Toward Automated Forensic Analysis of Obfuscated Malware

Ryan J. Farley
George Mason University
Department of Computer Science

Committee: Xinyuan Wang, Hakan Aydin, Songqing Chen, Brian Mark

Overview

- The Need for Forensics
- Forensics Problems and Our Contribution
- Background
- Problem Model
- Challenges and Solutions
- Empirical Evaluation
- Conclusion

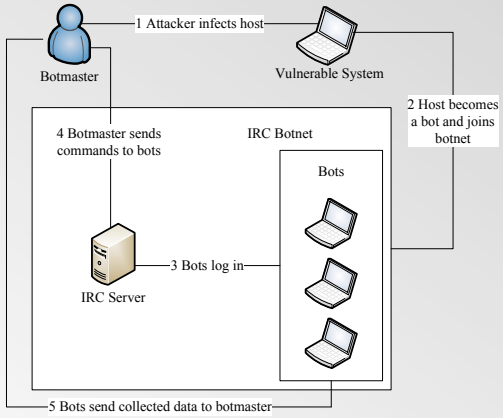
GEORGE MASON UNIVERSITY

The Need for Forensics

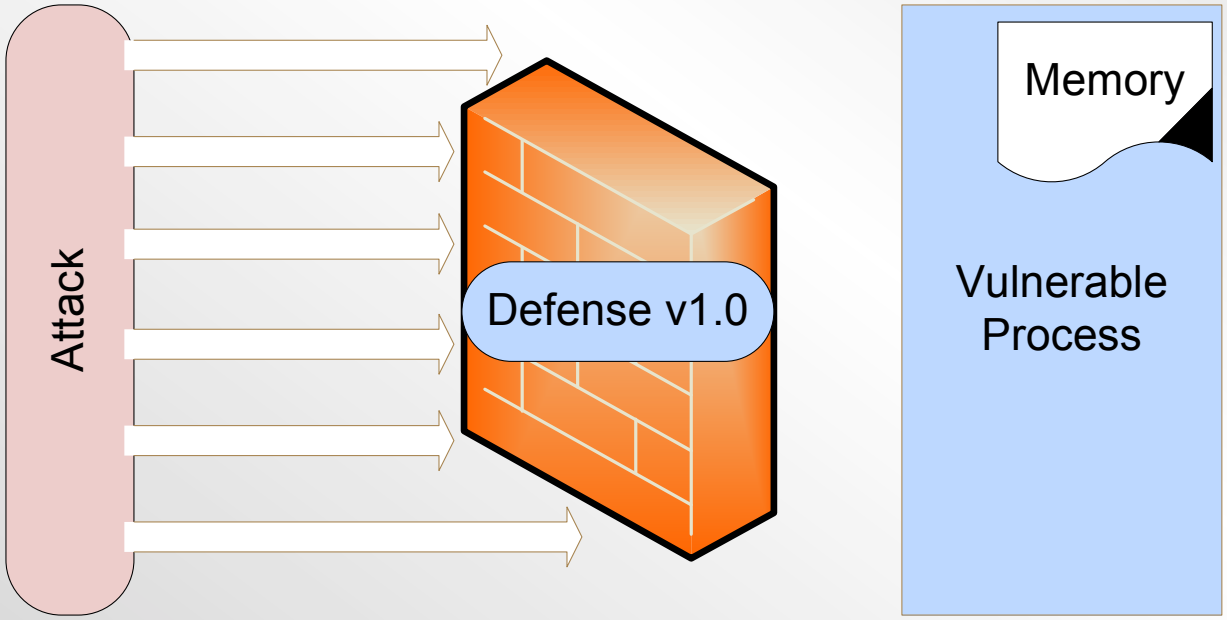
GEORGE MASON UNIVERSITY

Why?

- Malware is a serious threat
  - Internet of [Insecure] Things
  - Stuxnet, Regin
  - Christmas holiday tradition
  - Compromise is an eventuality
- Forensics seeks to understand the *how*
  - Embrace the ownage
  - Collect evidence, Analyze, Extrapolate
- Enables us to build better defenses

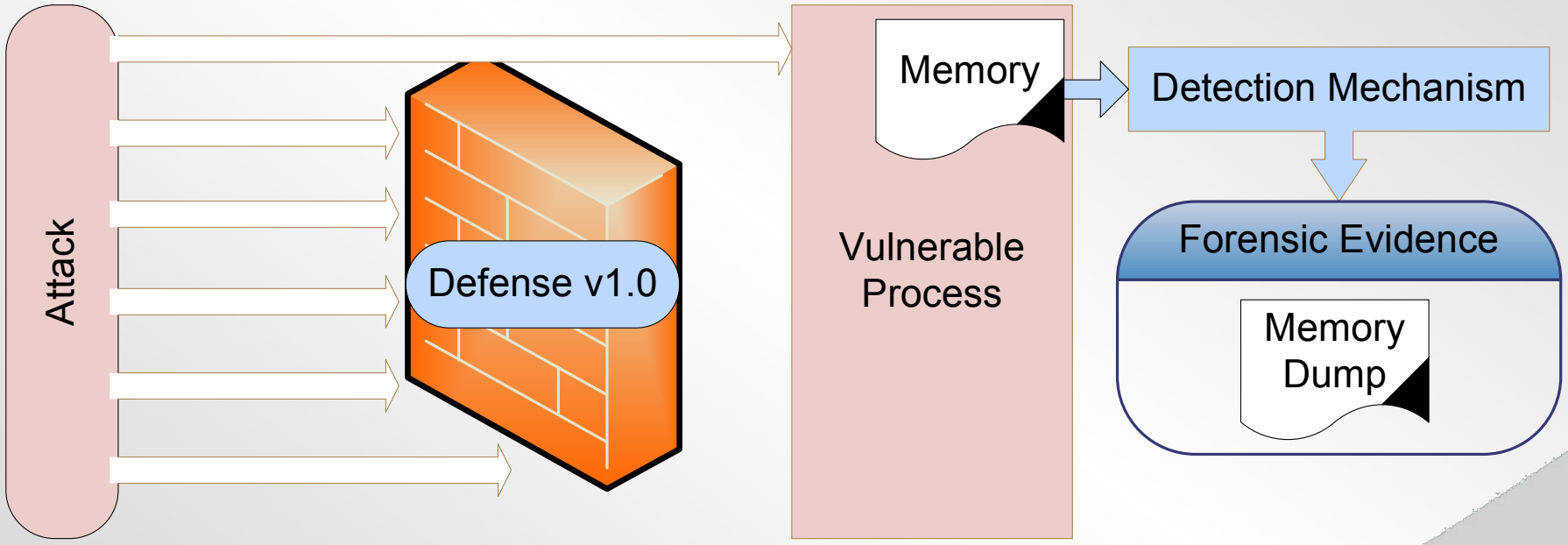


Scenario: Vulnerable Web Server



GEORGE MASON UNIVERSITY

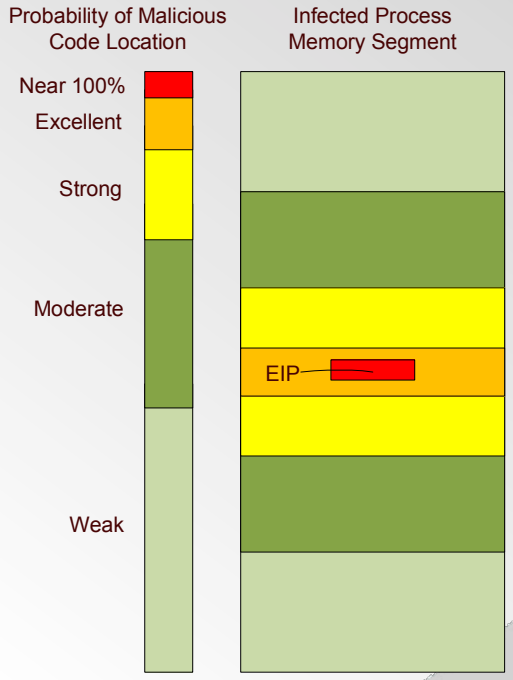
Scenario: Exploit, What Now?



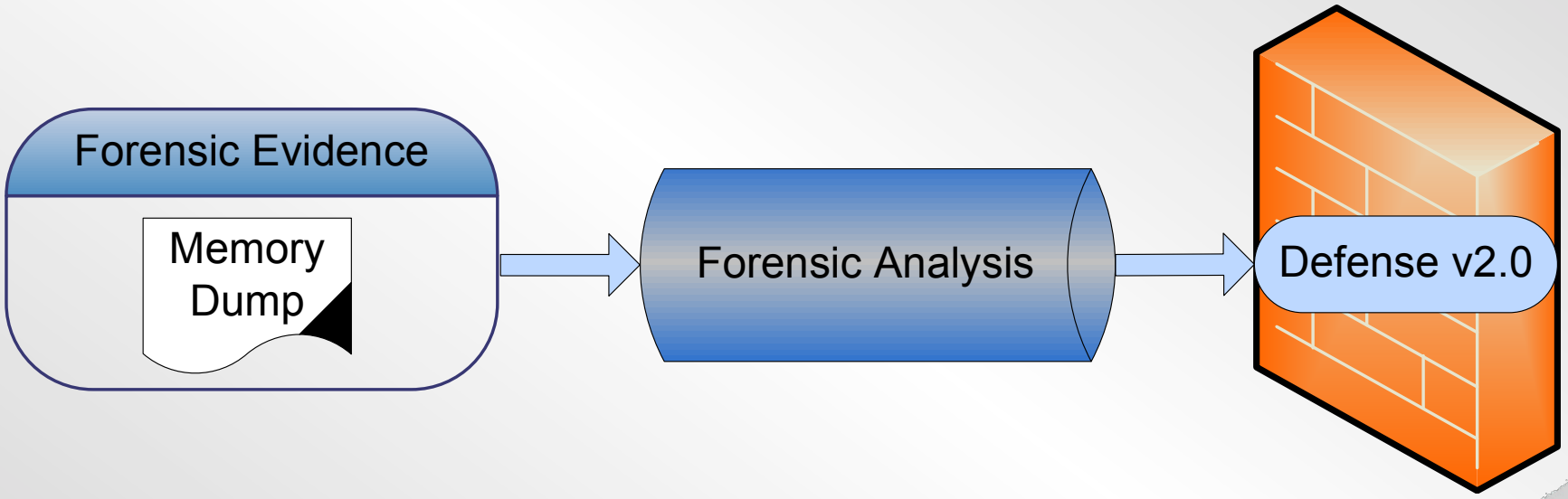
GEORGE MASON UNIVERSITY

Scenario: What Now?

- Upon first non-self system call
  - Attack code fragments remain in memory
- Packing, self-modification, armoring
  - Staged C2
    - Can the fragments reveal clues?
  - Robust system needed to generically model execution



Scenario: Build Better Defense



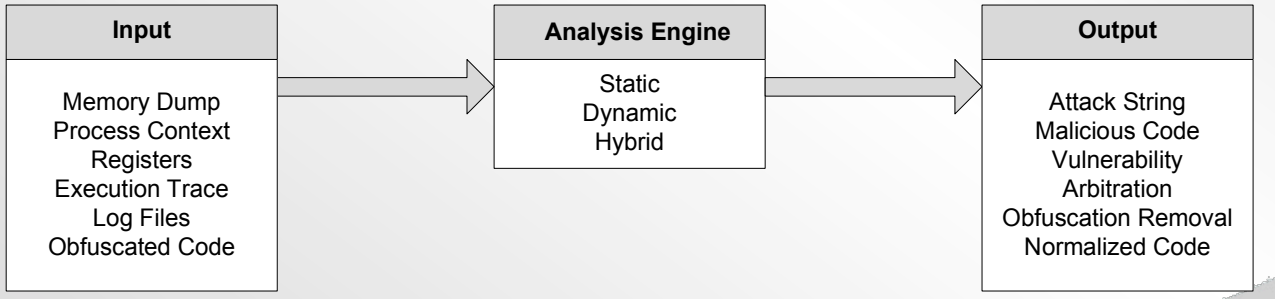
GEORGE MASON UNIVERSITY

Forensics Problems and Our Contribution

GEORGE MASON UNIVERSITY

Problem

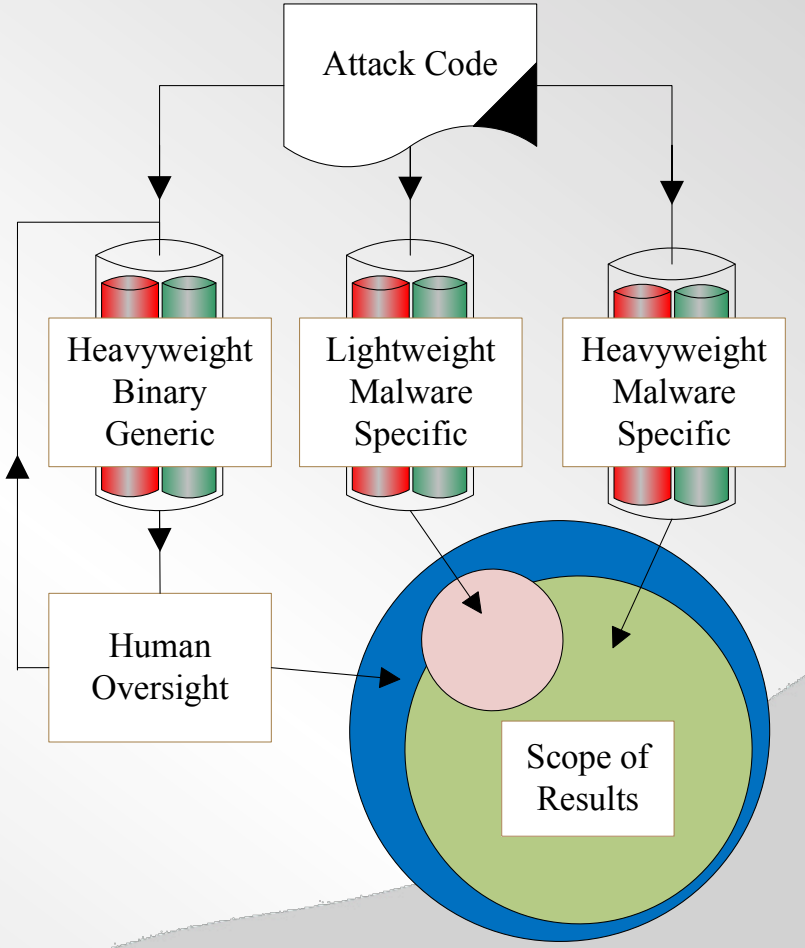
- Need to automate forensic response upon detection in memory
  - Avoid substantial manual effort
    - Automatically recover malcode
    - Extract/unpack/recover attack code
  - Memory dump, transient artifacts



GEORGE MASON UNIVERSITY

Problem

- Human oversight is costly
- Trade-off between
  - Generic binary
  - Malware specific
- Need
  - Automated generic malware tool that approaches detail from generic binary tools



GEORGE MASON UNIVERSITY

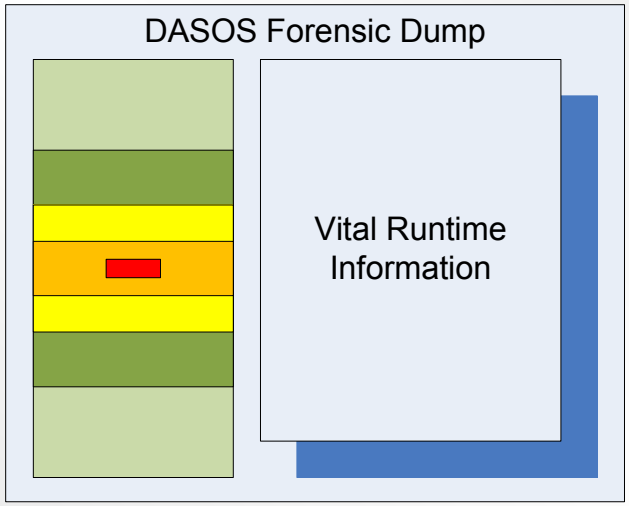
Motivation, Existing Tools

- Only work within known boundaries
  - Typically exclude support for code fragments
    - e.g., shellcode
  - Things get messy without given boundaries
    - e.g., arbitrary byte streams
- Do not generically handle:
  - Malformed, Misaligned
  - Obfuscated, Armored
  - Too specific or too abstracted

GEORGE MASON UNIVERSITY

Solution: CodeXt

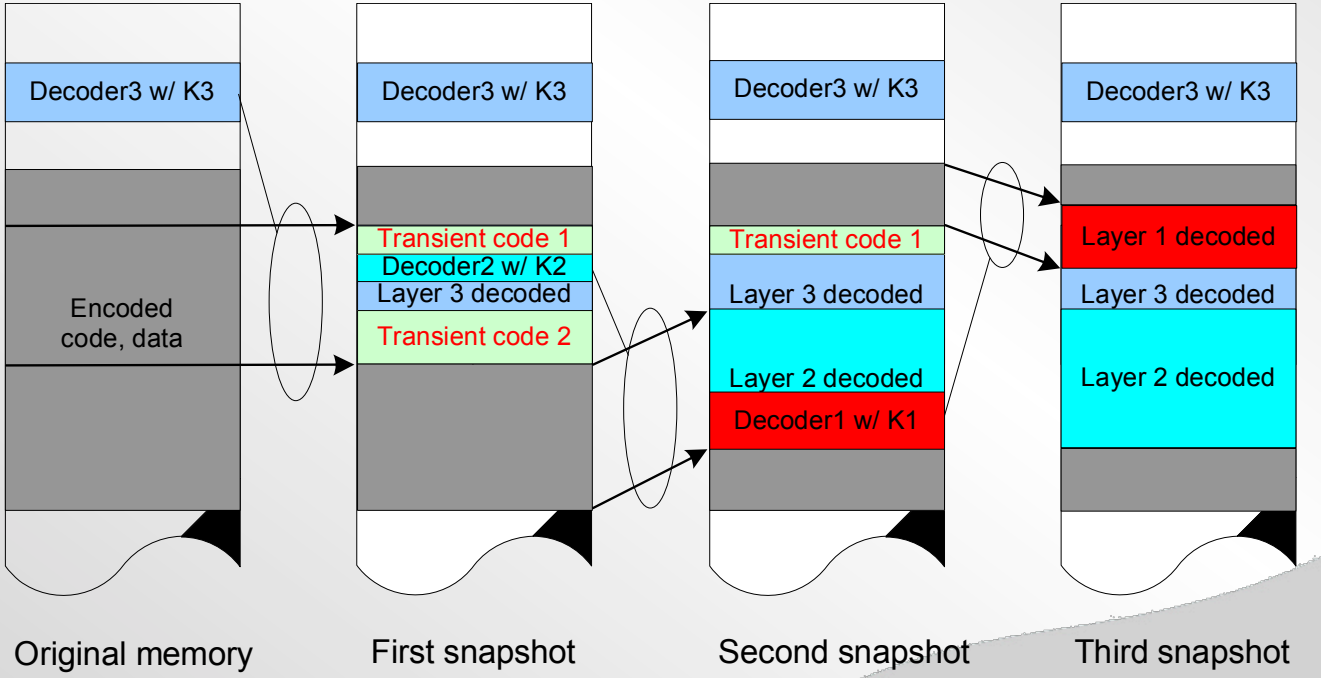
- Discovers executable code within memory dump
  - Upon real-time detection



GEORGE MASON UNIVERSITY

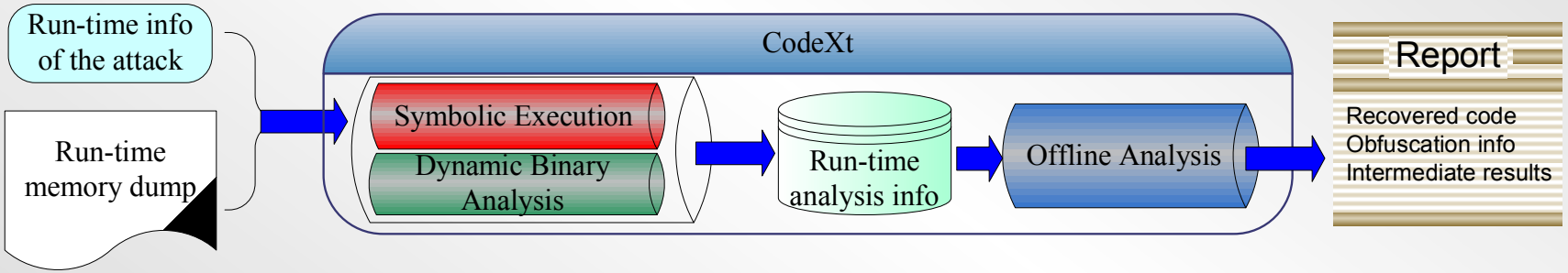
Solution: CodeXt

- Extracts packed or obfuscated malcode
  - First to generically handle Incremental and Shikata-Ga-Nai



Solution: CodeXt

- Uses data-flow analysis (taint tracking)
  - Finds attack string within network traffic
- Models both shellcode and full executables



- Framework built upon S2E
  - Selective means QEMU vs. KLEE (LLVM)

GEORGE MASON UNIVERSITY

Background

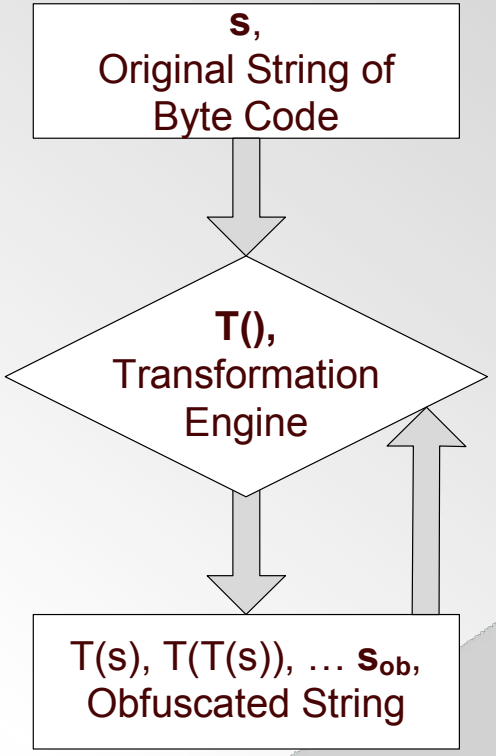
Background

- S2E, Selective Symbolic Execution
  - KLEE for symbolic
  - QEMU for concrete
    - We extended QEMU to detect system calls
- KLEE
  - Expressive IR allows low level operations
    - Down to the bit
  - States = Shadow Memory + Constraints
  - Memory = Expressions
    - Even concrete values are expressions

GEORGE MASON UNIVERSITY

Attack Code vs. Attack String

- Attack string:
  - Crafted input to the process
  - May include non-code
- Attack code:
  - Executed within process
  - May include immediate values (data)
- Removing layers of obfuscation
  - How many, and by what function?
  - What about self-destructive code?



GEORGE MASON UNIVERSITY

Framing the Problem

- Assumptions
  - All malicious code exists within dump
  - Malicious code has not overwritten itself destructively
- Requirements
  - No code semantics known
  - Coding conventions irrelevant
  - Capable of accuracy with self-modifying code
  - Capable of modeling network-based server applications

GEORGE MASON UNIVERSITY

CodeXt Output

- Instruction Trace of executed instructions
  - Grouping of fragments into chunks
  - Reveals original and unpacked malcode
  - Assisted by a translation trace
- Data Trace of memory writes
  - Intelligent memory update clustering
  - Multi-layer snapshots
- Call Trace of system calls
  - With CPU context

GEORGE MASON UNIVERSITY

Data-flow Analysis Output

- For each labeled byte
  - Follow propagration
  - Generate trace
  - Generate memory map
- Add events that qualify as success
  - EIP contains tainted values

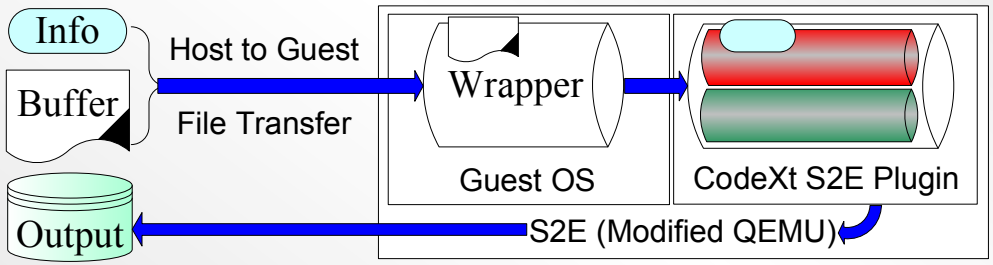
GEORGE MASON UNIVERSITY

Problems + Challenges + Solutions

GEORGE MASON UNIVERSITY

Handling Byte Streams

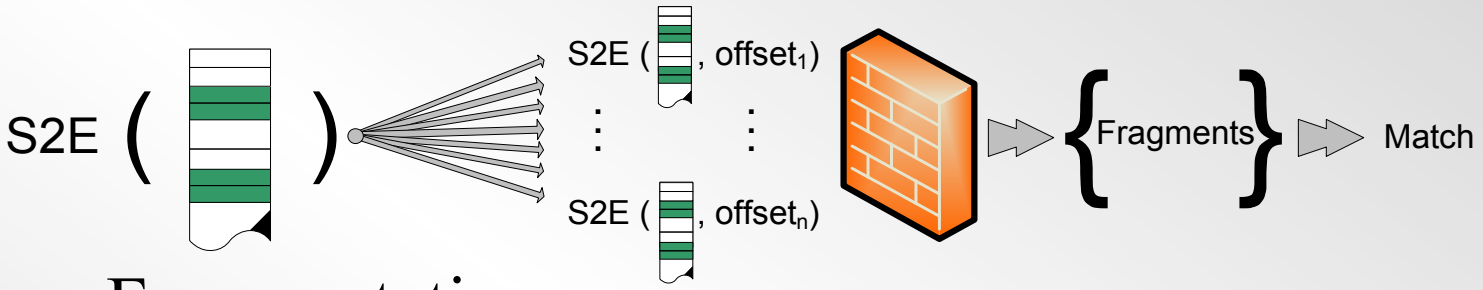
- S2E expects well structured binaries
  - We wrap the binary for execution



- S2E uses basic block granularity
  - Our modified QEMU translation returns more info
  - We leverage translation and execution hooks to verify

GEORGE MASON UNIVERSITY

Code Fragments



- Fragmentation
  - Clustering into Chunks, adjacency, execution trace
- Density
  - Usage: Executed/Range
  - Overlay: Unique executed/Range over snapshots
- Enclosure
  - Continuous executable bytes adjacent to end

GEORGE MASON UNIVERSITY

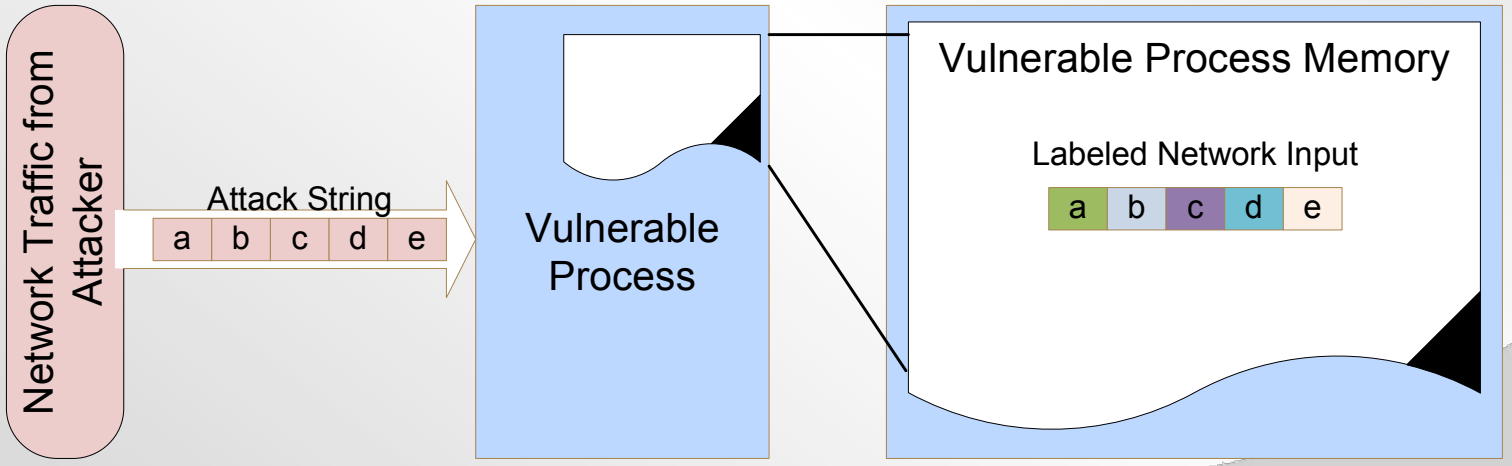
Defeating Obfuscation

- FPU instructions, fnstenv
  - Added small change to QEMU to comply
- Intra-basic block self-modification
  - We know address range of each translated block
  - During execution we track writes
  - If any write is to same block we retranslate block
- Emulator detection
  - Tested for a set of obscure instructions used as canaries

GEORGE MASON UNIVERSITY

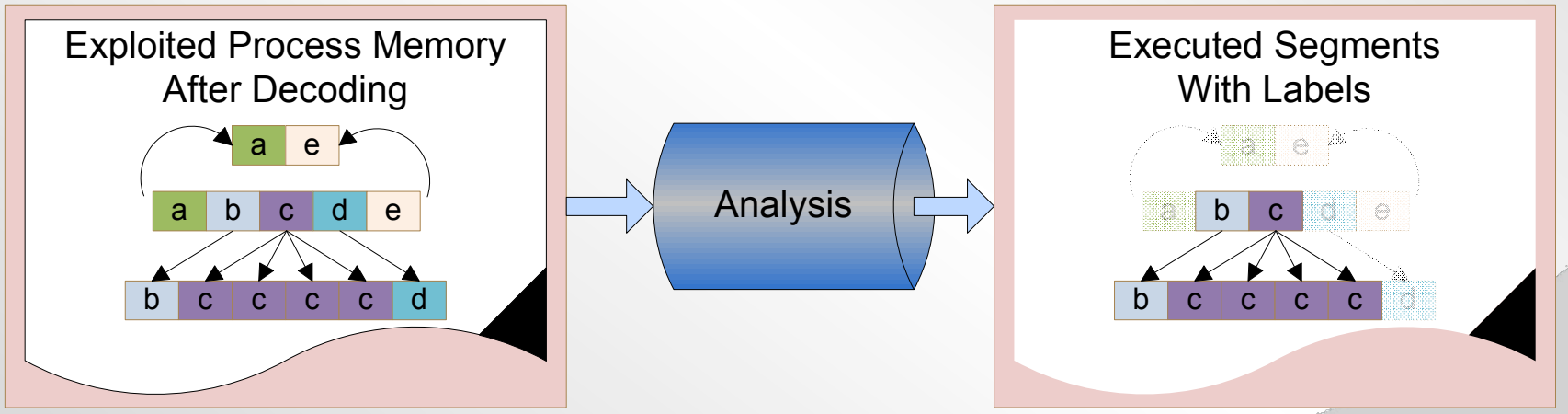
Multipath, Arbitrary Bytes

- Multipath Execution
  - Existing trace tool manages path merging
  - KLEE manages state forking and resources
- Mark Arbitrary Bytes as Symbolic



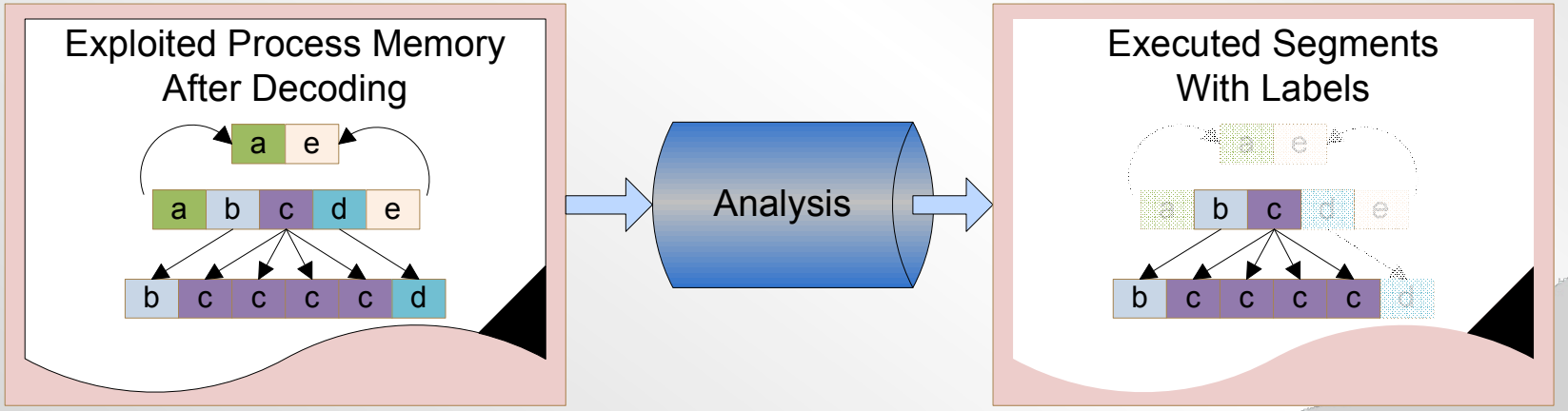
Executing Symbolic Code

- Taint labels can be search upon events
  - KLEE prefers constraints over solving
    - Constraint cleanup
  - Silent concretization



Executing Symbolic Code, con't

- Data-flow validity, intermingled code
  - Symbolic EIP
  - Periodic or triggered custom simplifier
    - Inheritance enforcer
    - Bit-wise and mov



Executable Modeling

- OS introspection
  - Snag CR2
- Load and link overhead
  - 95,000 instructions to ignore
  - Canary
- Real-time attacks
  - Buffer overflow
  - Sockets
  - SSL

GEORGE MASON UNIVERSITY

Empirical Evaluation

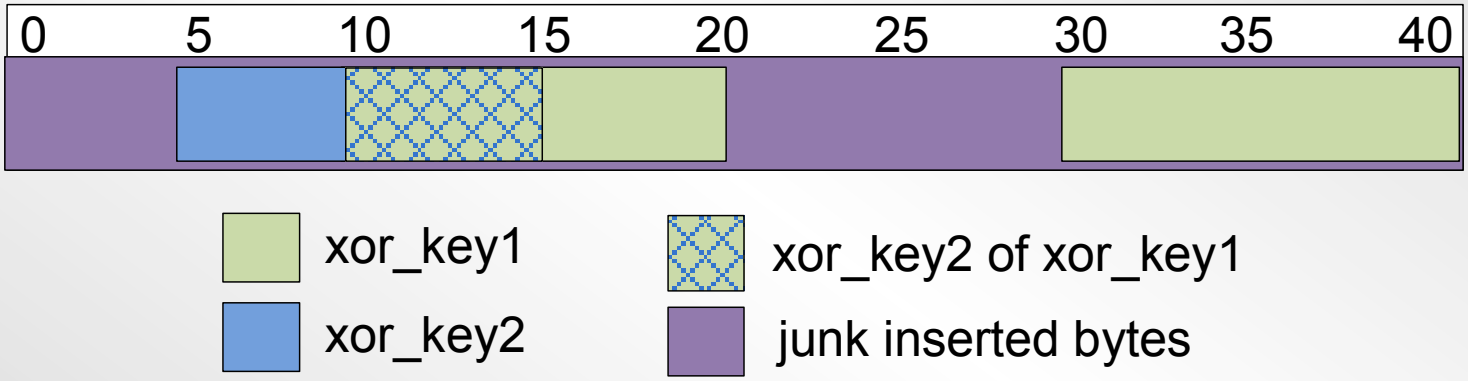
GEORGE MASON UNIVERSITY

Experiments, Part 1

- Hidden code search
  - 1KB to 100KB buffers, 40B to 80B shellcodes
  - Filled with either null, live-capture, or random bytes
  - Varied assistance data: EIP, EAX, both, neither
- Accuracy
  - De-obfuscation, Anti-emulation detection
  - Various packers mentioned in previous research
  - In-shop: Junk code insertion, Ranged xor, Incremental
- Symbolic Branching

GEORGE MASON UNIVERSITY

Multi-Layered Encoders



Publicly Available and Advanced

Technique	Extracted?	Technical Challenge
Junk code insertion	Yes	None
Ranged XOR	Yes	None
Multi-layer combinations of above	Yes	Multi-layer encoding
Incremental	Yes	Live annotation required Block based feedback key
ADMmutate	Yes	Complicated code combinations
Clet	Yes	Polymorphism
Alpha2	Yes	None
MSF `call+4 dword` XOR	Yes	Instruction misalignment
MSF Single-byte XOR Countdown	Yes	Changing key
MSF Variable-length `fnstenv/mov` XOR	Yes	FPU handling
MSF `jmp/call` XOR Additive Feedback Encoder	Yes	Additive feedback key Canary to end loop
MSF BloXor	Yes	Metamorphic block based XOR
MSF Shikata-Ga-Nai	Yes	Same block polymorphic Additive feedback key

Table 4.2: *Encoding Techniques Tested.*

GEORGE MASON UNIVERSITY

Shikata-Ga-Nai, Techniques

```
Offset Bytecode     Mnemonic            ; Comment
0000   DAD4         fcmovbe st4         ; fpu stores PC
0002   B892BA1E5C   mov eax,0x5c1eba92  ; the key
0007   D97424F4     fnstenv [esp-0xc]   ; push 0x0s addr
000B   5B           pop ebx             ; ebx = 0x0s addr
000C   29C9         sub ecx,ecx
000E   B10B         mov cl,0xb          ; words to decode
0010   83C304       add ebx,0x4         ; inc target
0013   314314       xor [ebx+0x14],eax  ; update [0x18]
0016   034386       add eax, [ebx-0x7a] ; 0x18 is encoded
0019   58           pop eax             ; part of decoder
001A   EBB7         jmp 0xd3            ; part of decoder
001C   B5C5         mov ch,0xc5
```

```
Offset Bytecode     Mnemonic          ; Comment
0000   DAD4         fcmovbe st4
0002   B892BA1E5C   mov eax,0x5c1eba92
0007   D97424F4     fnstenv [esp-0xc]
000B   5B           pop ebx
000C   29C9         sub ecx,ecx
000E   B10B         mov cl,0xb
0010   83C304       add ebx,0x4         ; inc target
0013   314314       xor [ebx+0x14],eax ; decode target
0016   034314       add eax,[ebx+0x14] ; modify key
0019   E2F5         loop 0x10          ; jmp 0x10, ecx--
001B   <deobfuscated 1st byte of shellcode>
001C   <obfuscated shellcode>
```

Table 4.3: *Anti-emulation Techniques Tested.*

Technique	Evaded?
FPU instruction `fpstenv`	Yes
Same block modification	Yes
Repeated string instruction `rep stosb`	Yes
Obscure instructions `sal`	Yes
Alternate encodings `test`	Yes
Undocumented opcodes `salc`	Yes

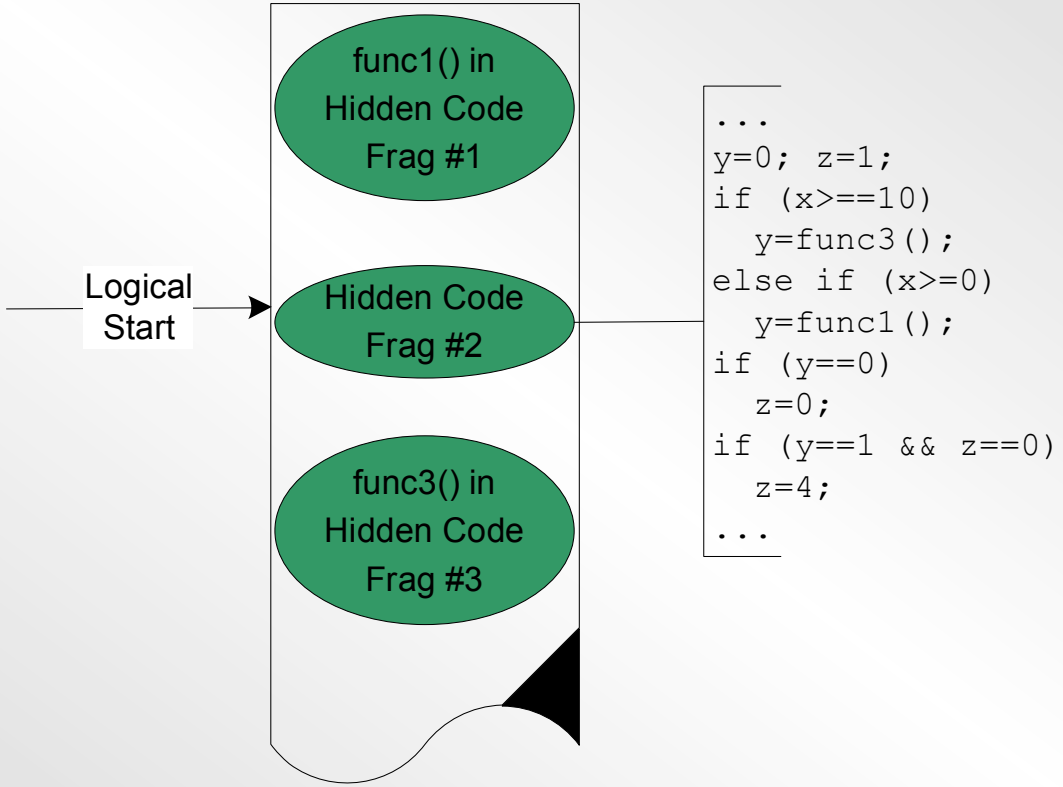
GEORGE MASON UNIVERSITY

Incremental Encoder

```
>> Printing the Data_trace memory map (8 snapshots)
>>    Printing snapshot 0
          0 1 2 3  4 5 6 7  8 9 a b  c d e f
0xbfd7cf50                  7200873d ca3c872f
0xbfd7cf60 ab57d0be a98db797 f96e5730 7b6e4a6d
0xbfd7cf70 6ba626bc baa6f76d baa6266d ba77266d
0xbfd7cf80 6b772614 76184902

>>    Printing snapshot 1
          0 1 2 3  4 5 6 7  8 9 a b  c d e f
0xbfd7cf50                  89e731c0 31db31d2
0xbfd7cf60 50b06643 526a016a 0289e1cd 8089fc90
0xbfd7cf70 90419041 41414190 41419090 41909090
0xbfd7cf80 909090e9 8dffffff

>>    Printing snapshot 2
          0 1 2 3  4 5 6 7  8 9 a b  c d e f
0xbfd7cf50                  0d28d966 37cc80c2
0xbfd7cf60 84cfe9db ece8f8db 3acde8db d2460ad7
0xbfd7cf70 949db80d e2460970 04976141 148ea9fc
0xbfd7cf80 145f7854 09301742

>>    Printing snapshot 3
          0 1 2 3  4 5 6 7  8 9 a b  c d e f
0xbfd7cf50                  89e731db b303687f
0xbfd7cf60 00000166 68271066 be020066 5689e26a
0xbfd7cf70 105250b0 6689e1cd 805889fc 90414141
0xbfd7cf80 909090e9 8dffffff

>>    Printing snapshot 4
          0 1 2 3  4 5 6 7  8 9 a b  c d e f
0xbfd7cf50                  3c7e935a 3c77aaa6
0xbfd7cf60 bcb7d719 bd88ab98 c0378ad8 4cf65b09
0xbfd7cf70 9d275bd8 4cf68ad8 4c275bd8 4c278ad8
0xbfd7cf80 9d278a70 8048e566

>>    Printing snapshot 5
          0 1 2 3  4 5 6 7  8 9 a b  c d e f
0xbfd7cf50                  31c989c3 31c0b03f
0xbfd7cf60 b100cd80 b03fb101 cd809041 41414190
0xbfd7cf70 90904141 41419041 41904141 41909041
0xbfd7cf80 909090e9 8dffffff

>>    Printing snapshot 6
          0 1 2 3  4 5 6 7  8 9 a b  c d e f
0xbfd7cf50                  0f49f534 11afd734
0xbfd7cf60 56afc635 50b164ec 3509470d b762f7d5
0xbfd7cf70 df4d24cc 7fc1e51d ae10e51d 7fc1e5cc
0xbfd7cf80 ae1034b5 b37f5ba3

>>    Printing snapshot 7
          0 1 2 3  4 5 6 7  8 9 a b  c d e f
0xbfd7cf50                  31c95168 2f2f7368
0xbfd7cf60 682f6269 6e31c0b0 0b89e351 89e25389
0xbfd7cf70 e1cd8090 41414141 90904141 41414190
0xbfd7cf80 909090e9 8dffffff
```

GEORGE MASON UNIVERSITY

Symbolic Conditionals

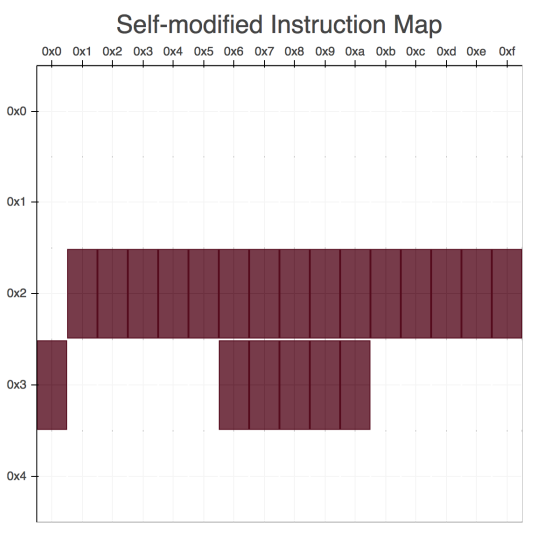


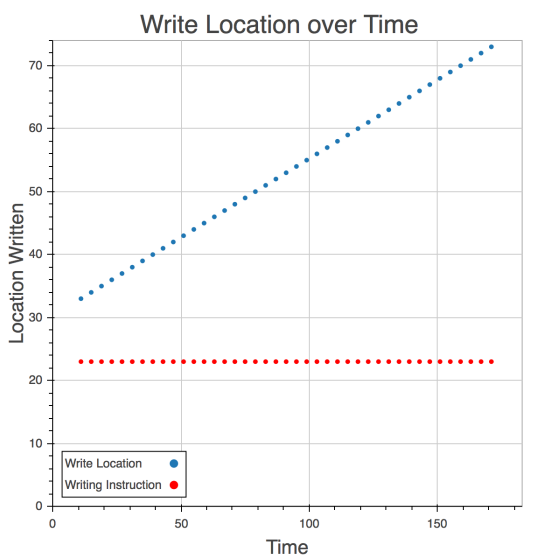
Experiments, Part 2

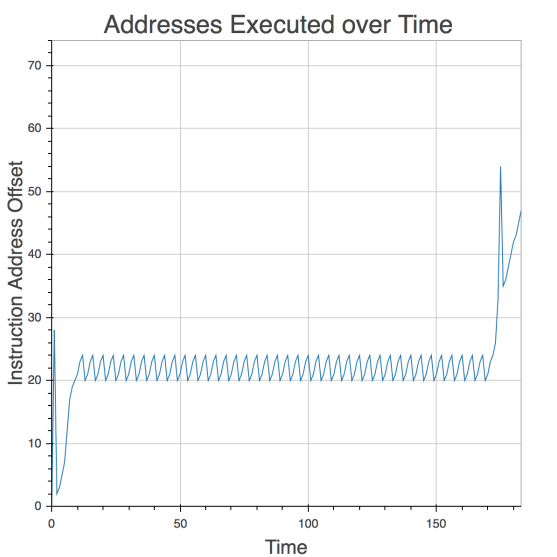
- Extrapolating malicious behavior
  - Detecting self-modifying code
- Test data-flow analysis robustness
  - Key identification
  - Network servers
    - Including SSL sockets
  - No source-code modifications
    - Full executable

GEORGE MASON UNIVERSITY

Detecting Write-then-Execute







Call +4 Dword, Key Identification

```
 >> Printing the memory map "code_Key0000" (1 snapshot)
           0 1 2 3  4 5 6 7  8 9 a b  c d e f   ASCII
0x09e13170                                 e5               .
0x09e13180 -------- -------- eb------ c0------  ................
0x09e13190 db------ b2------ b0------ 80------  ................
0x09e131a0 ff------ 6c------ 20------ 6c------  ....l... ...l...

 >> Printing the memory map "code_Key0001" (1 snapshot)
           0 1 2 3  4 5 6 7  8 9 a b  c d e f   ASCII
0x09e13180 c2------ -------- --13---- --b0----  ................
0x09e13190 --43---- --0f---- --01---- --e8----  .C..............
0x09e131a0 --ff---- --6c---- --77---- --64      .....l...w...d

 >> Printing the memory map "code_Key0002" (1 snapshot)
           0 1 2 3  4 5 6 7  8 9 a b  c d e f   ASCII
0x09e13180   5e---- -------- ----59-- ----04--   ^........Y.....
0x09e13190 ----31-- ----cd-- ----4b-- ----e8--  ..1.......K.....
0x09e131a0 ----48-- ----6f-- ----6f-- ----21    ..H...o...o...!

 >> Printing the memory map "code_Key0003" (1 snapshot)
           0 1 2 3  4 5 6 7  8 9 a b  c d e f   ASCII
0x09e13180     9b-- -------- ------31 ------31    .........1...1
0x09e13190 ------d2 ------80 ------cd ------ff  ................
0x09e131a0 ------65 ------2c ------72 ------0a  ...e...,...r....
```

Monitoring Full Executables

- Network server
  - Standard socket and SSL versions

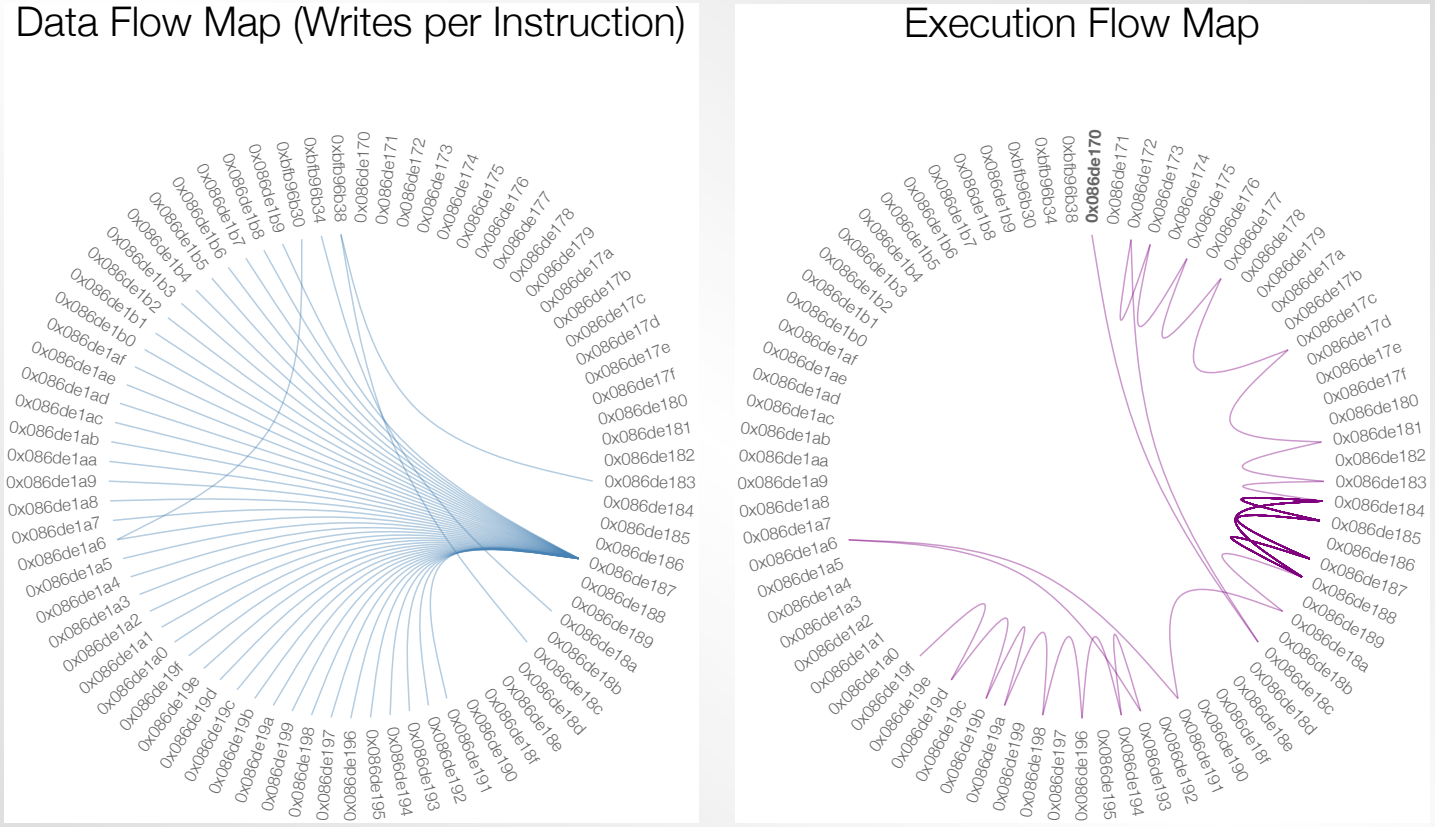
Table 5.1: *Outcome following monitored execution of standard network vs. SSL socket servers when exploited with different shellcode types.*

Server	Shellcode	Control-flow (CodeXt)	Data-flow (Taint)
Standard	Unpacked	Success	Success
	Ranged XOR	Success	Success
	Shikata-Ga-Nai	Success	Success
SSL	Unpacked	Success	S2E Failure
	Ranged XOR	Success	S2E Failure
	Shikata-Ga-Nai	Success	S2E Failure

  - With Data-flow enabled, S2E forked excessive states
    - SSL_accept function

GEORGE MASON UNIVERSITY

Data and Execution Traces

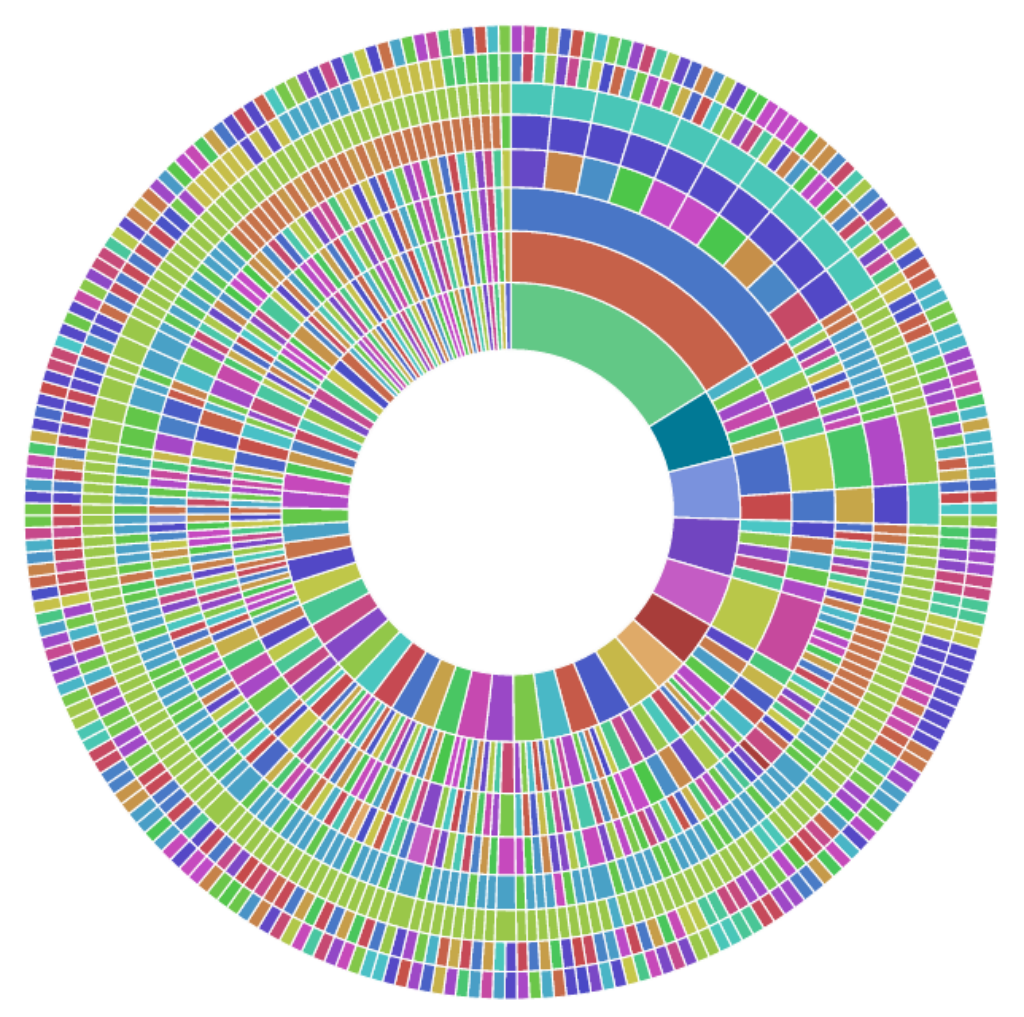


(a) Edges indicate data influence.

(b) Edges indication consecutive execution.

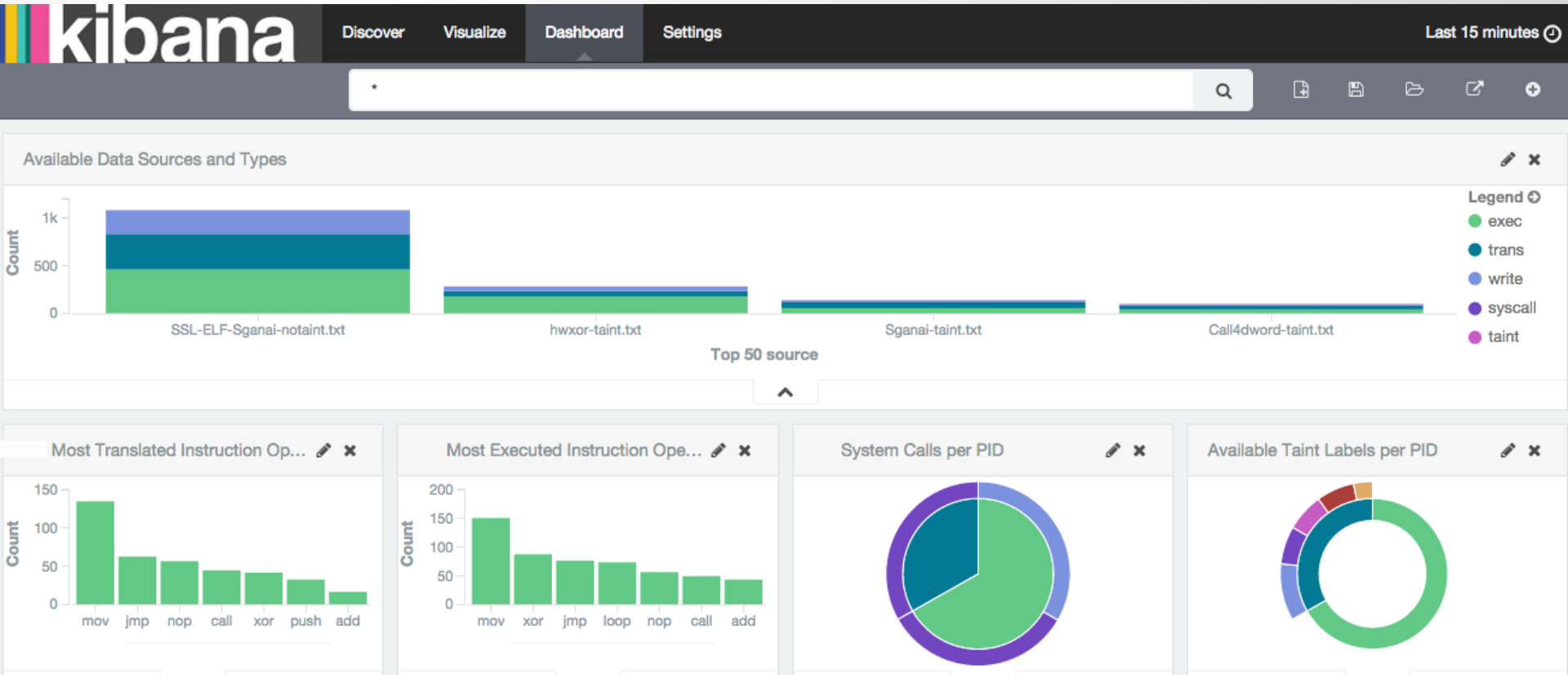
Figure 4.12: *Interactive D3 based visualization output from single byte XOR decoding.*

SSL Server Execution Trace



GEORGE MASON UNIVERSITY

Analytics


kibana
Discover
Visualize
Dashboard
Settings
Last 15 minutes
*
Available Data Sources and Types
Count
1k
500
0
SSL-ELF-Sganai-notaint.txt
hwxor-taint.txt
Sganai-taint.txt
Call4dword-taint.txt
Top 50 source
Legend
exec
trans
write
syscall
taint
Most Translated Instruction Op...
150
100
50
0
mov
jmp
nop
call
xor
push
add
Most Executed Instruction Ope...
200
150
100
50
0
mov
xor
jmp
loop
nop
call
add
System Calls per PID
Available Taint Labels per PID

Conclusion

GEORGE MASON UNIVERSITY

Conclusion

- Although emulation is heavyweight, it is:
  - Accurate and enables anti-anti-sandbox techniques
  - OS independent
- Symbolic analysis engine opens avenues
  - Taint propagation and analysis
  - Fuller branch exploration and pruning heuristics
- Our Framework
  - CodeXt accurately pinpoints and models even highly obfuscated code in adverse conditions
  - Executable extensions enable black-box analysis

GEORGE MASON UNIVERSITY

Summary

- Gaps exist in current malware forensics
  - Obfuscated shellcode
    - Solutions for advanced samples are not generic
  - Code fragments vs. portable binaries
    - No framework to handle deep analysis in both
  - Disconnect between real-time detection and low level analysis
    - Tool pipeline is not seamless
  - No system in place for aggregated analytics

GEORGE MASON UNIVERSITY

Future Development

- Coordination with Open Source projects
  - Already on Github
    - Needs cleaner repository
  - Merge contributions into QEMU/S2E
  - Create new KLEE Expr type for taint labels
  - Cuckoo sandbox
- Better analytics
  - Integrate knowledge for anomaly detection

GEORGE MASON UNIVERSITY

Thank you for your time

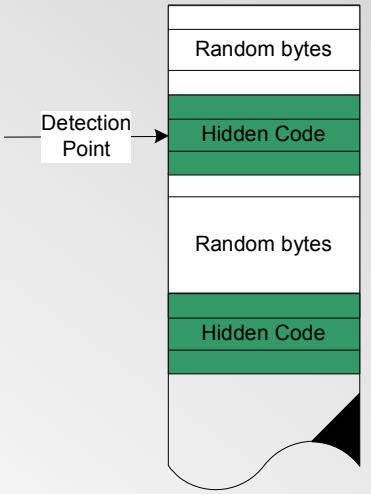
- Any questions?

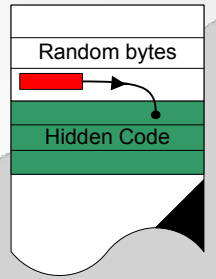
Spare Slides

GEORGE MASON UNIVERSITY

Recognizing Code

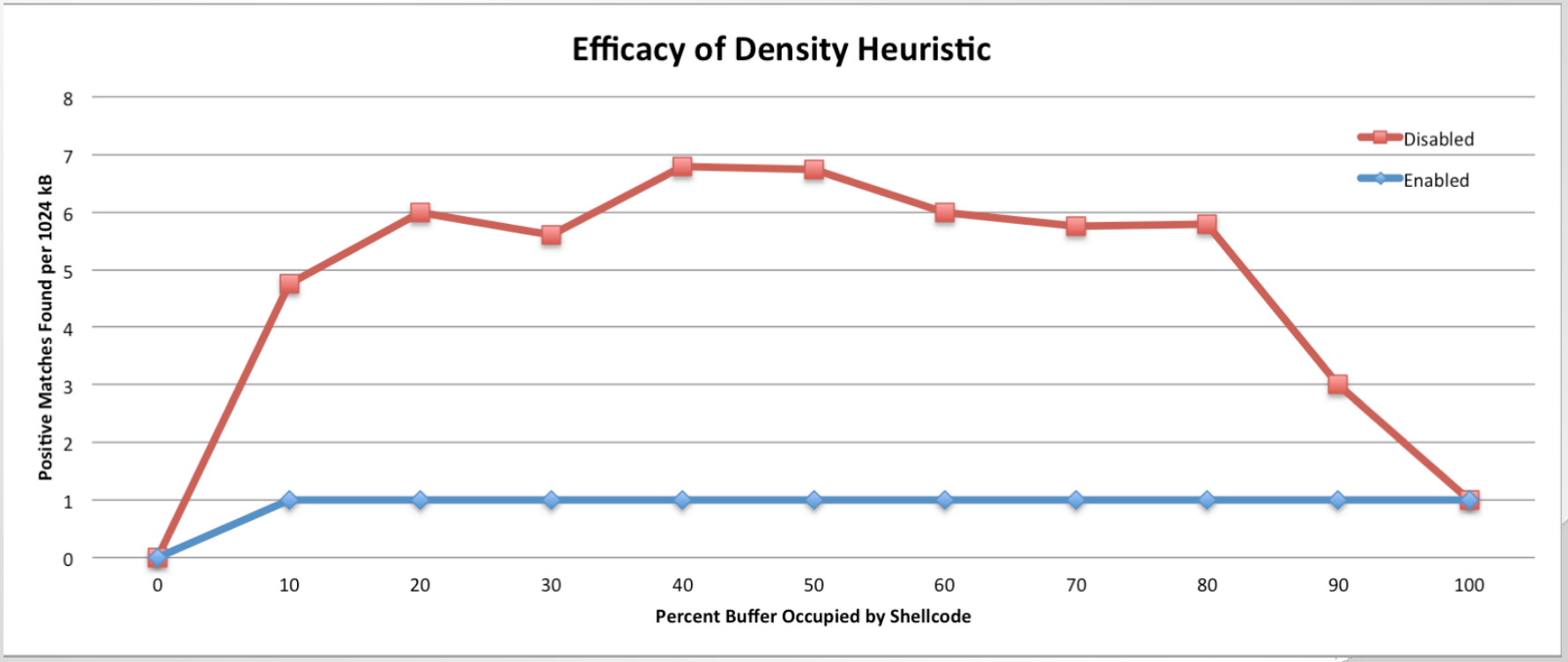
- Avoiding the Halting Problem
  - No infinite loops
  - Caps on executed instructions
    - Different types: target, non-target, system
- False cognates
  - Illegal first instructions
  - False jumps into suffix
- Many substrings
  - Matched code fragment: ends on system call, EAX within range





GEORGE MASON UNIVERSITY

Density Heuristic



GEORGE MASON UNIVERSITY

Concrete Execution Challenges

- Handling Byte Streams
  - Wrapper
  - QEMU translator returns more instruction information
- Dealing with Code Fragments
  - Chunk creation
  - Density and Enclosure
- Defeating Obfuscation
  - FPU handler
  - Intra-basic block monitoring
  - Obscure instructions

GEORGE MASON UNIVERSITY

Symbolic Execution Challenges

- Multipath Execution
- Mark Arbitrary Bytes as Symbolic
  - Detect system call semantics
- Pruning Label Propagation
  - Periodic garbage cleanup
- Executing Symbolic Code
  - Intercept onSilentConcretize

GEORGE MASON UNIVERSITY

S2E

- Selective symbolic execution engine
  - KLEE for symbolic
  - QEMU for concrete
  - Decision made on the basic block level
    - Sub-basic block interaction allowed
    - Emits pre- and post-instruction signals
  - Extended QEMU to detect system calls

GEORGE MASON UNIVERSITY

S2E Plugins

- Hooks establish events
  - S2E direct or pass through to KLEE
- Compiled into S2E
  - A fork of QEMU
- Major hooks we use:
  - onExecuteInstruction
  - onDataMemoryAccess
  - onPrivilegeChange

GEORGE MASON UNIVERSITY

KLEE Expressions

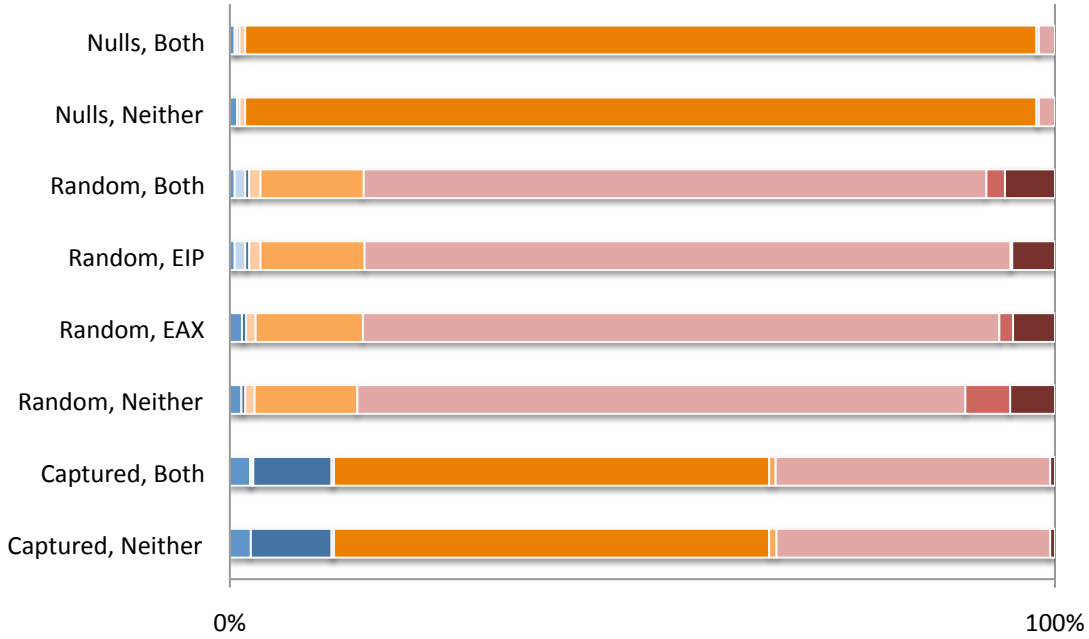
```
(Extract w8 0 (Xor w32 (w32 3085654150)
   (Concat w32 (Add w8 (w8 92) (Read w8 0 v5_prop_code_Key0003_5))
               (Concat w24 (Add w8 (w8 30) (Read w8 0 v6_prop_code_Key0002_6))
                           (Concat w16 (Add w8 (w8 186) (Read w8 0 v7_prop_code_Key0001_7))
                                       (Add w8 (w8 146) (Read w8 0 v8_prop_code_Key0000_8))))))
```

Extract 8b at offset 0 of 308565150^146

```
(Add w8 (w8 (N0) (Read w8 0 v8_prop_code_Key0000_8))
```

GEORGE MASON UNIVERSITY

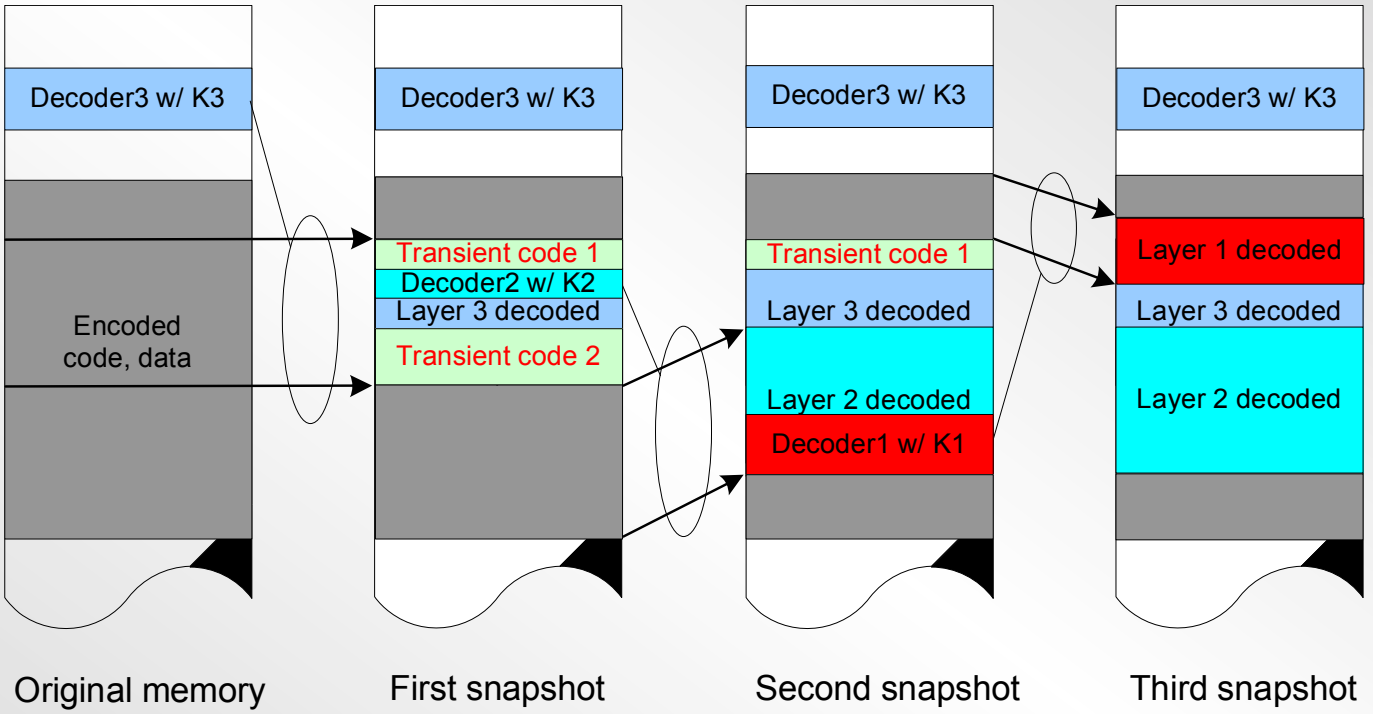
Negative Match Reasons



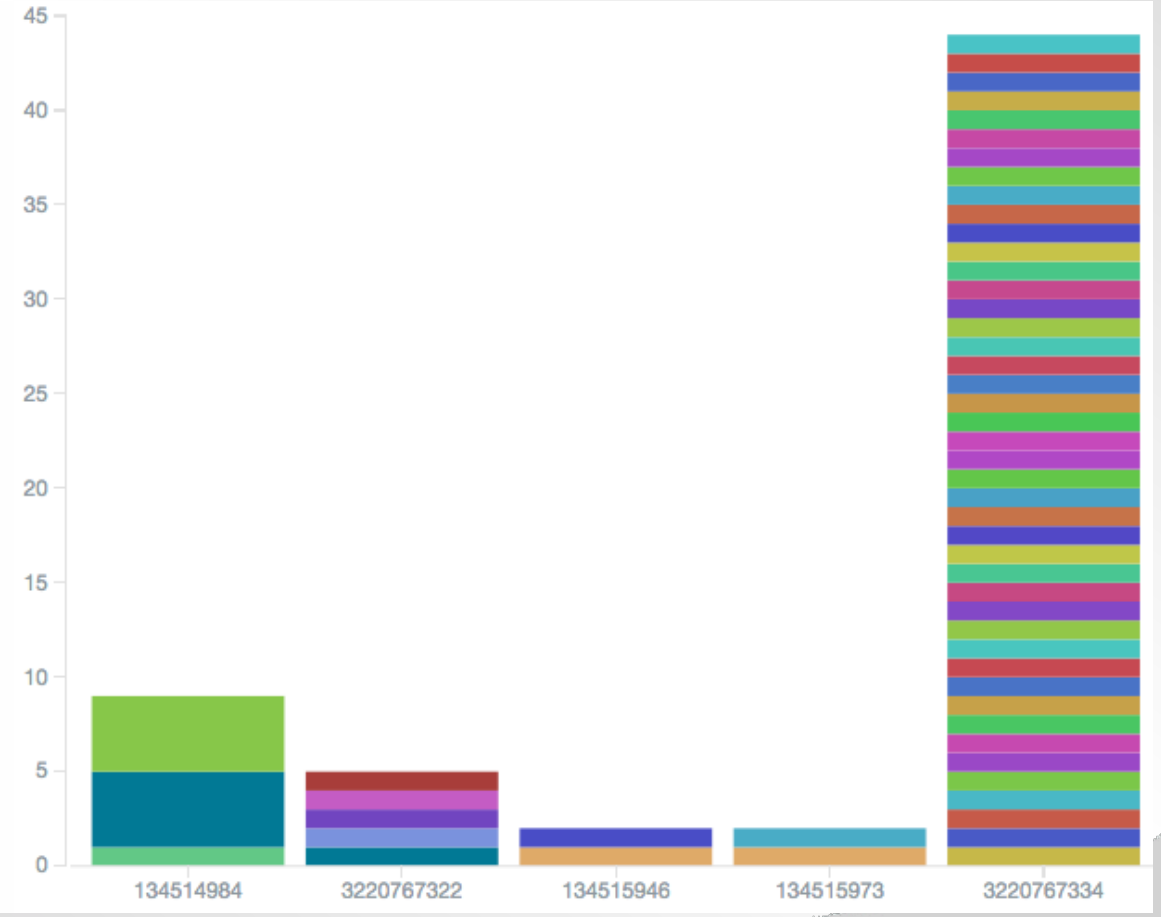
	Captured, Neither	Captured, Both	Random, Neither	Random, EAX	Random, EIP	Random, Both	Nulls, Neither	Nulls, Both
FP Irregular EAX	26	25	15	15	6	6	9	6
FP Wrong EAX	0	1	0	0	0	0	0	0
FP Wrong EIP	0	3	0	0	13	13	0	3
FP Subset	100	97	5	5	5	5	3	3
Fatal Signal OS	3	3	12	12	14	14	7	7
Invalid First Insn	539	539	0	0	0	0	981	981
Invalid OOB Jump	9	8	133	133	129	128	3	3
Unexpected OOB Jump	339	340	787	789	801	772	20	20
Runaway Kernel	0	0	58	17	2	23	0	0
Runaway Other	6	6	58	52	53	62	0	0

GEORGE MASON UNIVERSITY

Incremental Encoder



Executable Trace



GEORGE MASON UNIVERSITY