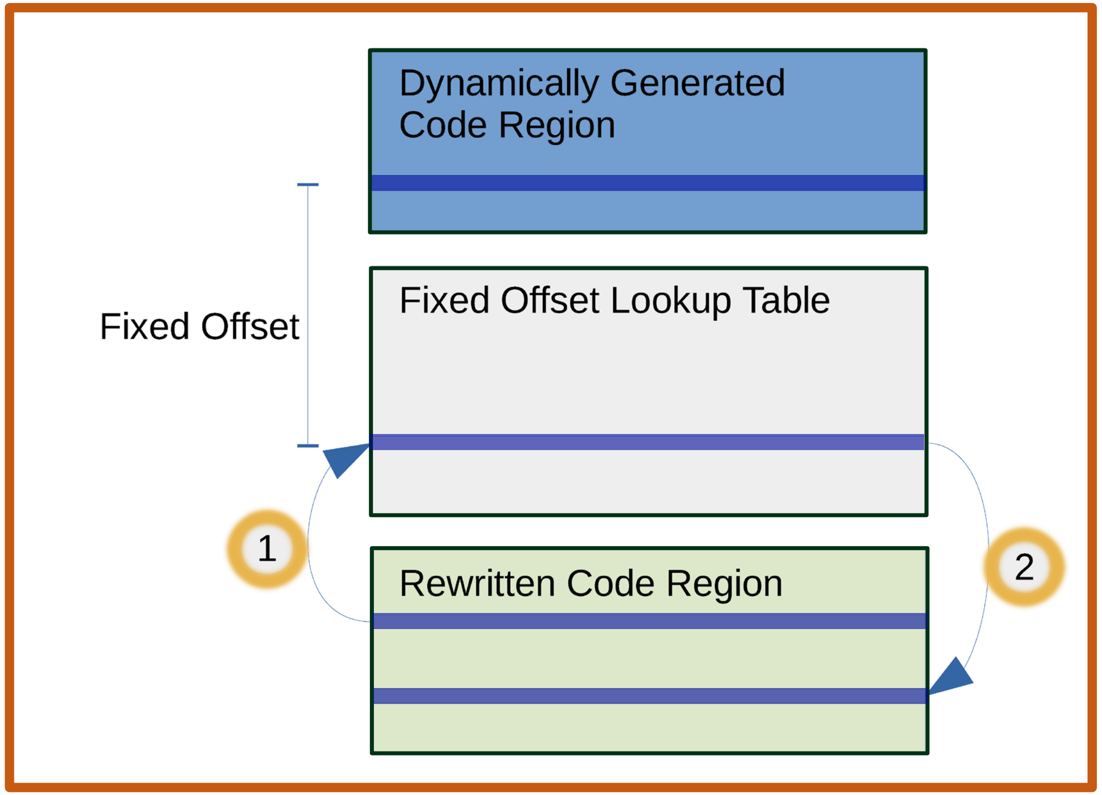
E. Bauman, Z. Lin, and K.W. Hamlen. “Superset Disassembly: Statically Rewriting x86 Binaries Without Heuristics”. In *Proc. Network & Distributed Systems Security (NDSS)*, 2018.

RENEW Overview

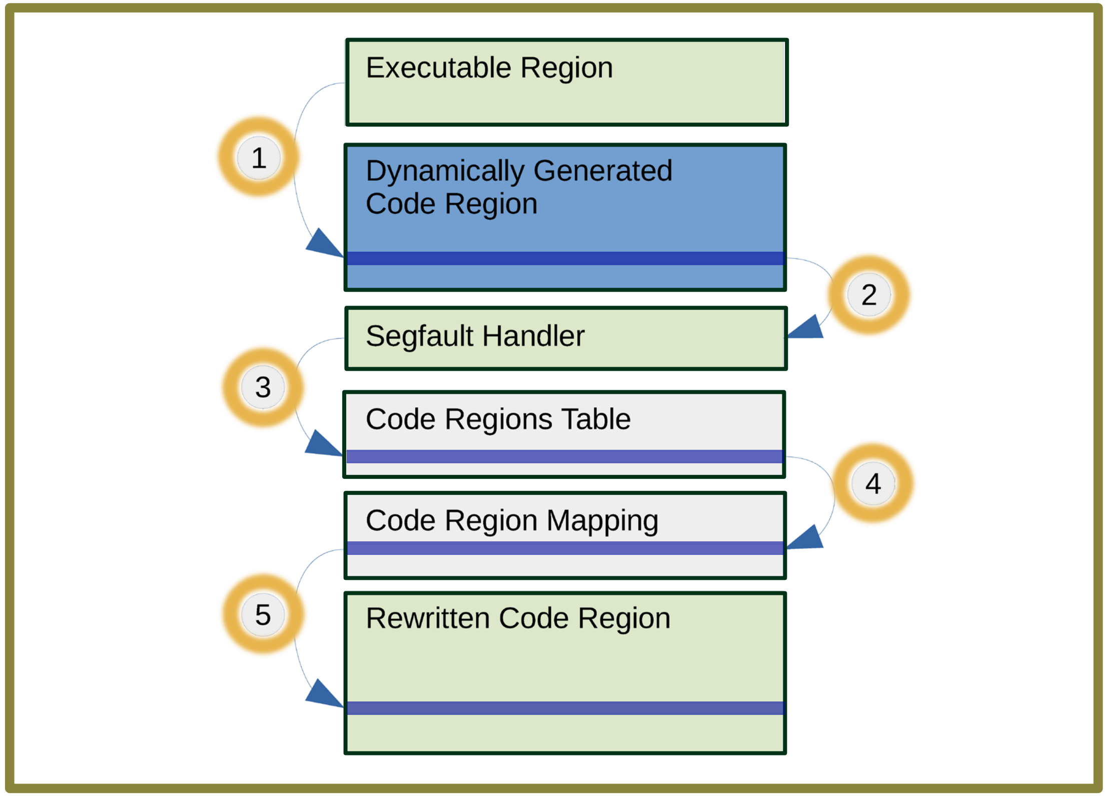


Optimization Strategy: Fast path + Slow path

Fast Path: Statically predictable flows



Slow path: Fall back to signal handlers



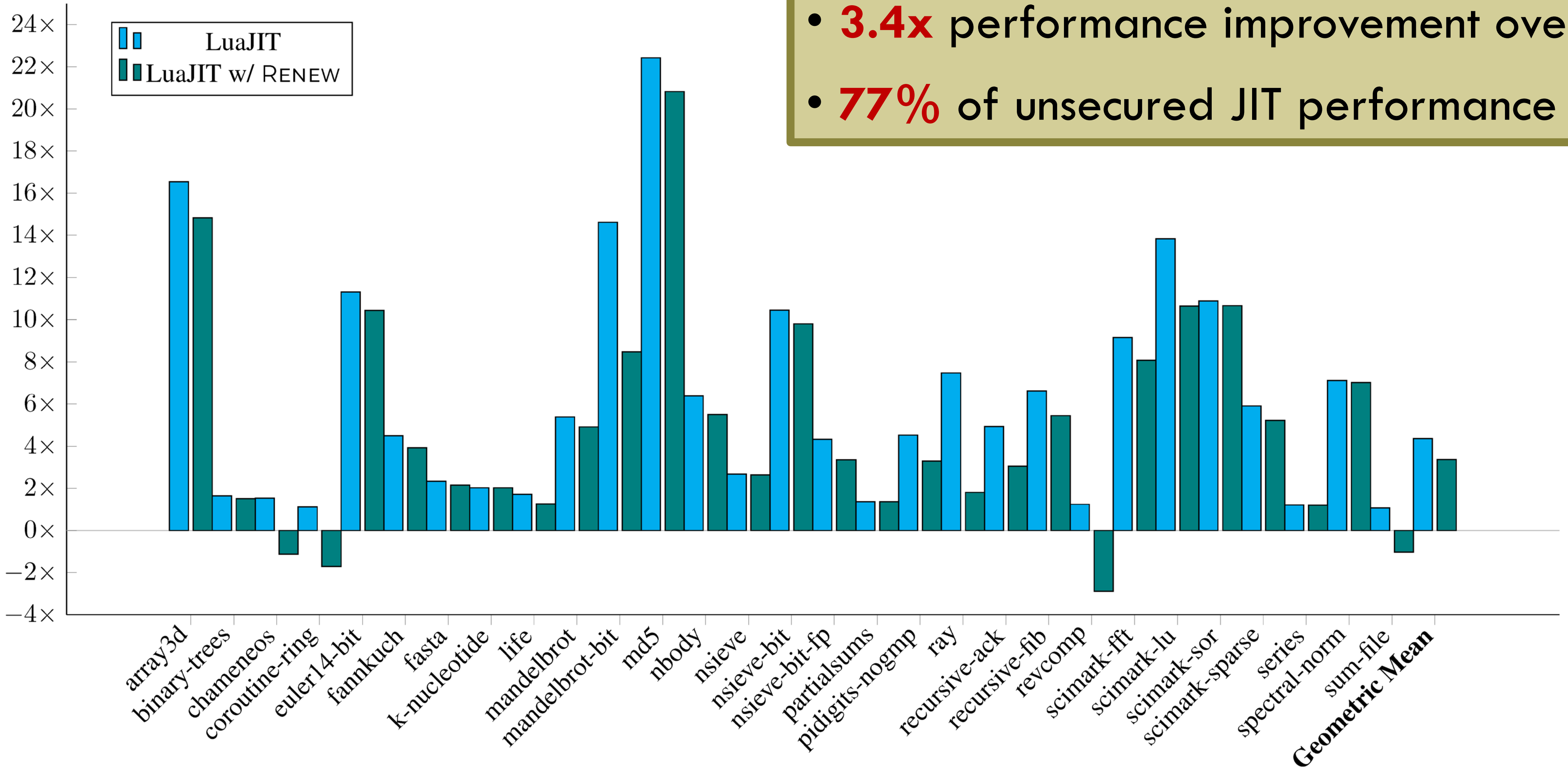
Proof-of-Concept Implementation

- Compact design and implementation
  - ~2000 lines of C code, plus in-lined assembly
  - includes entire disassembler, rewriter, interposition layer, etc.
  - included into target applications, so must be small
- Injected into target applications during compilation
  - shared library (`-Wl,-wrap=mmap -Wl,-wrap=mprotect`)
  - one-line change to application main function to call Renew initializer
- CFI static instrumentation of main app assumed
  - static CFI policy must permit Renew's static flows
  - Renew handles the dynamic flows
- Dramatically easier process than manual JIT redesign!

Evaluation

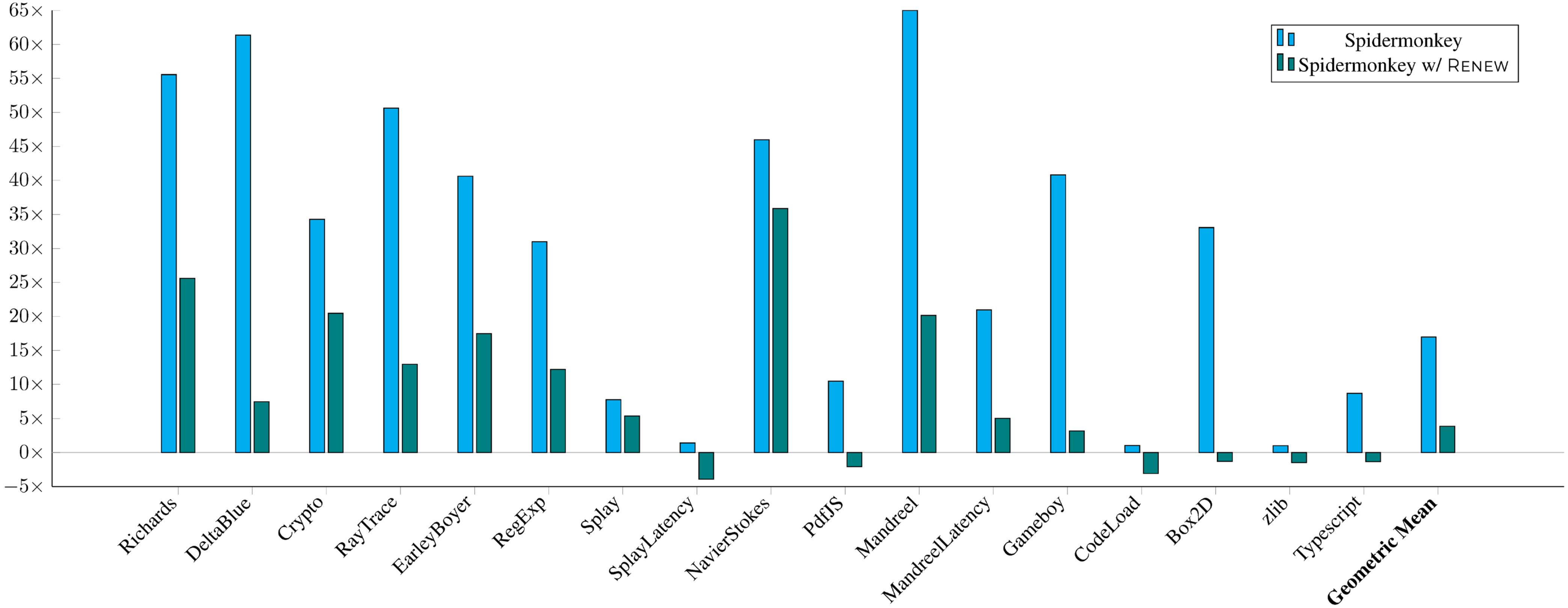
- Three main case-studies:
  - Lua (JIT compiler)
  - Firefox JavaScript (Spidermonkey JIT compiler)
  - UPX (installer / code unpacker)
- Completely different rewriting strategies
  - No common code generation patterns
  - Completely different control-flow patterns
  - Highly optimized, highly complex, high churn (most popular JITs and unpacker today, latest versions at time of implementation)
- No existing CFI solution works correctly on any of these target applications.

LuaJIT Evaluation Results



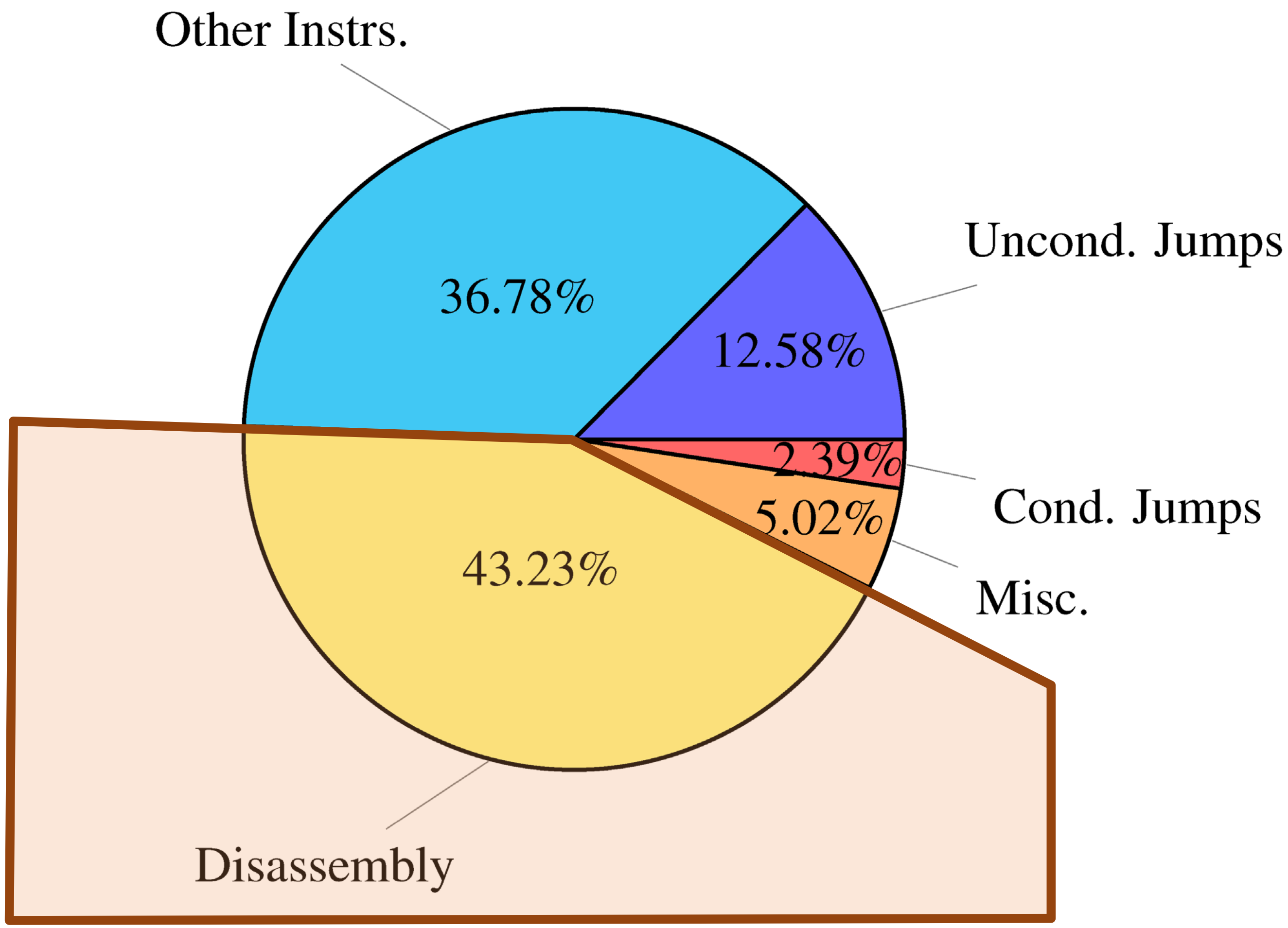
- **3.4x** performance improvement over JIT-off
- **77%** of unsecured JIT performance preserved

Firefox JS Evaluation



- **3.9x** performance improvement over JIT-off
- **23%** of unsecured JIT performance preserved

Overhead Breakdown



UPX Evaluation

- SPEC CPU benchmarks + GNU binutils apps packed using UPX
- Extremely difficult compatibility challenge
  - UPX uses a custom binary header format and IAT to save space
  - completely arbitrary code generation behavior (depends on packed code)
  - two highly compressed layers of unpacking
  - first layer defies conventional disassembly
- Results
  - **all tests worked out-of-the-box** (no changes to Renew required)
  - predictably high overhead (3.6x slowdown)
    - not really a fair performance test; we mainly wanted to test compatibility

Related Work Comparison

System	Year	SFI	CFI	Code-reuse Immunity	JIT	Packers	Source-agnostic
NaCl-JIT	2011	✓	✗	✓	✓	✗	✗
Librando	2013	✗	✗	⚠	✓	✗	✓
RockJIT	2014	✓	✓	✓	✓	✗	✗
SDCG	2015	✗	✗	✗	✓	✗	✗
JITScope	2015	✓	✓	✓	✓	✗	✗
JITGuard	2017	✗	✗	⚠	✓	✗	✗
RENEW	2022	✓	✓	✓	✓	✓	✓

⚠= Defense is diversity-based, so can be compromised by information disclosure.

THANK YOU!

Dr. Kevin Hamlen
Louis A. Beecherl, Jr. Distinguished Professor
Computer Science Department
Cyber Security Research and Education Institute
The University of Texas at Dallas

Software Attack Surface Reduction On The Fly

ONR Award N00014-21-1-2654