







Deobfuscation in the Age of Agentic Reverse Engineering

Tim Blazytko

 @mr_phrazer
 synthesis.to
 tim@blazytko.to

Nicolò Altamura

 @nicolodev
 nicolo.dev
 altamura@nicolo.dev

About Us

- Tim Blazytko
 - independent trainer & consultant
 - PhD in binary program analysis & reverse engineering
 - focus on (de)obfuscation & automation



- Nicolò Altamura
 - security engineer at emproof
 - reverse engineering & program analysis
 - code deobfuscation by night



deobfuscation needs to **catch up**

deobfuscation needs to **catch up**



interprocedural data-flow

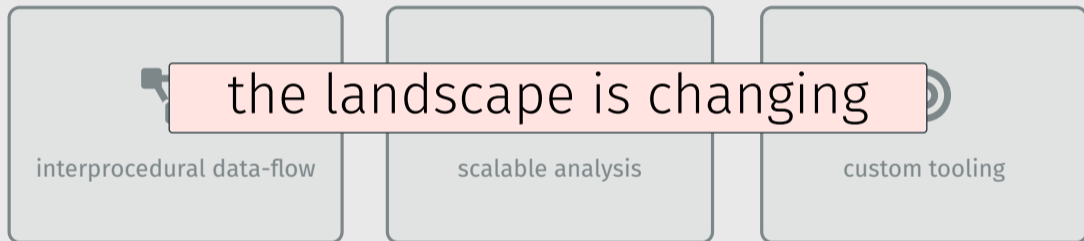


scalable analysis




custom tooling

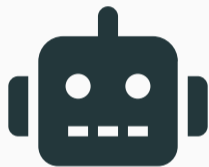
deobfuscation needs to **catch up**



 Agentic Reverse Engineering

 Agentic deobfuscation at scale

 Anti-agentic obfuscation



Agentic Reverse Engineering

A Shift in Reverse Engineering



manual

disassembler

+ analyst

A Shift in Reverse Engineering



manual

disassembler

+ analyst



assisted

LLM as copilot


A Shift in Reverse Engineering


manual
disassembler
+ analyst




assisted
LLM as copilot




agentic
LLM drives tools

A Shift in Reverse Engineering



manu
disassem
+ analyst



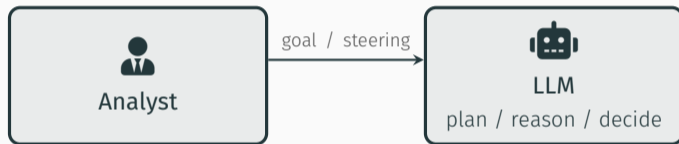
LLM as copilot



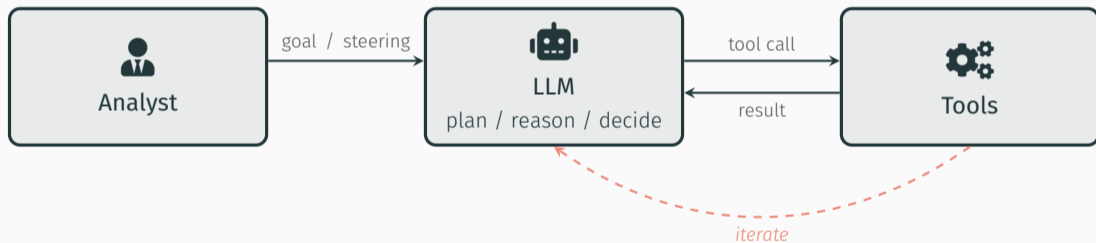
agent
LLM drives tools

agents now drive the analysis

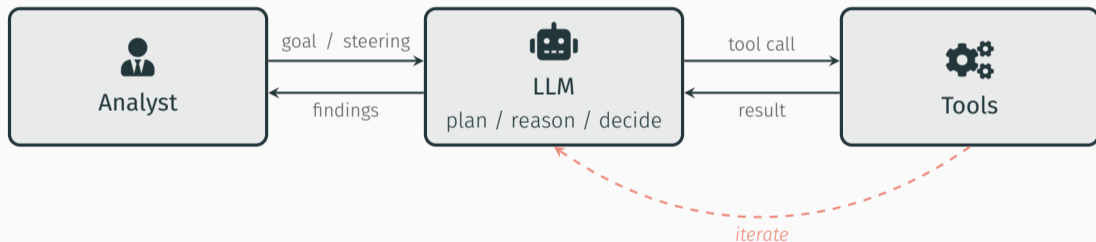
The Agent Loop



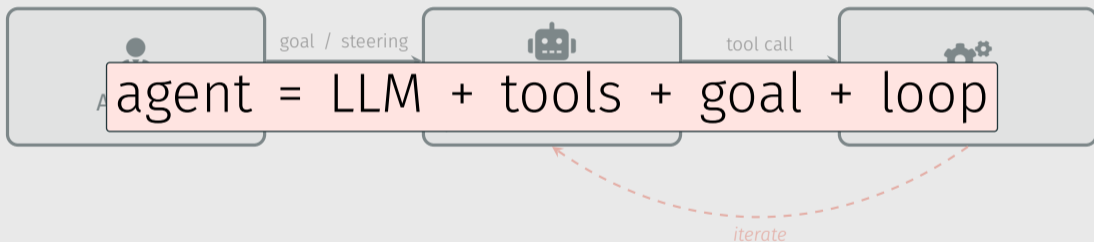
The Agent Loop



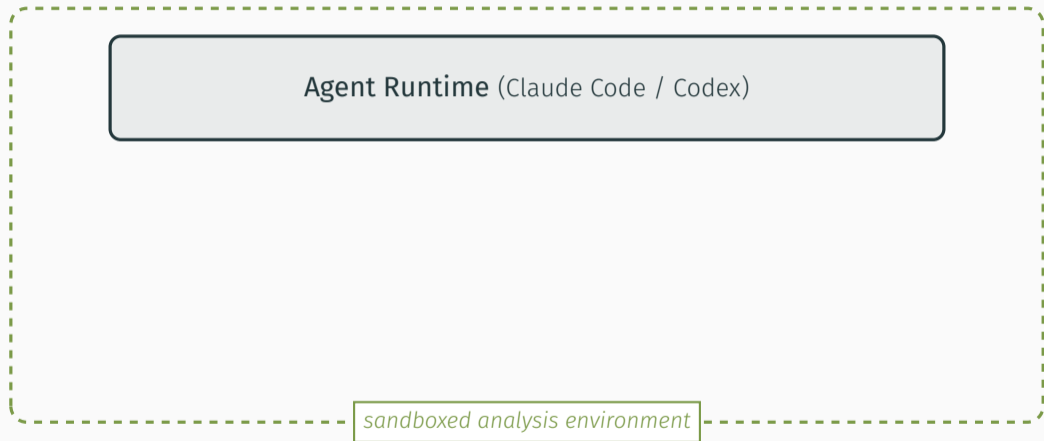
The Agent Loop



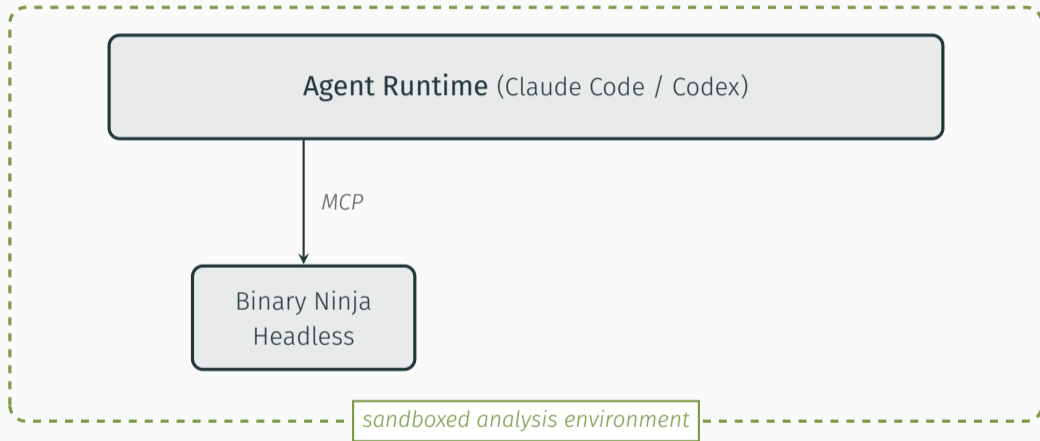
The Agent Loop



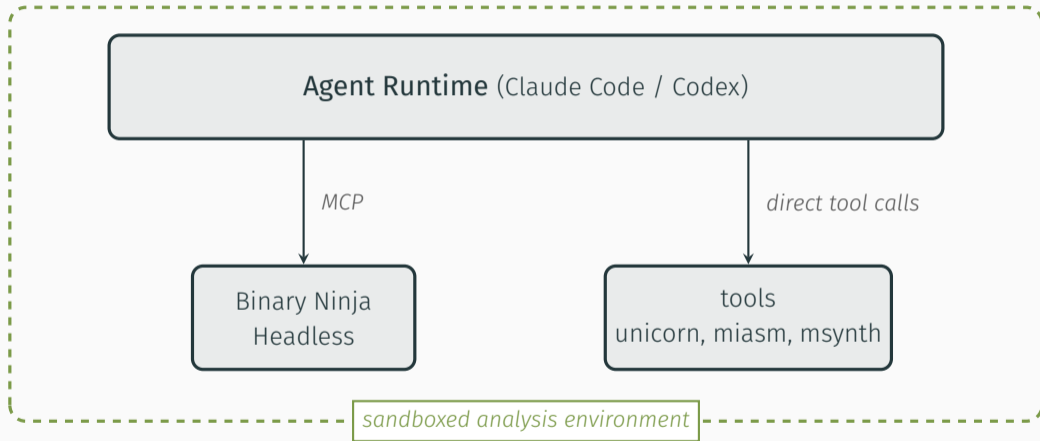
Our Analysis Environment



Our Analysis Environment



Our Analysis Environment





Agentic Deobfuscation at Scale

Experiment Setup

- experiments run in **May/June 2026**
- **frontier models**: Codex (GPT-5.5) & Claude Code (Opus 4.8)
- the agent was given **problem-specific tooling**, frameworks & examples
- workflow **guided** by a **seasoned reverse engineer**



Case Study

Virtual Machines

```
mov ecx, [esp+4]
xor eax, eax
mov ebx, 1

__secret_ip:
  mov edx, eax
  add edx, ebx
  mov eax, ebx
  mov ebx, edx
  loop __secret_ip

mov eax, ebx
ret
```

```
mov ecx, [esp+4]
xor eax, eax
mov ebx, 1
```

```
__secret_ip:
mov edx, eax
add edx, ebx
mov eax, ebx
mov ebx, edx
loop __secret_ip
```

```
mov eax, ebx
ret
```

```
mov ecx, [esp+4]
xor eax, eax
mov ebx, 1
```

```
__secret_ip:
  mov edx, eax
  add edx, ebx
  mov eax, ebx
  mov ebx, edx
  loop __secret_ip
```

```
mov eax, ebx
ret
```



```
mov ecx, [esp+4]
xor  eax, eax
mov  ebx, 1

__secret_ip:
mov  edx, eax
add  edx, ebx
mov  eax, ebx
loop ebx, edx
__secret_ip
mov  eax, ebx
ret
```



made-up instruction set

```
__bytecode:  vld  r1
             vld  r0      vpop  r2
             vpop  r1      vldi  #1
             vld  r2      vld  r3
             vld  r1      vsub  r3
             vadd  r1      vld  #0
             vld  r2      veq   r3
             vpop  r0      vbr0  #-0E
```

```
mov ecx, [esp+4]
xor  eax, eax
mov  ebx, 1
```

```
__secret_ip:
  push __bytecode
  call vm_entry
```

```
mov  eax, ebx
ret
```



made-up instruction set

```
__bytecode:
  db 54 68 69 73 20 64 6f
  db 65 73 6e 27 74 20 6c
  db 6f 6f 6b 20 6c 69 6b
  db 65 20 61 6e 79 74 68
  db 69 6e 67 20 74 6f 20
  db 6d 65 2e de ad be ef
```

```
mov ecx, [esp+4]
xor eax, eax
mov ebx, 1
```

```
__secret_ip:
push __bytecode
call vm_entry
```

```
mov eax, ebx
ret
```



made-up instruction set

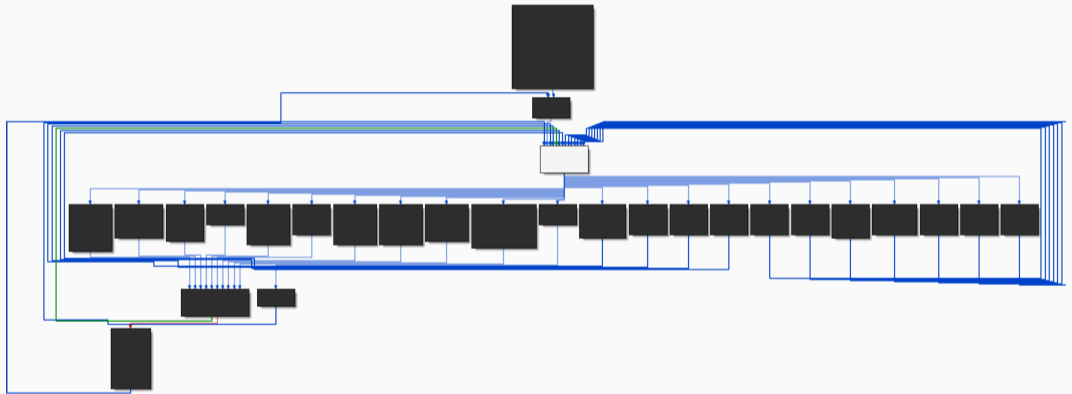
```
__bytecode:
```

```
db 54 68 69 73 20 64 6f
db 65 73 6e 27 74 20 6c
db 6f 6f 6b 20 6c 69 6b
db 65 20 61 6e 79 74 68
db 6e 67 20 74 6f 20
db 5 2e de ad be ef
```

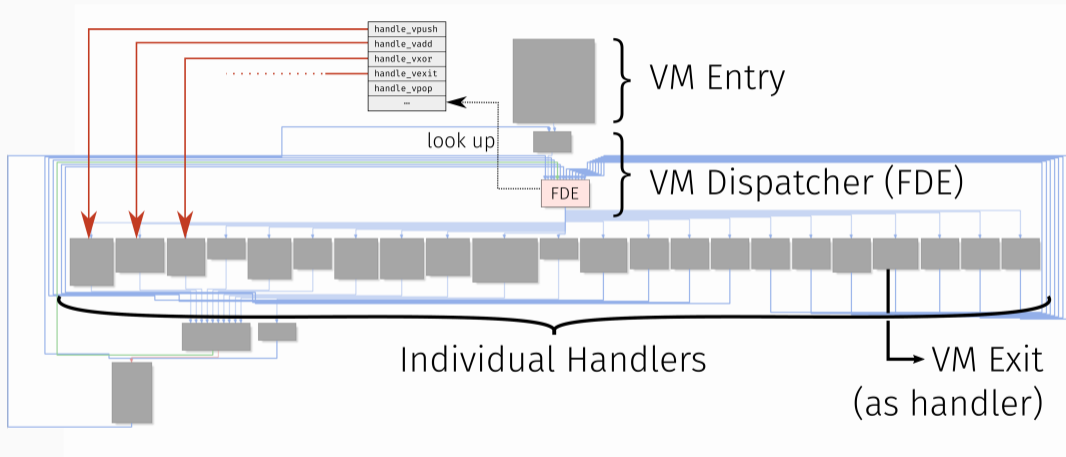


Simple VM

VM Architecture



VM Architecture



VM Deobfuscation: Reconstruct High-Level Code from Bytecode

understand the VM structure

VM Deobfuscation: Reconstruct High-Level Code from Bytecode

understand the VM structure → write a VM disassembler

VM Deobfuscation: Reconstruct High-Level Code from Bytecode

understand the VM structure → **write a VM disassembler** → reconstruct high-level code

VM Deobfuscation: Reconstruct High-Level Code from Bytecode

understand the VM structure → **write a VM disassembler** → reconstruct high-level code



VM Deobfuscation: Reconstruct High-Level Code from Bytecode

understand the VM structure → write a VM disassembler → reconstruct high-level code



0a 01 02 0b 01 05

VM Deobfuscation: Reconstruct High-Level Code from Bytecode

understand the VM structure → write a VM disassembler → reconstruct high-level code

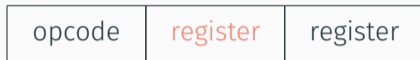


0a 01 02 0b 01 05

add

VM Deobfuscation: Reconstruct High-Level Code from Bytecode

understand the VM structure → write a VM disassembler → reconstruct high-level code



0a 01 02 0b 01 05

add r1

VM Deobfuscation: Reconstruct High-Level Code from Bytecode

understand the VM structure → write a VM disassembler → reconstruct high-level code



0a 01 02 0b 01 05

add r1, r2

VM Deobfuscation: Reconstruct High-Level Code from Bytecode

understand the VM structure → write a VM disassembler → reconstruct high-level code



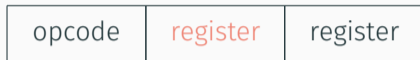
0a 01 02 0b 01 05

add r1, r2

mul

VM Deobfuscation: Reconstruct High-Level Code from Bytecode

understand the VM structure → write a VM disassembler → reconstruct high-level code



0a 01 02 0b 01 05

```
add r1, r2
```

```
mul r1
```

VM Deobfuscation: Reconstruct High-Level Code from Bytecode

understand the VM structure → write a VM disassembler → reconstruct high-level code



0a 01 02 0b 01 05

```
add r1, r2
```

```
mul r1, r5
```

VM Deobfuscation: Reconstruct High-Level Code from Bytecode

understand the VM structure → write a VM disassembler → reconstruct high-level code



0a 01 02 0b 01 05

add r1, r2

mul r1, r5

→ high-level: $(r1 + r2) * r5$



Given the obfuscated function f , deobfuscate it and reconstruct the high-level code.

Agentic VM Deobfuscation



Given the obfuscated function f , deobfuscate it and reconstruct the high-level code.



- ▶ discovered the VM architecture
- ▶ recovered the handler semantics
- ▶ wrote a VM disassembler
- ▶ reconstructed the high-level code:

```
uint64_t f(uint64_t r1, uint64_t r2, uint64_t r5)
{ return (r1 + r2) * r5; }
```



Given the obfuscated function f , deobfuscate it and reconstruct the high-level code.



▶ discovered the VM architecture

▶ recovered

essentially a one-shot

▶ wrote a VM disassembler

▶ reconstructed the high-level code:

```
uint64_t f(uint64_t r1, uint64_t r2, uint64_t r5)
{ return (r1 + r2) * r5; }
```

Commercial VM Obfuscator



decentralized dispatch

cannot enumerate handlers



decentralized dispatch

cannot enumerate handlers



handler duplication

thousands of diversified handlers

Production-Grade Hardening



decentralized dispatch

cannot enumerate handlers



handler duplication

thousands of diversified handlers



bytecode encryption

control-flow-dependent rolling key

Production-Grade Hardening



decentralized dispatch

cannot enumerate handlers



handler duplication

thousands of diversified handlers

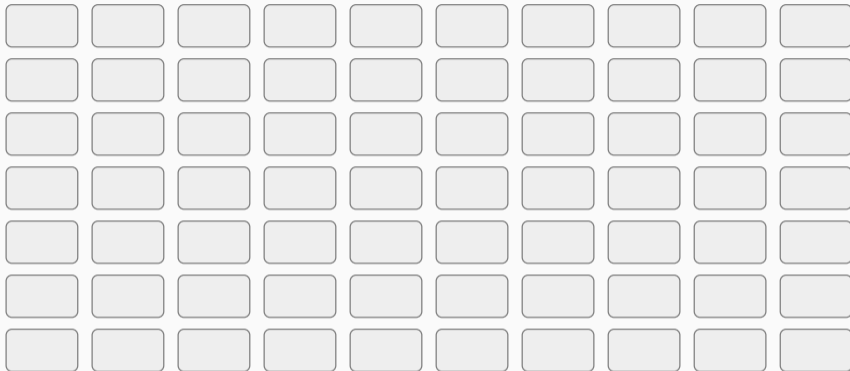


bytecode encryption

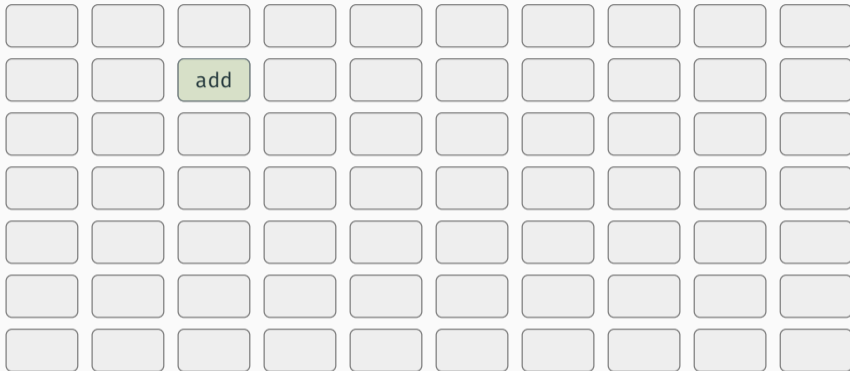
control-flow-dependent rolling key

enforces **large-scale automation** and **following the execution flow**

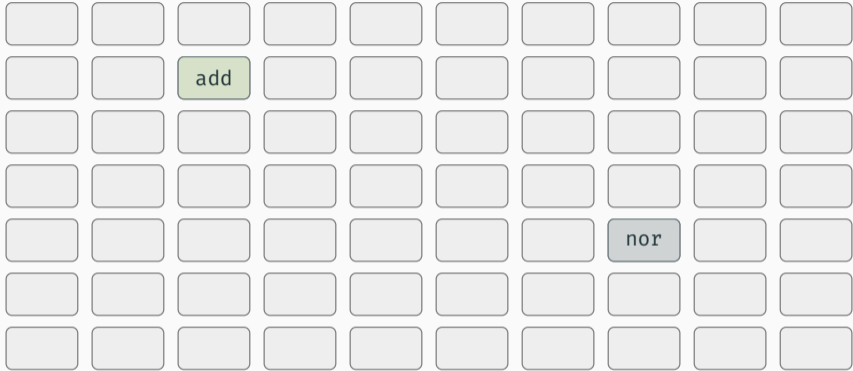
Handler Duplication



Handler Duplication



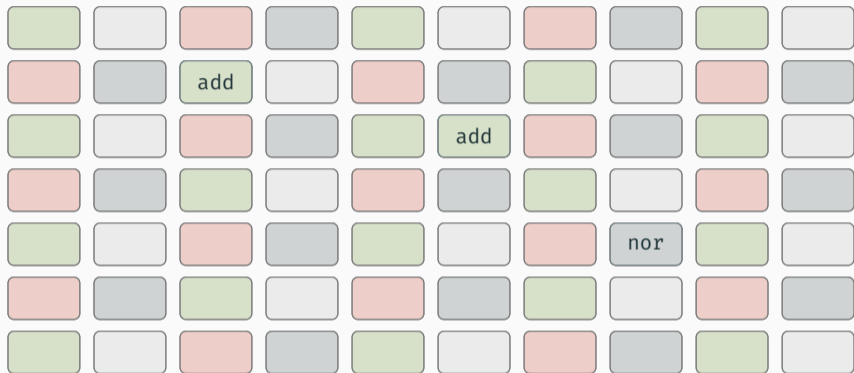
Handler Duplication



Handler Duplication

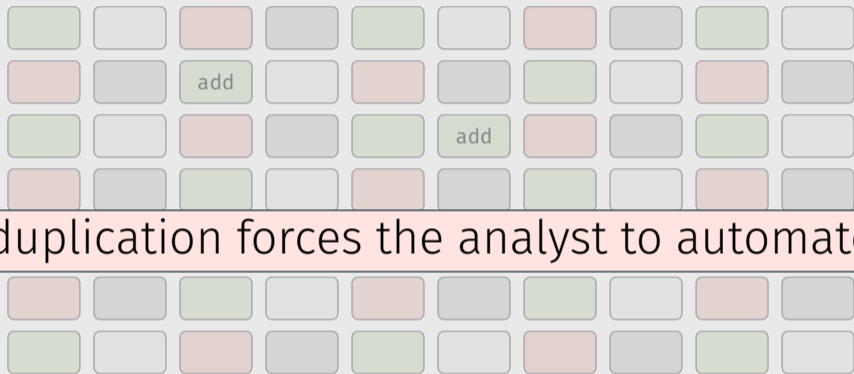


Handler Duplication



~12,000 diversified handlers · ~30 families

Handler Duplication



~12,000 diversified handlers · ~30 families

Control-Flow-Sensitive Bytecode Encryption

encrypted

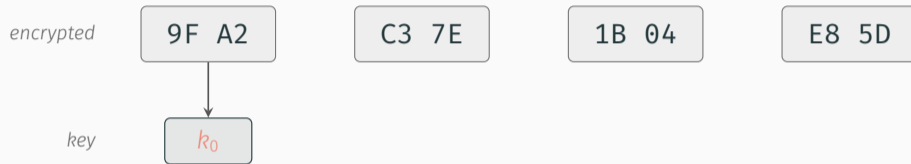
9F A2

C3 7E

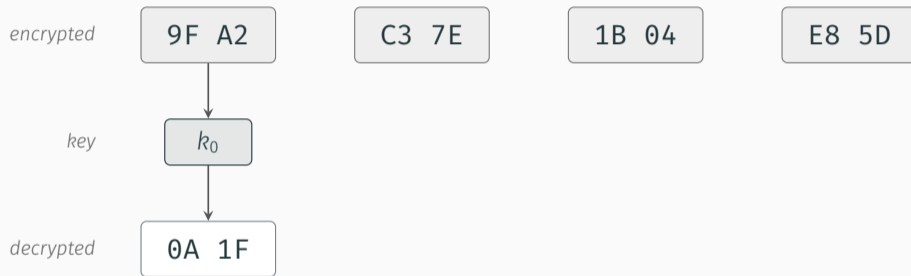
1B 04

E8 5D

Control-Flow-Sensitive Bytecode Encryption



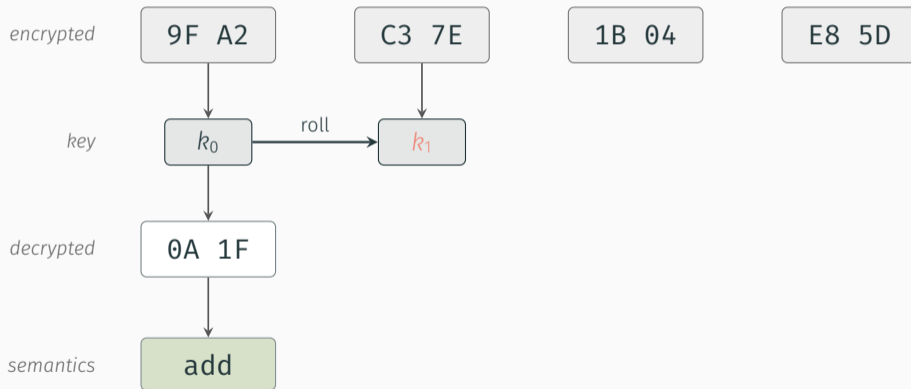
Control-Flow-Sensitive Bytecode Encryption



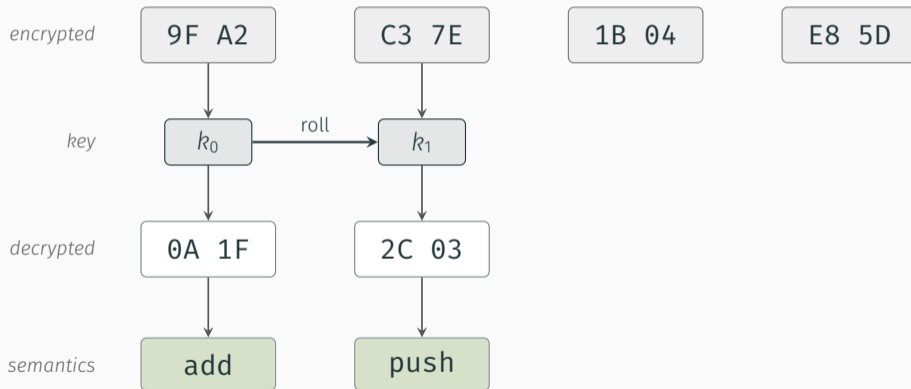
Control-Flow-Sensitive Bytecode Encryption



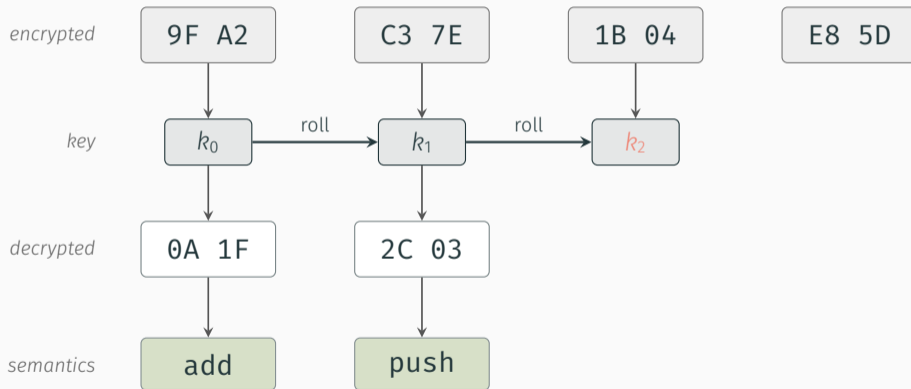
Control-Flow-Sensitive Bytecode Encryption



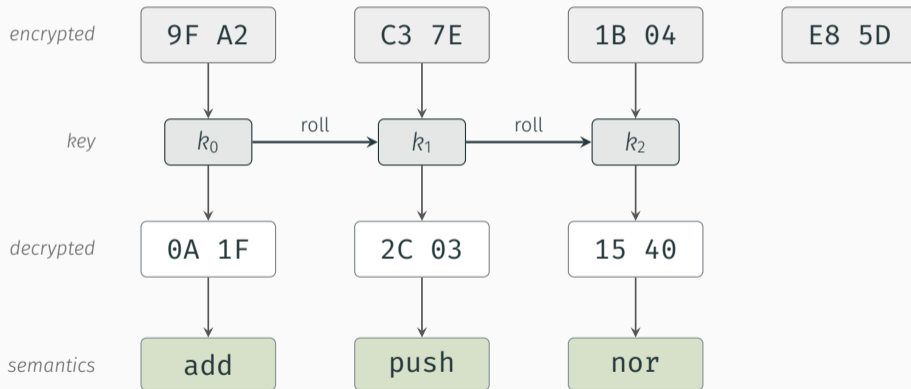
Control-Flow-Sensitive Bytecode Encryption



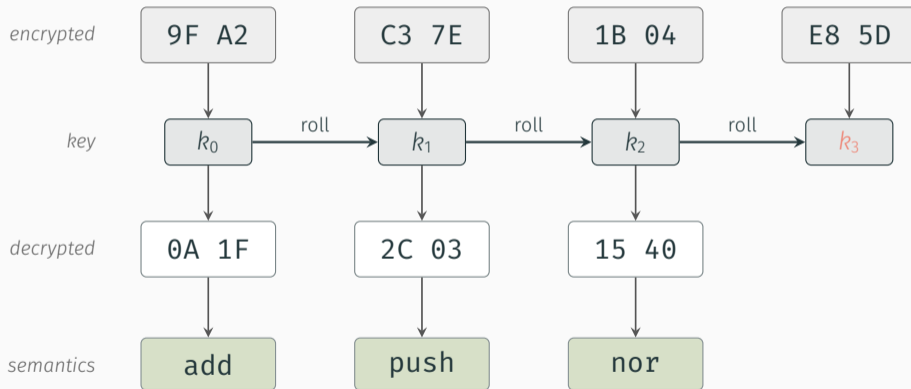
Control-Flow-Sensitive Bytecode Encryption



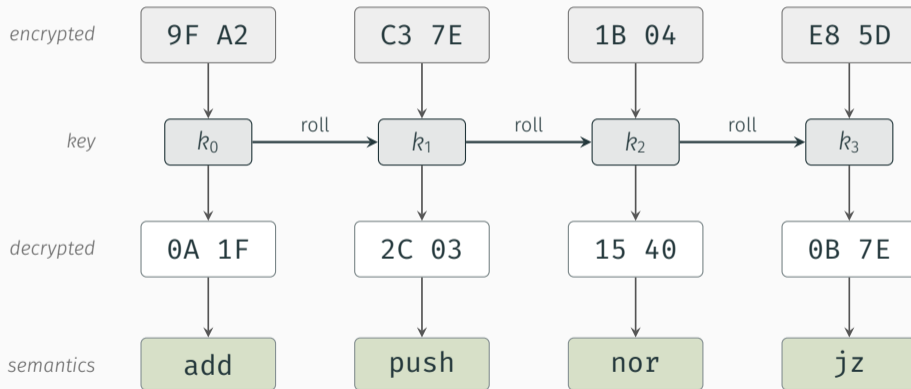
Control-Flow-Sensitive Bytecode Encryption



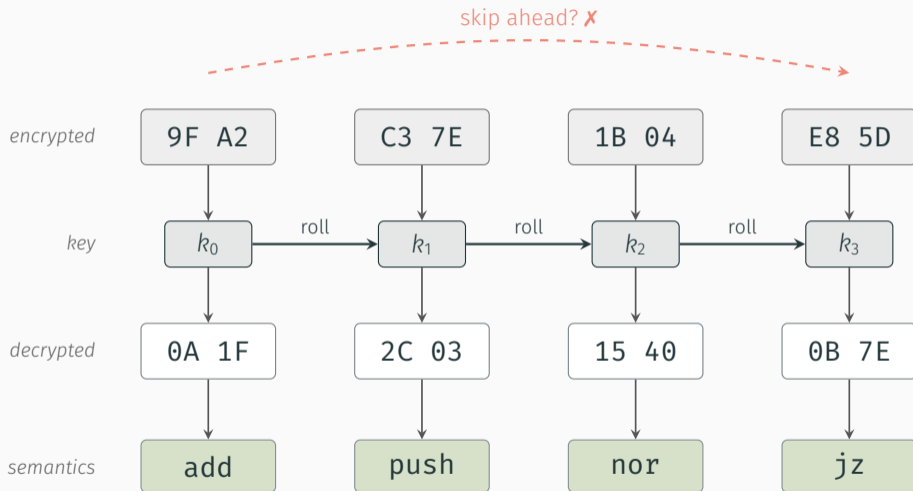
Control-Flow-Sensitive Bytecode Encryption



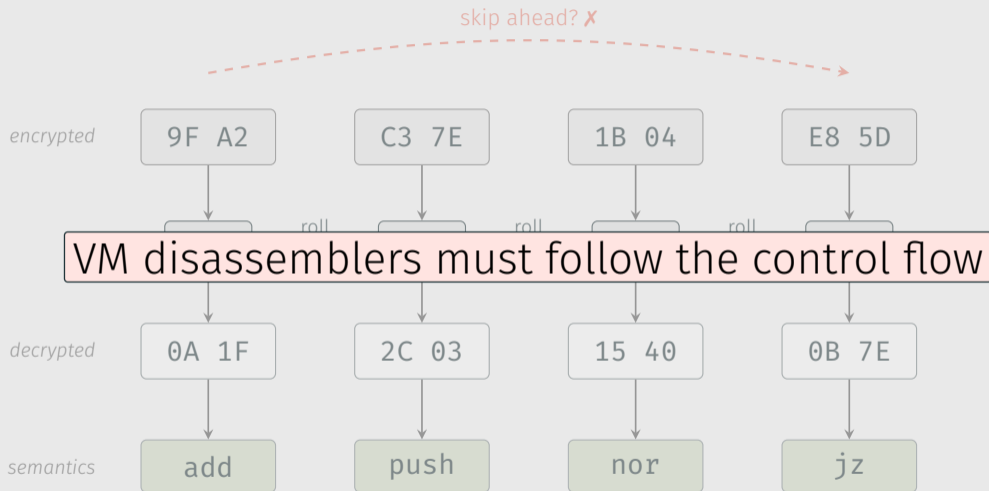
Control-Flow-Sensitive Bytecode Encryption



Control-Flow-Sensitive Bytecode Encryption



Control-Flow-Sensitive Bytecode Encryption



Agentic VM Deobfuscation Attempt



Given the obfuscated function f , deobfuscate it and reconstruct the high-level code.

Agentic VM Deobfuscation Attempt



Given the obfuscated function f , deobfuscate it and reconstruct the high-level code.



producing nothing usable after several hours of work



exploration /
path discovery

entry → vmexit

Deobfuscation of Commercial VM Obfuscators



exploration /
path discovery

entry → vmexit



handler
semantics

lift each handler

Deobfuscation of Commercial VM Obfuscators



exploration /
path discovery

entry → vmexit



handler
semantics

lift each handler



per-handler
artifacts

VM disassembly

Deobfuscation of Commercial VM Obfuscators



exploration /
path discovery

entry → vmexit



handler
semantics

lift each handler



per-handler
artifacts

VM disassembly



reconstruction

original code

Deobfuscation of Commercial VM Obfuscators



exploration /
path discovery

entry → vmexit



handler
semantics

lift each handler



per-handler
artifacts

VM disassembly



reconstruction

original code

common approaches: symbolic execution, LLVM-based lifting

Writing Disassemblers for VM-based Obfuscators

21 Oct 2021 - Tim Blazytko

knowledge

how to write a symbolic-execution-based VM disassembler

synthesis.to/2021/10/21/vm_based_obfuscation.html

Writing Disassemblers for VM-based Obfuscators

21 Oct 2021 - Tim Blazytko

knowledge

how to write a symbolic-execution-based VM disassembler

synthesis.to/2021/10/21/vm_based_obfuscation.html

scripts

templates for SE-based disassemblers (from the blog post)

Writing Disassemblers for VM-based Obfuscators

21 Oct 2021 - Tim Blazytko

knowledge

how to write a symbolic-execution-based VM disassembler

synthesis.to/2021/10/21/vm_based_obfuscation.html

scripts

templates for SE-based disassemblers (from the blog post)

unicorn → emulation

tools

miasm → symbolic execution

msynth → MBA simplification

Agentic Deobfuscation Workflow

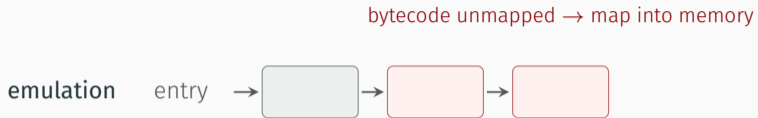
emulation entry → 

Agentic Deobfuscation Workflow

API call → hook / run it



Agentic Deobfuscation Workflow



Agentic Deobfuscation Workflow



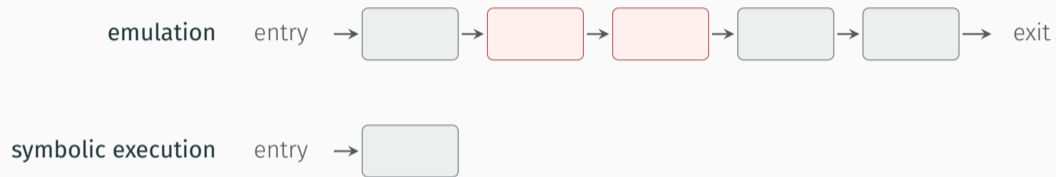
Agentic Deobfuscation Workflow



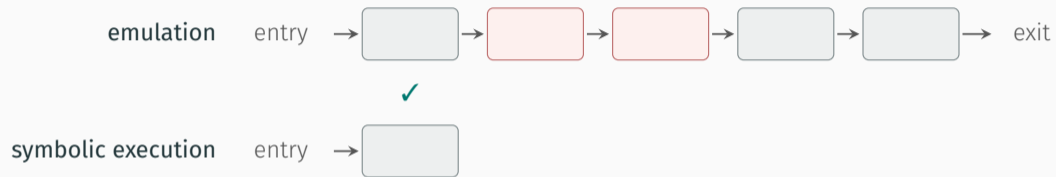


emulation as ground truth

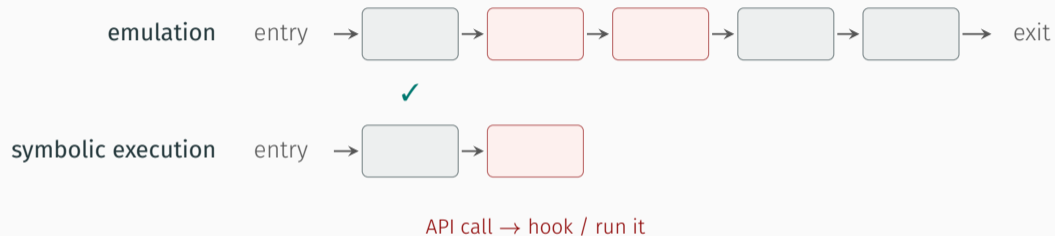
Agentic Deobfuscation Workflow



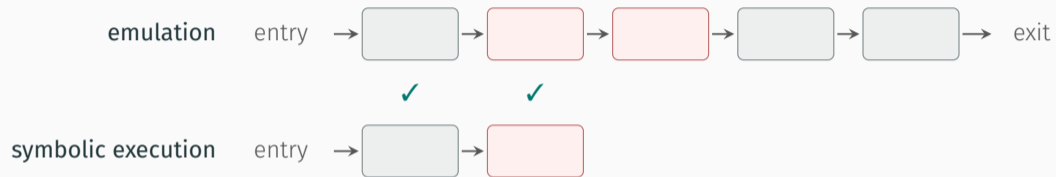
Agentic Deobfuscation Workflow



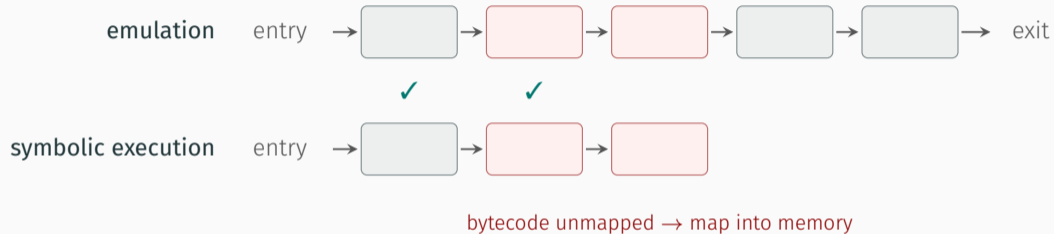
Agentic Deobfuscation Workflow



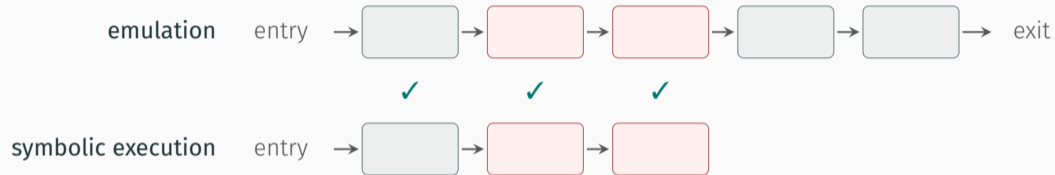
Agentic Deobfuscation Workflow



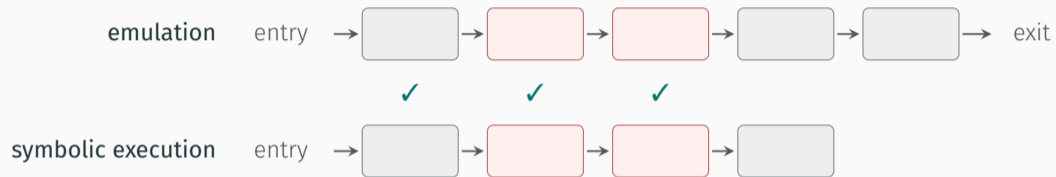
Agentic Deobfuscation Workflow



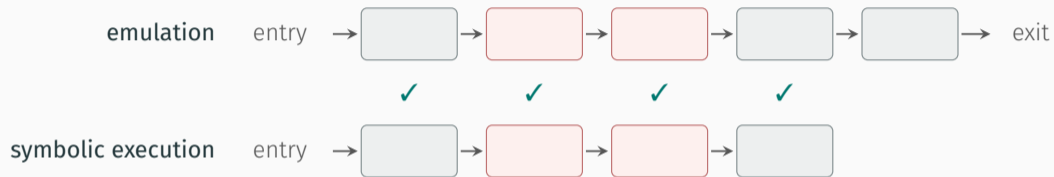
Agentic Deobfuscation Workflow



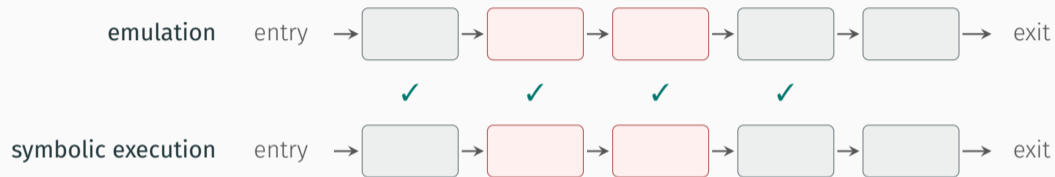
Agentic Deobfuscation Workflow



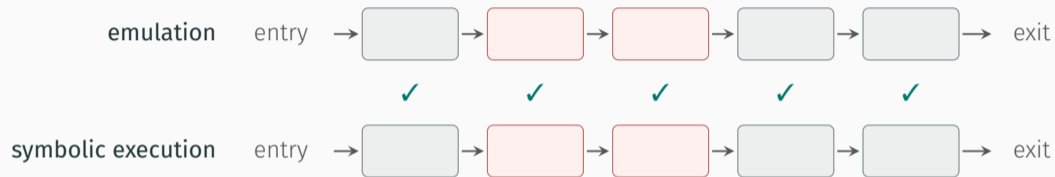
Agentic Deobfuscation Workflow



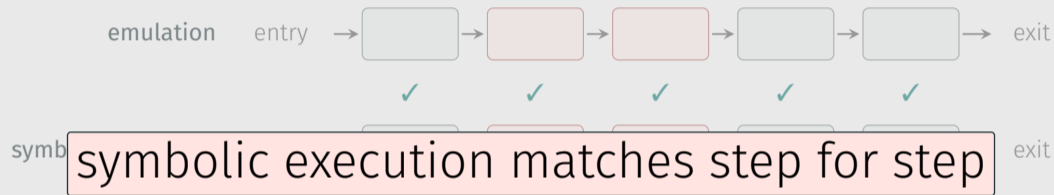
Agentic Deobfuscation Workflow



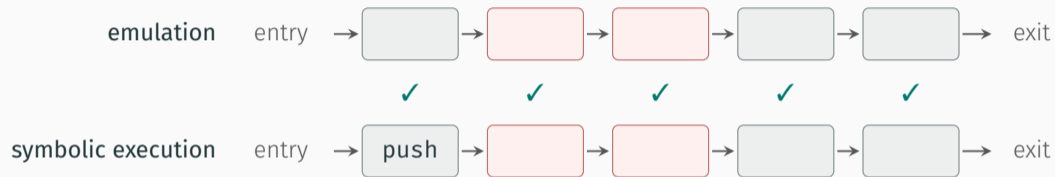
Agentic Deobfuscation Workflow



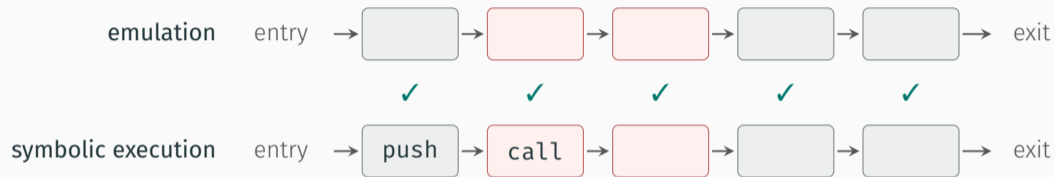
Agentic Deobfuscation Workflow



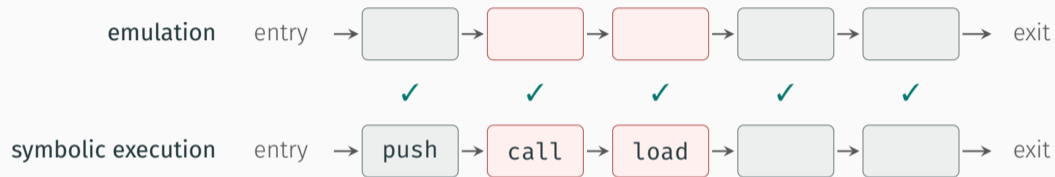
Agentic Deobfuscation Workflow



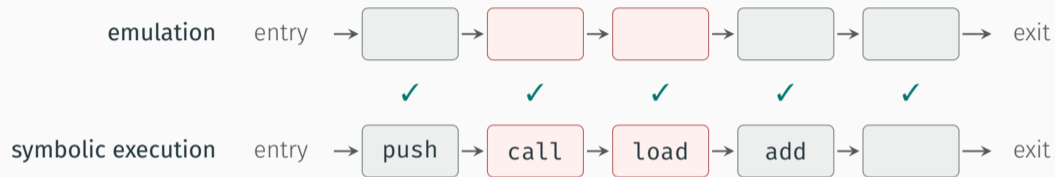
Agentic Deobfuscation Workflow



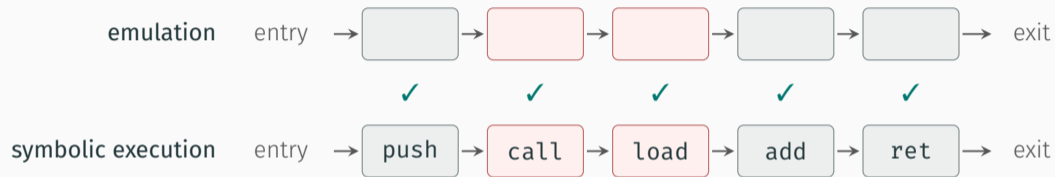
Agentic Deobfuscation Workflow



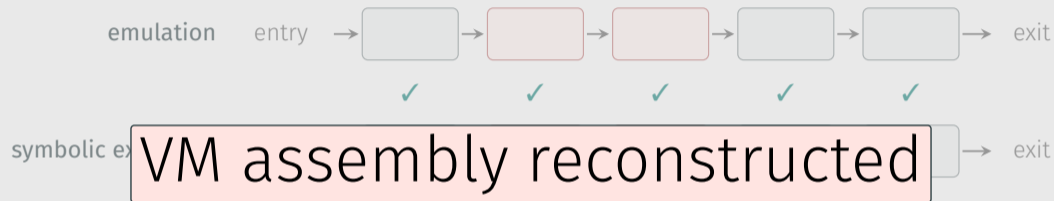
Agentic Deobfuscation Workflow



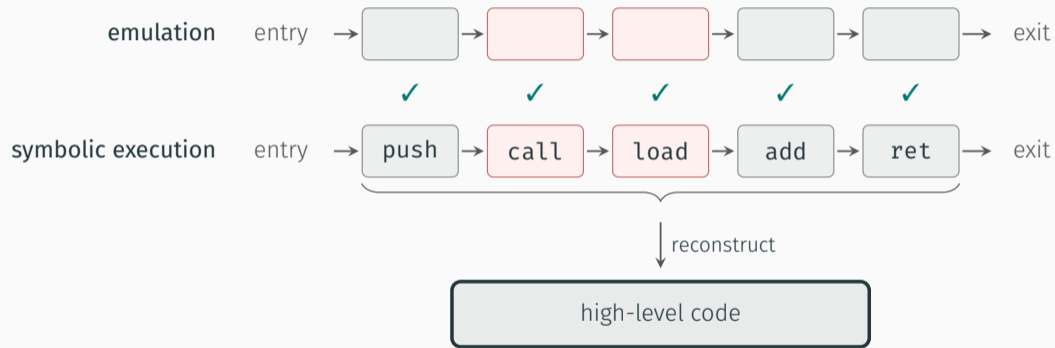
Agentic Deobfuscation Workflow



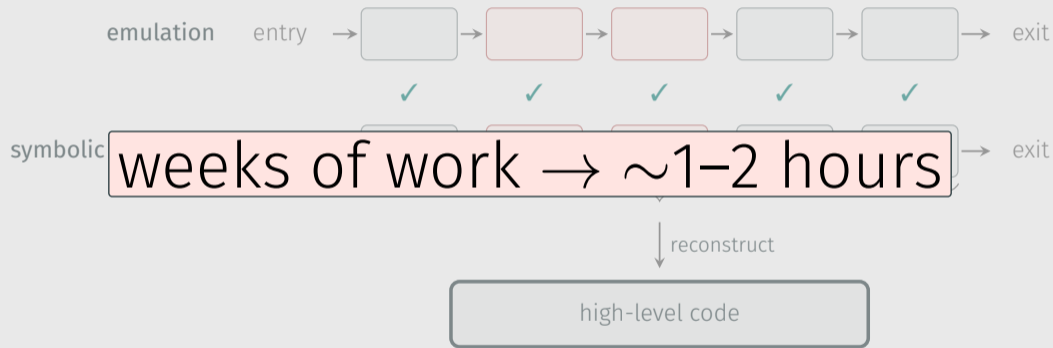
Agentic Deobfuscation Workflow



Agentic Deobfuscation Workflow



Agentic Deobfuscation Workflow

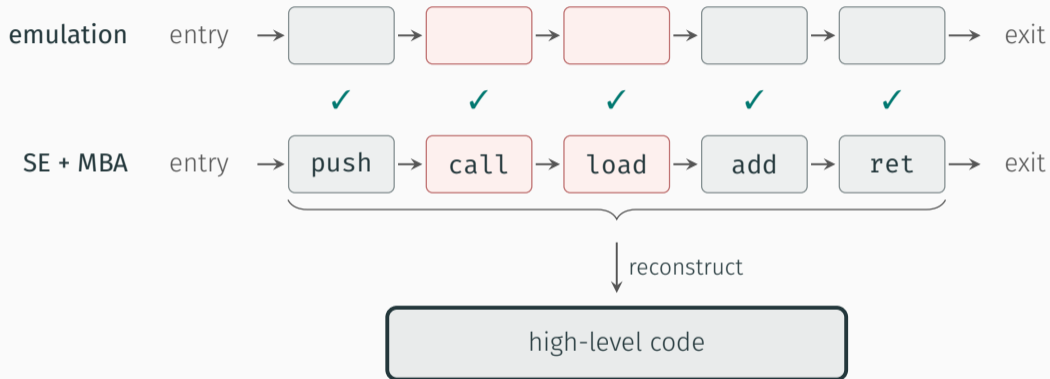


Anti Cheat

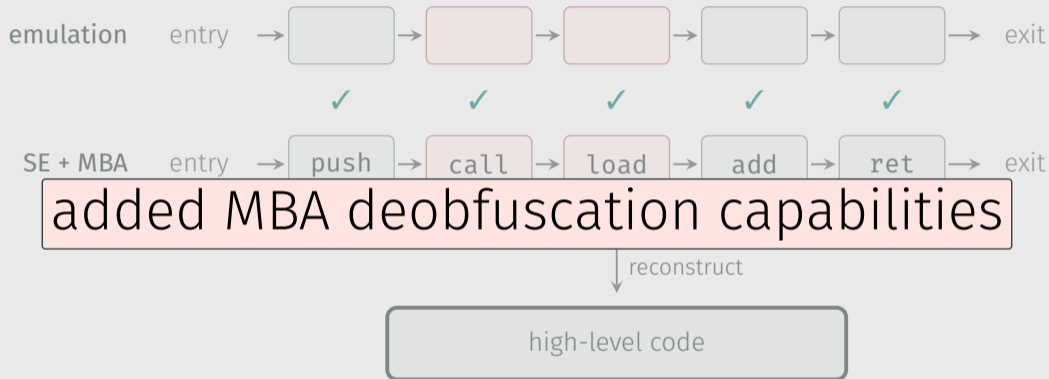
Kernel-Level Anti-Cheat

- custom in-house VM
- direct-threaded dispatch
- rolling-key bytecode encryption
- MBA-obfuscated handlers

Agentic Deobfuscation Workflow



Agentic Deobfuscation Workflow



Deobfuscation of the Anti-Cheat VM

shared VM interpreter

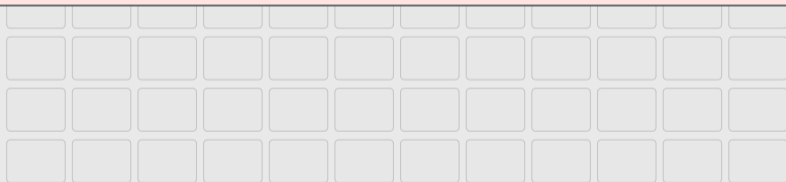
Deobfuscation of the Anti-Cheat VM

shared VM interpreter

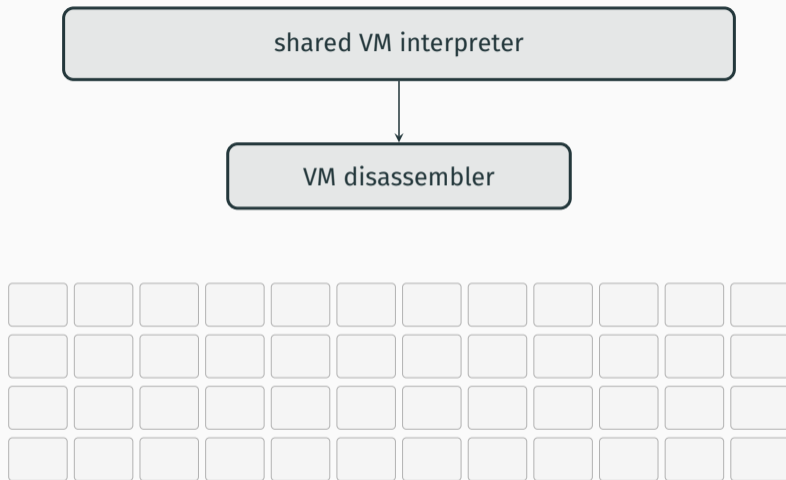


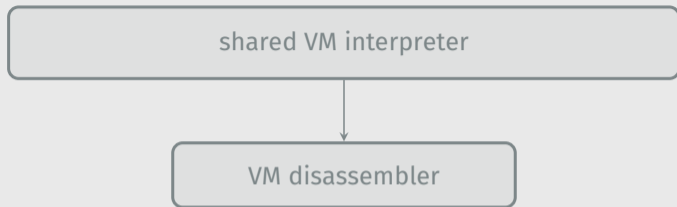
shared VM interpreter

shared by 1,023 VM-entry stubs



Deobfuscation of the Anti-Cheat VM

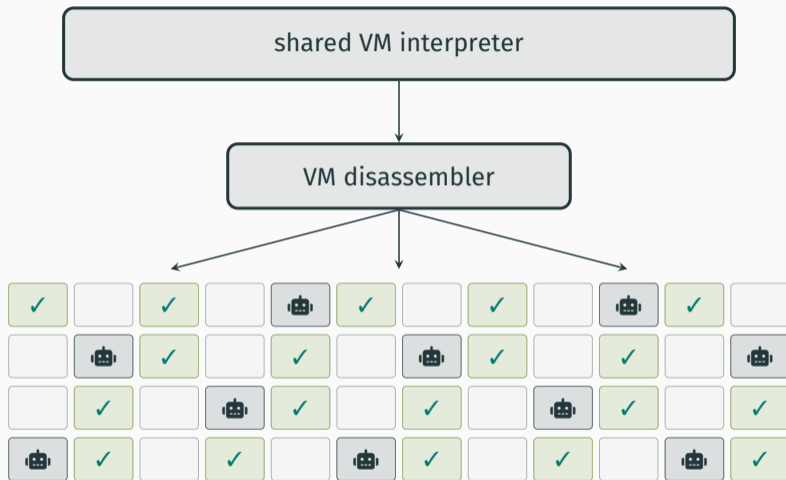




built *once* for the shared VM



Deobfuscation of the Anti-Cheat VM



Deobfuscation of the Anti-Cheat VM

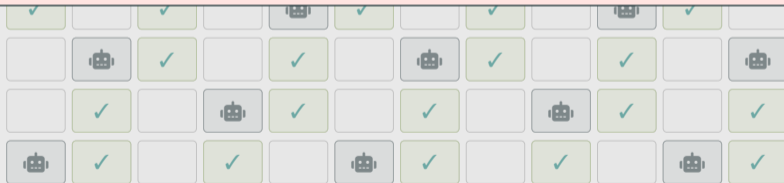
shared VM interpreter



VM disassembler



devirtualize VM-entries in parallel

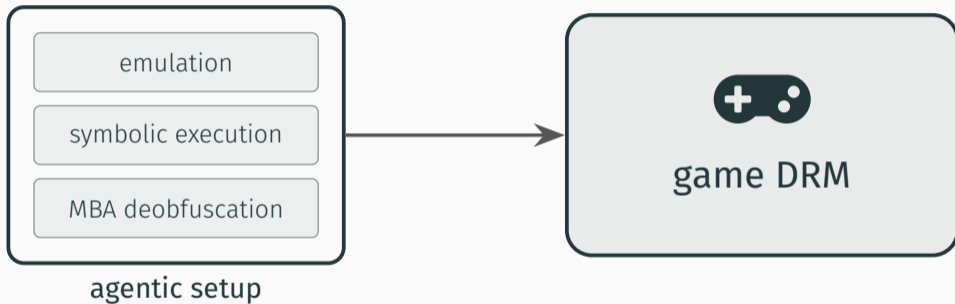


Analysis Results: Most of the Driver Recovered

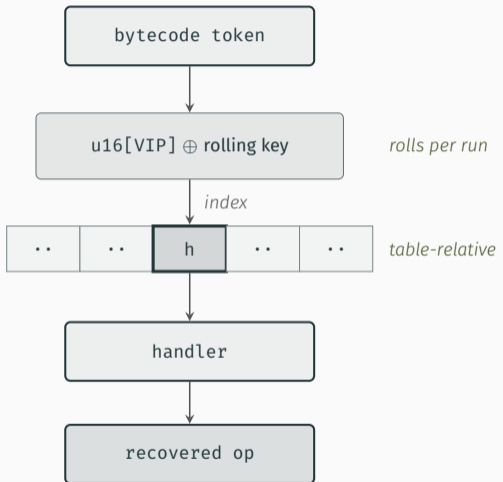
- ✓ process & cross-process memory monitoring
- ✓ physical-memory integrity scanning
- ✓ hypervisor, timing & anti-VM checks
- ✓ API-hook & tampering detection
- ✓ enforcement: telemetry + forced bugcheck

Game DRM

Game DRM: Highest-Tier Target

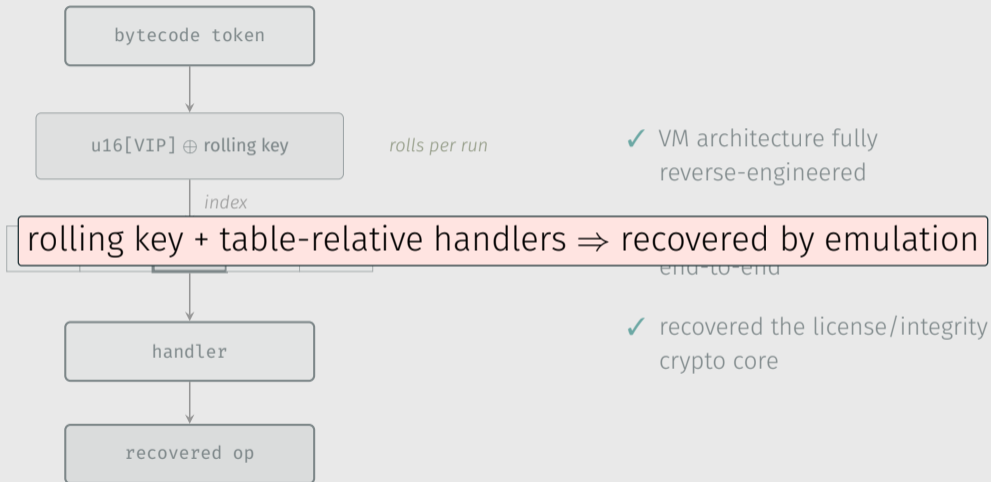


VM in an Older Game Title

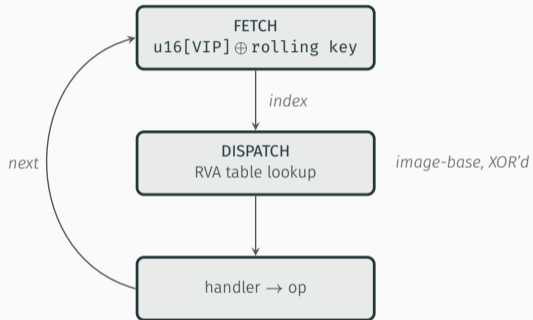


- ✓ VM architecture fully reverse-engineered
- ✓ executed path devirtualized end-to-end
- ✓ recovered the license/integrity crypto core

VM in an Older Game Title



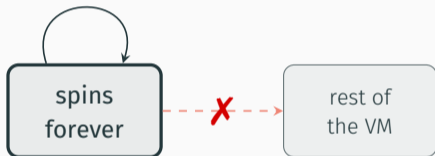
VM in a Recent Game Title



- ✓ VM architecture understood
- ✓ decode & dispatch recovered
- ✗ can't analyze the bytecode
- ✗ requires runtime state
- ✗ stalls under emulation

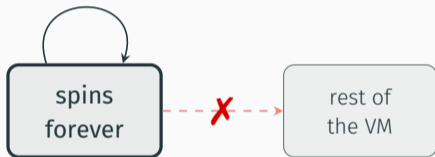
VM in a Recent Game Title: Analysis Walls

anti-emulation loop



VM in a Recent Game Title: Analysis Walls

anti-emulation loop



needs runtime state



VM in a Recent Game Title: Analysis Walls

anti-emulation loop

needs runtime state

static analysis stalls without runtime info

forever

the VM

present

present

Summary: Agentic Virtual Machine Deobfuscation

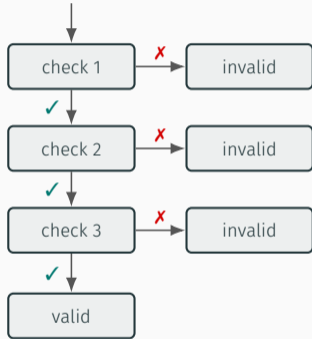
- ✓ **simple VM** — essentially a one-shotted
- ✓ **commercial VM** — full devirtualization in hours
- ✓ **anti-cheat** — devirtualized the majority of $\sim 1,000$ VM-entries in parallel
- ✓ **game DRM** — VM architectures understood; older VM path devirtualized
- ✗ **game DRM** — stalls under emulation and misses runtime information



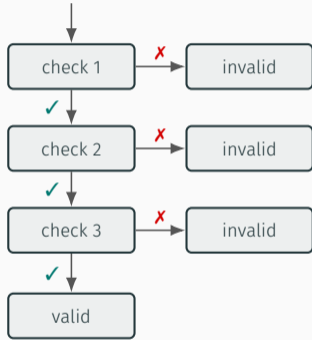
Case Study

Control-Flow Deflattening

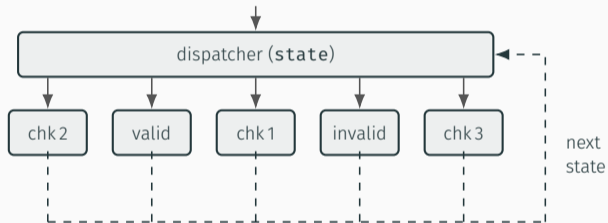
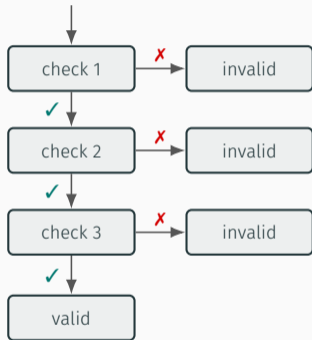
Control-Flow Flattening



Control-Flow Flattening



Control-Flow Flattening



```
_main:  
0 @ 10000046c int32_t var_14 = 0  
1 @ 100000470 int32_t state = 0
```

```
2 @ 100000480 while (true)
```

```
3 @ 100000480 if (state == 0)
```

```
4 @ 1000004c0 _printf("check1\n")  
5 @ 1000004c8 state = 1
```

```
6 @ 100000490 if (state == 1)
```

```
7 @ 1000004d8 _printf("check2\n")  
8 @ 1000004e0 state = 2
```

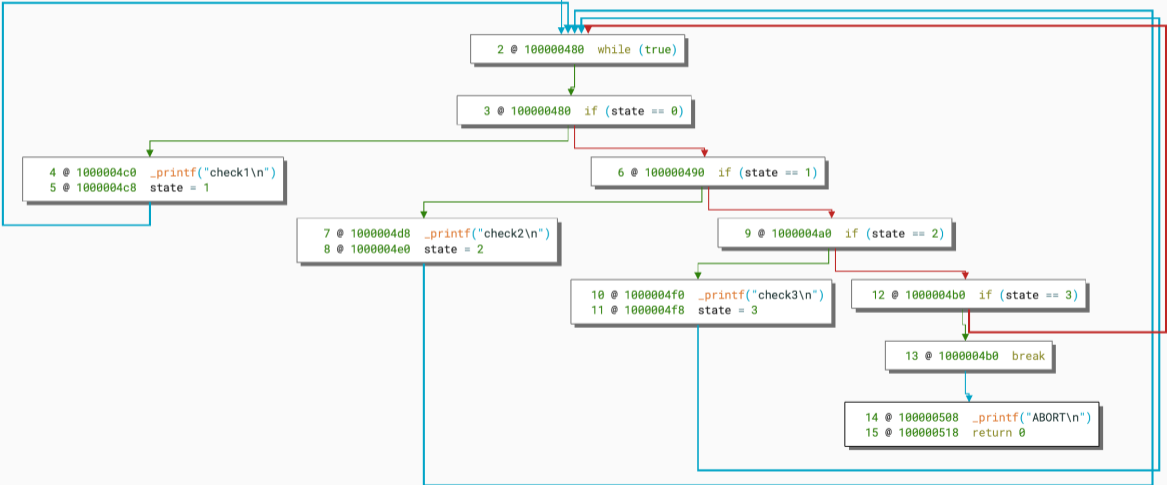
```
9 @ 1000004a0 if (state == 2)
```

```
10 @ 1000004f0 _printf("check3\n")  
11 @ 1000004f8 state = 3
```

```
12 @ 1000004b0 if (state == 3)
```

```
13 @ 1000004b0 break
```

```
14 @ 100000508 _printf("ABORT\n")  
15 @ 100000518 return 0
```



```
_main:  
0 @ 10000046c int32_t var_14 = 0  
1 @ 100000470 int32_t state = 0
```

```
2 @ 100000480 while (true)
```

```
3 @ 100000480 if (state == 0)
```

```
4 @ 1000004c0 _printf("check1\n")  
5 @ 1000004c8 state = 1
```

```
6 @ 100000490 if (state == 1)
```

flattened control flow in Binary Ninja

```
7 @ 1000004d8 _printf("check2\n")  
8 @ 1000004e0 state = 2
```

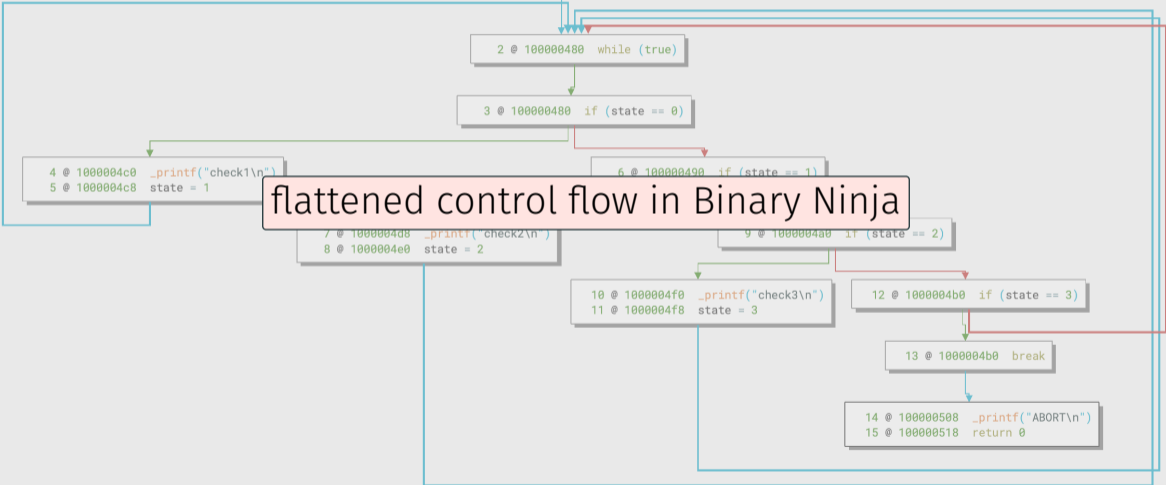
```
9 @ 1000004a8 if (state == 2)
```

```
10 @ 1000004f0 _printf("check3\n")  
11 @ 1000004f8 state = 3
```

```
12 @ 1000004b0 if (state == 3)
```

```
13 @ 1000004b0 break
```

```
14 @ 100000508 _printf("ABORT\n")  
15 @ 100000518 return 0
```



Control-Flow Deflattening Process



**dispatcher &
state var**

locate the
structure

Control-Flow Deflattening Process



dispatcher &
state var

locate the
structure



enumerate
states

state → block
map

Control-Flow Deflattening Process



**dispatcher &
state var**

locate the
structure



**enumerate
states**

state → block
map



**resolve
successors**

next state per
block

Control-Flow Deflattening Process



**dispatcher &
state var**

locate the
structure



**enumerate
states**

state → block
map



**resolve
successors**

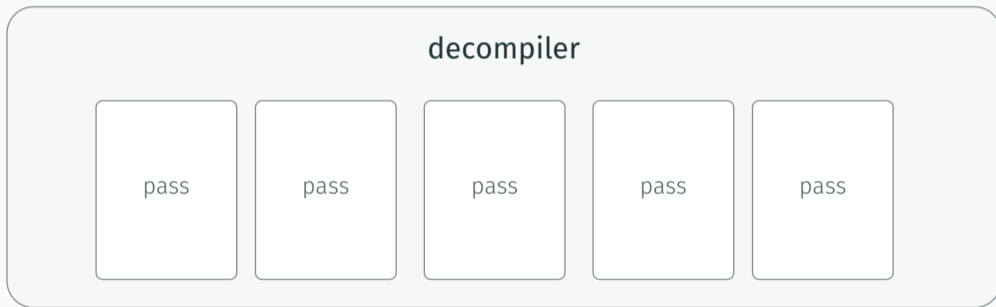
next state per
block



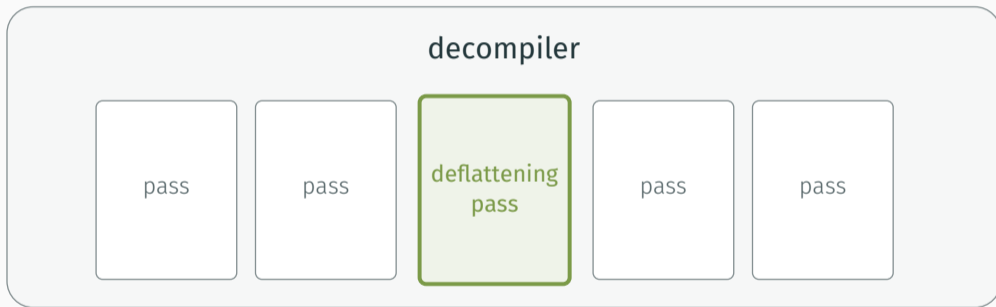
patch

direct edges

Goal: Cleaning Decompiler Output

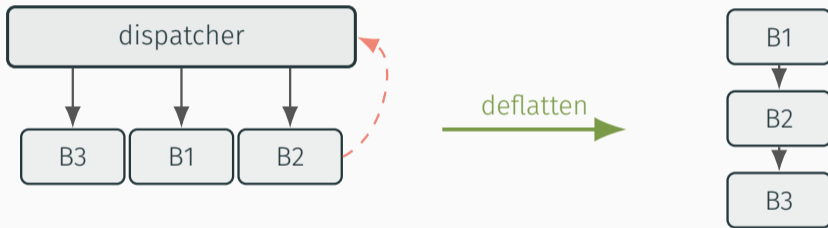


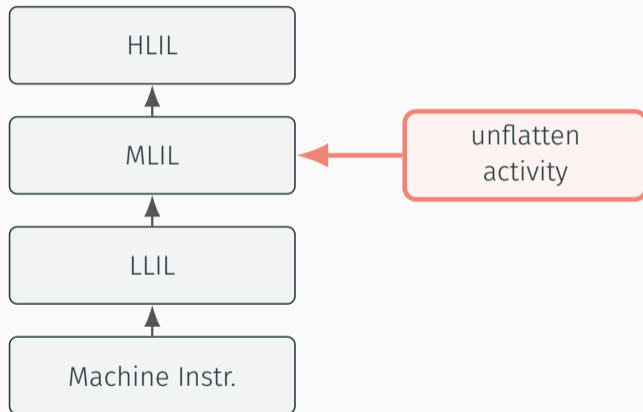
Goal: Cleaning Decompiler Output



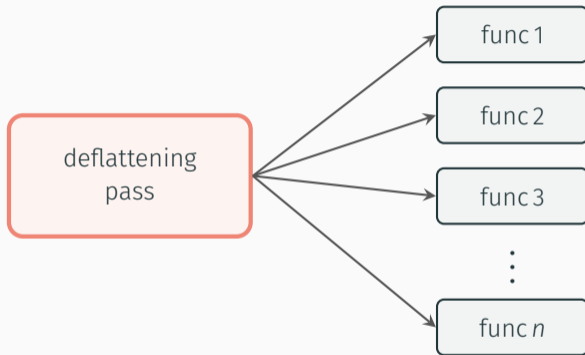
Emotet

Deflating Emotet's Obfuscation

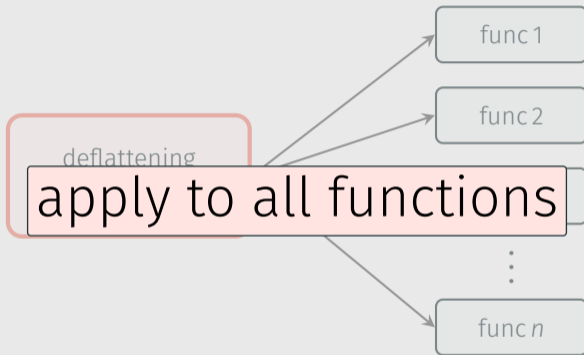




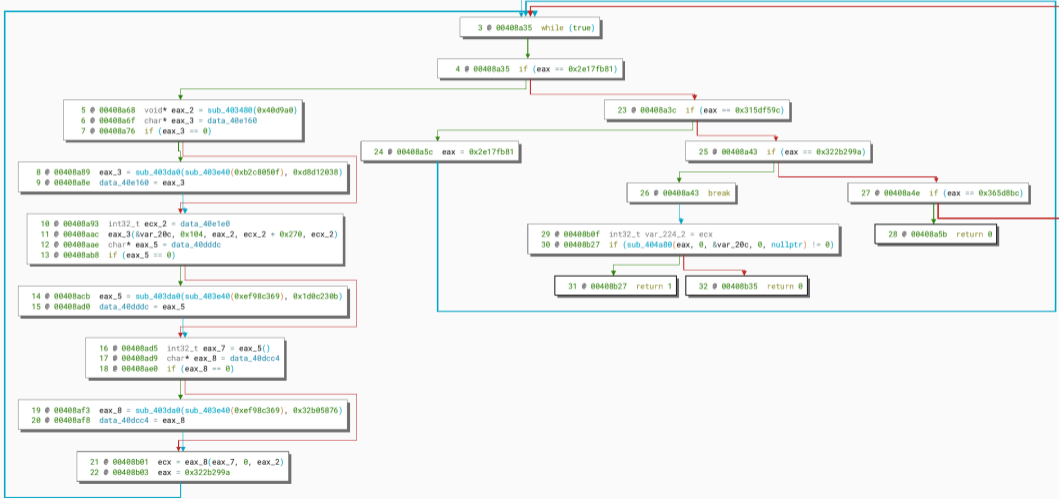
Deflattening pass



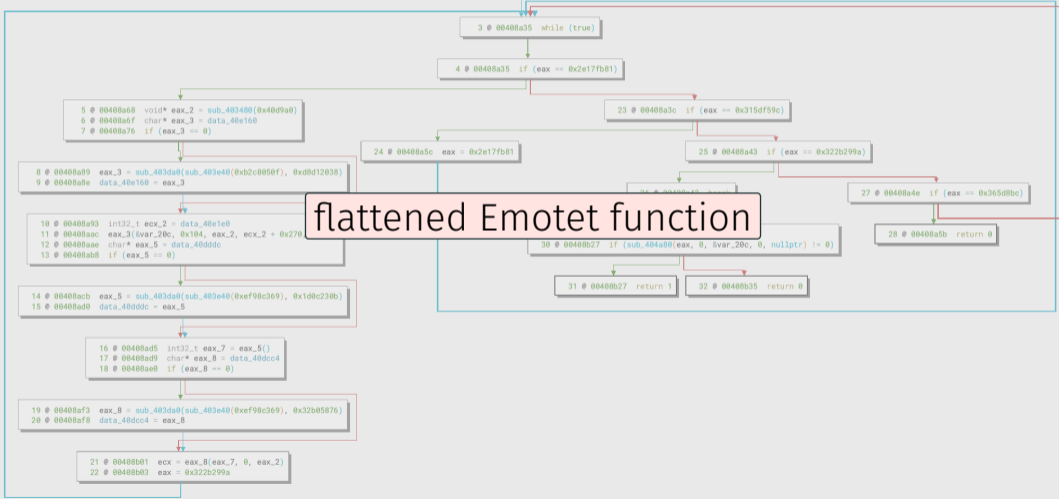
Deflattening pass



```
sub_408a20:  
0 @ 00408a26 int32_t eax = 0x315df59c  
1 @ 00408a35 void var_20c  
2 @ 00408a35 int32_t ecx
```



```
sub_408a20:
0 @ 00408a26 int32_t eax = 0x315df59c
1 @ 00408a35 void var_20c
2 @ 00408a35 int32_t ecx
```



flattened Emotet function

```
// [emotet-unflatten] CFF removed: 4 states, 3 edges redirected, dispatcher root
// 0x408a30
```

```
sub_408a20:
```

```
0 @ 00408a68 void* eax = sub_403480(0x40d9a0)
1 @ 00408a6f char* eax_1 = data_40e160
2 @ 00408a76 if (eax_1 == 0)
```

```
3 @ 00408a89 eax_1 = sub_403da0(sub_403e40(0xb2c8050f), 0xd8d12038)
4 @ 00408a8e data_40e160 = eax_1
```

```
5 @ 00408a93 int32_t ecx_2 = data_40e1e0
6 @ 00408aac void var_20c
7 @ 00408aac eax_1(&var_20c, 0x104, eax, ecx_2 + 0x270, ecx_2)
8 @ 00408aae char* eax_3 = data_40dddc
9 @ 00408ab8 if (eax_3 == 0)
```

```
10 @ 00408acb eax_3 = sub_403da0(sub_403e40(0xef98c369), 0x1d0c230b)
11 @ 00408ad0 data_40dddc = eax_3
```

```
12 @ 00408ad5 int32_t eax_5 = eax_3()
13 @ 00408ad9 char* eax_6 = data_40dcc4
14 @ 00408ae0 if (eax_6 == 0)
```

```
15 @ 00408af3 eax_6 = sub_403da0(sub_403e40(0xef98c369), 0x32b05876)
16 @ 00408af8 data_40dcc4 = eax_6
```

```
17 @ 00408b01 int32_t eax_8
18 @ 00408b01 int32_t ecx_6
19 @ 00408b01 eax_8, ecx_6 = eax_6(eax_5, 0, eax)
20 @ 00408b0f int32_t var_224_1 = ecx_6
21 @ 00408b27 if (sub_404a80(eax_8, 0, &var_20c, 0, nullptr) != 0)
```

```
22 @ 00408b27 return 1
```

```
23 @ 00408b35 return 0
```

```
// [emotet-unflatten] CFF removed: 4 states, 3 edges redirected, dispatcher root
// 0x408a30
```

```
sub_403c20:
```

```
0 @ 00408a68 void* eax = sub_403480(0x40d9a0)
1 @ 00408a6f char* eax_1 = data_40e160
2 @ 00408a76 if (eax_1 == 0)
```

```
3 @ 00408a89 eax_1 = sub_403da0(sub_403e40(0xb2c8050f), 0xd8d12038)
4 @ 00408a8e data_40e160 = eax_1
```

```
5 @ 00408a93 int32_t ecx_2 = data_40e1e0
6 @ 00408aac void var_20c
7 @ 00408aac eax_1(&var_20c, 0x104, eax, ecx_2 + 0x270, ecx_2)
8 @ 00408aae char* eax_3 = data_40dddc
9 @ 00408ab8 if (eax_3 == 0)
```

deflattened - the pass generalizes from one function to all

```
12 @ 00408ad5 int32_t eax_5 = eax_3()
13 @ 00408ad9 char* eax_6 = data_40dcc4
14 @ 00408ae0 if (eax_6 == 0)
```

```
15 @ 00408af3 eax_6 = sub_403da0(sub_403e40(0xef98c369), 0x32b05876)
16 @ 00408af8 data_40dcc4 = eax_6
```

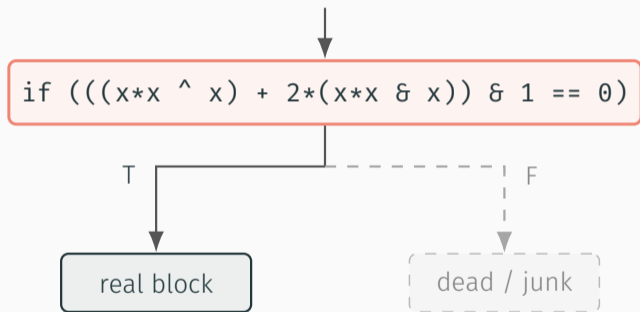
```
17 @ 00408b01 int32_t eax_8
18 @ 00408b01 int32_t ecx_6
19 @ 00408b01 eax_8, ecx_6 = eax_6(eax_5, 0, eax)
20 @ 00408b0f int32_t var_224_1 = ecx_6
21 @ 00408b27 if (sub_404a80(eax_8, 0, &var_20c, 0, nullptr) != 0)
```

```
22 @ 00408b27 return 1
```

```
23 @ 00408b35 return 0
```

FinFisher

FinFisher: MBA-based Opaque Predicates



```

sub_448d08:
0 # 00448c04 int64_t r15
1 # 00448c04 int64_t var_10 = r15
2 # 00448c06 int64_t r14
3 # 00448c06 int64_t var_10 = r14
4 # 00448c08 int64_t r13
5 # 00448c08 int64_t var_20 = r13
6 # 00448c0a int64_t r12
7 # 00448c0a int64_t var_20 = r12
8 # 00448c0c int64_t rbx
9 # 00448c0c int64_t var_30 = rbx
10 # 00448c0e uint32_t rdx = *x52
11 # 00448d1a bool var_32 = ((rdx * (rdx - 1)) & 1) == 0 & 1
12 # 00448d2c bool var_31 = *y53 < 0xa & 1
13 # 00448d30 int32_t var_30 = 0xd147bf6

```

```

14 # 00448d45 while (true)

```

```

15 # 00448d45 int32_t var_40_1 = var_30 - 0xd147bf5
16 # 00448d48 if (var_30 >= 0xd147bf5)

```

```

17 # 00448d71 int32_t var_48_1 = var_30 - 0xd147bf6

```

```

18 # 00448d74 int64_t rdx_3
19 # 00448d74 if (var_30 == 0xd147bf6)

```

```

42 # 00448d5b int32_t var_44_1 = var_30 + 0x6313ba03
43 # 00448d5e if (var_30 == 0x9cec45fd)

```

```

44 # 00448ec5 var_30 = 0x5df8346f

```

```

20 # 00448d60 int32_t rax_10 = -0x5313ba03
21 # 00448d6a rdx_3_b = 1
22 # 00448d72 rdx_3_b = 1
23 # 00448d79 rdx_3_b = 1
24 # 00448e06 if (((((var_32 & 1) ^ (var_31 & 1)) | (((var_32 ^ 1) | (var_31 ^ 1)) ^ 1) & 1) != 0)

```

```

25 # 00448e06 rax_10 = 0x5df8346f

```

```

26 # 00448e09 var_30 = rax_10

```

```

27 # 00448d87 int32_t var_4c_1 = var_30 - 0x1ec001ad
28 # 00448d8a if (var_30 == 0x1ec001ad)

```

```

29 # 00448d8a break

```

```

45 # 00448ec4 return 0

```

```

30 # 00448d9d int32_t var_50_1 = var_30 - 0x5df8346f
31 # 00448da0 if (var_30 == 0x5df8346f)

```

```

32 # 00448e11 int32_t rax_11 = -0x5313ba03
33 # 00448e1b rdx_3_b = 1
34 # 00448e2b uint32_t r8_1 = *x52
35 # 00448e2c bool r14_1 = ((r8_1 * (r8_1 - 1)) & 1) == 0 ^ 1
36 # 00448e73 bool r15_1 = *y53 < 0xa ^ 1
37 # 00448e77 rdx_3_b = 0
38 # 00448e9d rdx_3_b = 1
39 # 00448eaa if (((((r14_1 & 1) ^ (r15_1 & 1)) | (((r14_1 | r15_1) ^ 1) & 1) & 1) != 0)

```

```

40 # 00448eaa rax_11 = 0x1ec001ad

```

```

41 # 00448ead var_30 = rax_11

```

```

sub_448cd8:
0 # 0044bc04 int64_t r15
1 # 0044bc04 int64_t var_10 = r15
2 # 0044bc06 int64_t r14
3 # 0044bc06 int64_t var_10 = r14
4 # 0044bc08 int64_t r13
5 # 0044bc08 int64_t var_20 = r13
6 # 0044bc0a int64_t r12
7 # 0044bc0a int64_t var_20 = r12
8 # 0044bc0c int64_t rbx
9 # 0044bc0c int64_t var_30 = rbx
10 # 0044bc0e uint32_t rdx = *x52
11 # 0044bd1a bool var_32 = ((rdx * (rdx - 1)) & 1) == 0 & 1
12 # 0044bd2c bool var_31 = *y53 < 0xa & 1
13 # 0044bd30 int32_t var_30 = 0xd147bf6

```

```

14 # 0044bd45 while (true)

```

```

15 # 0044bd45 int32_t var_40_1 = var_30 - 0xd147bf5
16 # 0044bd48 if (var_30 >= 0xd147bf5)

```

```

17 # 0044bd71 int32_t var_48_1 = var_30 - 0xd147bf6

```

```

42 # 0044bd5b int32_t var_44_1 = var_30 + 0x6313ba03
43 # 0044bd5e if (var_30 == 0x9cec45fd)

```

```

44 # 0044bec5 var_30 = 0x5df8346f

```

a flattened FinFisher loader stage

```

20 # 0044bd00 int32_t rax_10 = -0x5313ba03
21 # 0044bd0a rdx_3_b = 1
22 # 0044bd02 rdx_3_b = 1
23 # 0044bd09 rdx_3_b = 1
24 # 0044be06 if (((((var_32 & 1) ^ (var_31 & 1)) | (((var_32 * 1) | (var_31 * 1)) * 1) & 1)) & 1) != 0)

```

```

25 # 0044be06 rax_10 = 0x5df8346f

```

```

26 # 0044be09 var_30 = rax_10

```

```

27 # 0044bd07 int32_t var_4c_1 = var_30 - 0x1ec001ad
28 # 0044bd0a if (var_30 == 0x1ec001ad)

```

```

29 # 0044bd0a break

```

```

45 # 0044bec4 return 0

```

```

30 # 0044bd0d int32_t var_50_1 = var_30 - 0x5df8346f
31 # 0044bd0e if (var_30 == 0x5df8346f)

```

```

32 # 0044be11 int32_t rax_11 = -0x5313ba03
33 # 0044be1b rdx_3_b = 1
34 # 0044be2b uint32_t r8_1 = *x52
35 # 0044becc bool r14_1 = ((r8_1 * (r8_1 - 1)) & 1) == 0 * 1
36 # 0044be73 bool r15_1 = *y53 < 0xa * 1
37 # 0044be77 rdx_3_b = 0
38 # 0044be9c rdx_3_b = 1
39 # 0044beaa if (((((r14_1 & 1) * (r15_1 & 1)) | (((r14_1 | r15_1) * 1) & 1)) & 1) != 0)

```

```

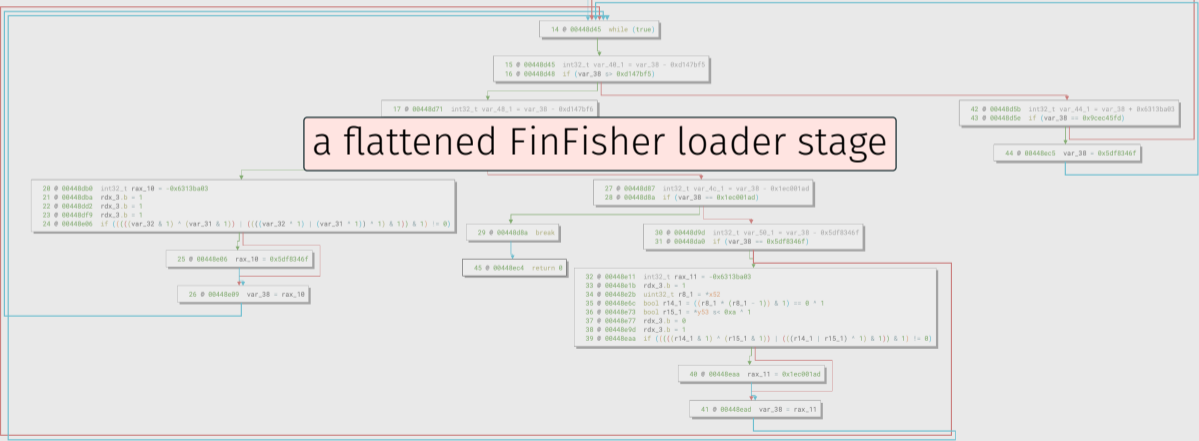
40 # 0044beaa rax_11 = 0x1ec001ad

```

```

41 # 0044bead var_30 = rax_11

```



sub_448cd0:

```
0 @ 00448ceb uint32_t rdx = *x52
1 @ 00448d1a bool var_32 = ((rdx * (rdx - 1)) & 1) == 0 & 1
2 @ 00448db0 int32_t rax = -0x6313ba03
3 @ 00448dc0 bool rdi_4 = *y53 s< 0xa & 1
4 @ 00448e06 if (((((var_32 & 1) ^ (rdi_4 & 1)) | (((var_32 ^ 1) | (rdi_4 ^ 1)) ^ 1) & 1)) & 1) != 0
```

```
5 @ 00448e06 rax = 0x5df8346f
```

```
6 @ 00448e09 int32_t var_38 = rax
7 @ 00448e11 int32_t rax_1 = -0x6313ba03
8 @ 00448e2b uint32_t r8_1 = *x52
9 @ 00448e6c bool r14 = ((r8_1 * (r8_1 - 1)) & 1) == 0 ^ 1
10 @ 00448e73 bool r15 = *y53 s< 0xa ^ 1
11 @ 00448eaa if (((((r14 & 1) ^ (r15 & 1)) | ((r14 | r15) ^ 1) & 1)) & 1) != 0
```

```
12 @ 00448eaa rax_1 = 0x1ec001ad
```

```
13 @ 00448ead int32_t var_38_1 = rax_1
14 @ 00448ec4 return 0
```

sub_448cd0:

```
0 @ 00448ceb uint32_t rdx = *x52
1 @ 00448d1a bool var_32 = ((rdx * (rdx - 1)) & 1) == 0 & 1
2 @ 00448db0 int32_t rax = -0x6313ba03
3 @ 00448dc0 bool rdi_4 = *y53 s< 0xa & 1
4 @ 00448e06 if (((((var_32 & 1) ^ (rdi_4 & 1)) | (((var_32 ^ 1) | (rdi_4 ^ 1)) ^ 1) & 1)) & 1) != 0
```

```
5 @ 00448e06 rax = 0x5df8346f
```

successors recovered across the FinFisher modules

```
8 @ 00448e2b uint32_t r8_1 = *x52
9 @ 00448e6c bool r14 = ((r8_1 * (r8_1 - 1)) & 1) == 0 ^ 1
10 @ 00448e73 bool r15 = *y53 s< 0xa ^ 1
11 @ 00448eaa if (((((r14 & 1) ^ (r15 & 1)) | ((r14 | r15) ^ 1) & 1)) & 1) != 0
```

```
12 @ 00448eaa rax_1 = 0x1ec001ad
```

```
13 @ 00448ead int32_t var_38_1 = rax_1
14 @ 00448ec4 return 0
```

Summary: Control-Flow Deflattening

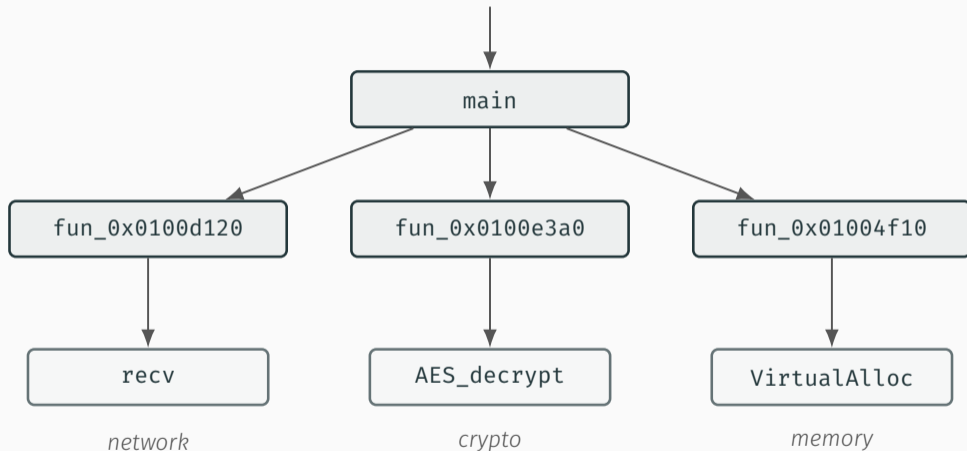
- ✓ agents can write **decompiler-based** deflattening passes
- ✗ each pass stays **sample-specific**



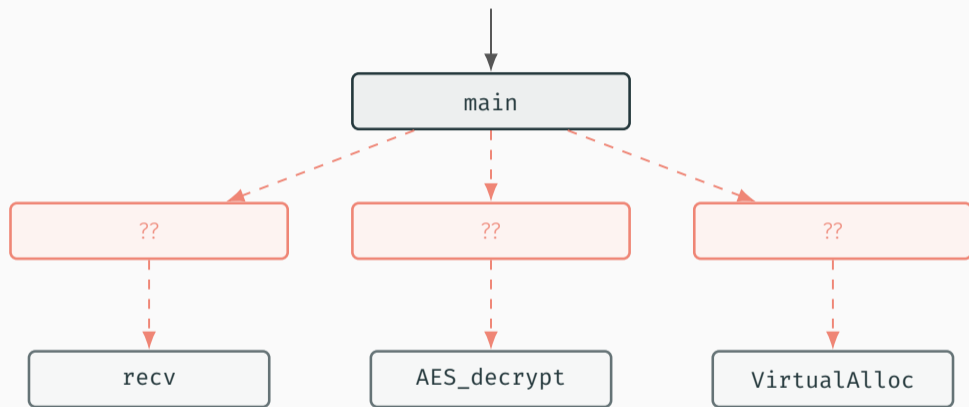
Case Study

Global Call Obfuscation

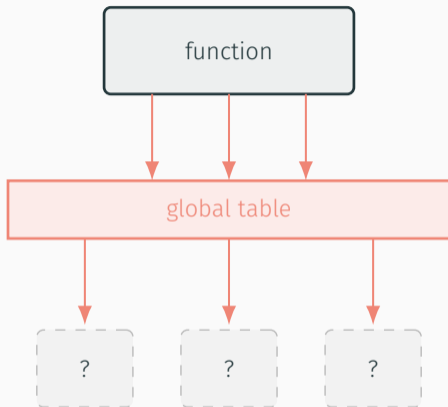
Call Graph Obfuscation

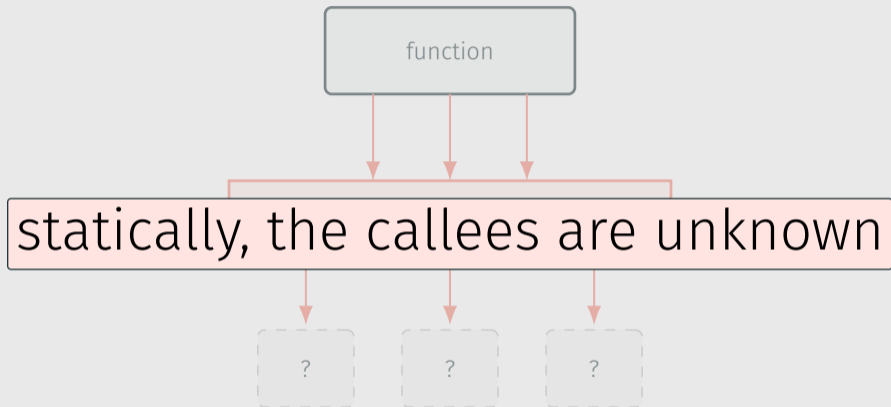


Call Graph Obfuscation



Call Graph Obfuscation



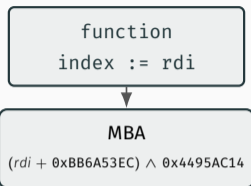


A Modern DRM System

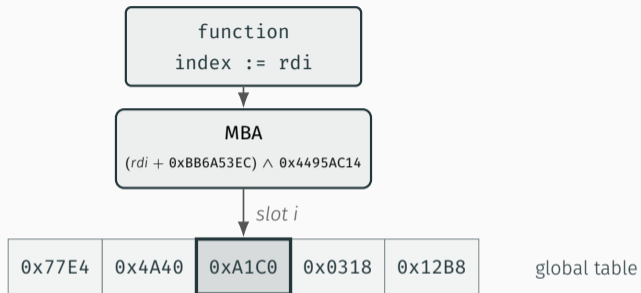
Call Obfuscation Details

```
function  
index := rdi
```

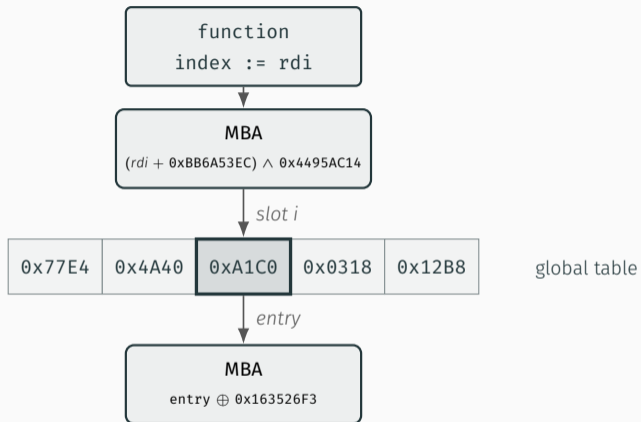
Call Obfuscation Details



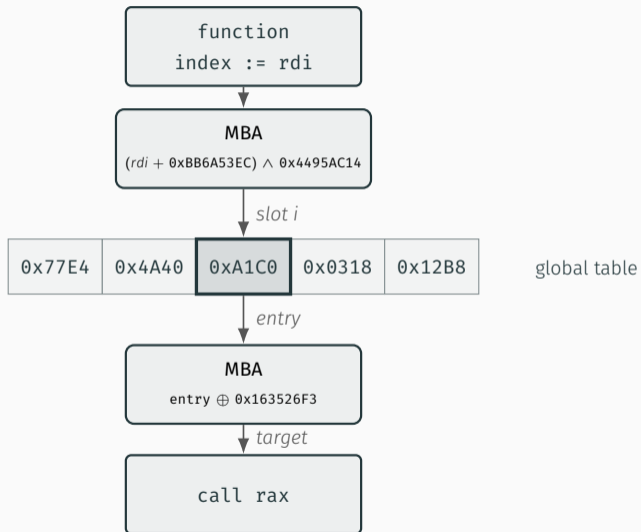
Call Obfuscation Details



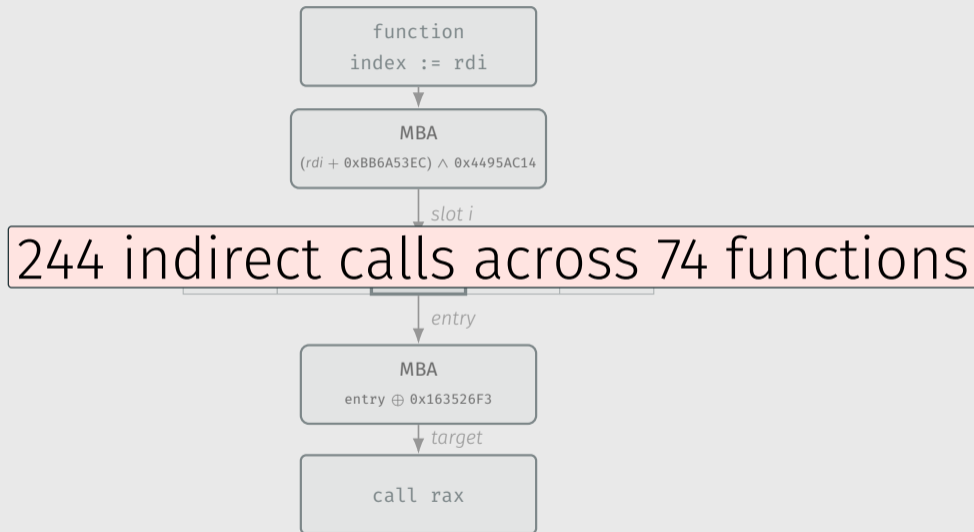
Call Obfuscation Details



Call Obfuscation Details



Call Obfuscation Details



Agentic Recovery: False Starts



Most of the calls in this binary are indirect - recover their targets.



Agentic Recovery: False Starts



Most of the calls in this binary are indirect - recover their targets.

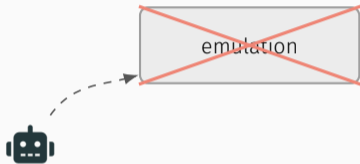


emulation

Agentic Recovery: False Starts



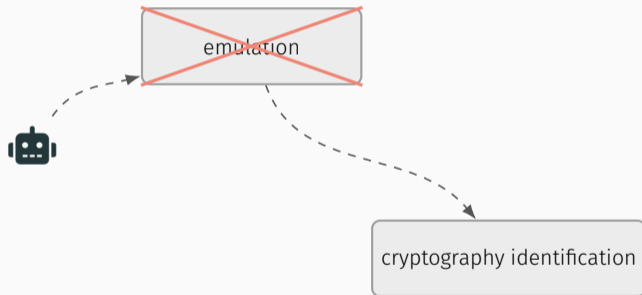
Most of the calls in this binary are indirect - recover their targets.



Agentic Recovery: False Starts



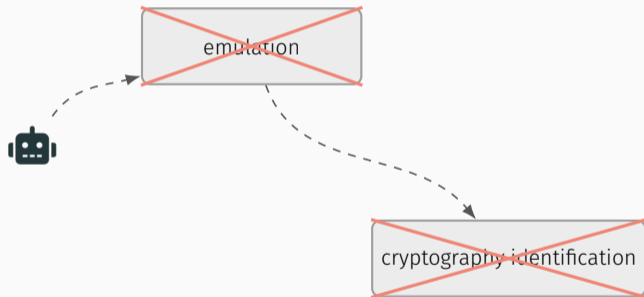
Most of the calls in this binary are indirect - recover their targets.



Agentic Recovery: False Starts



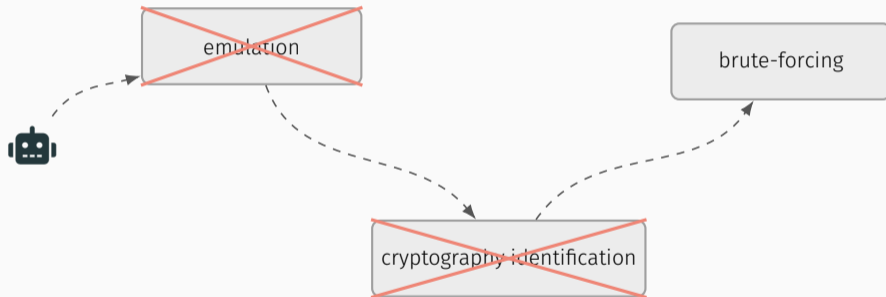
Most of the calls in this binary are indirect - recover their targets.



Agentic Recovery: False Starts



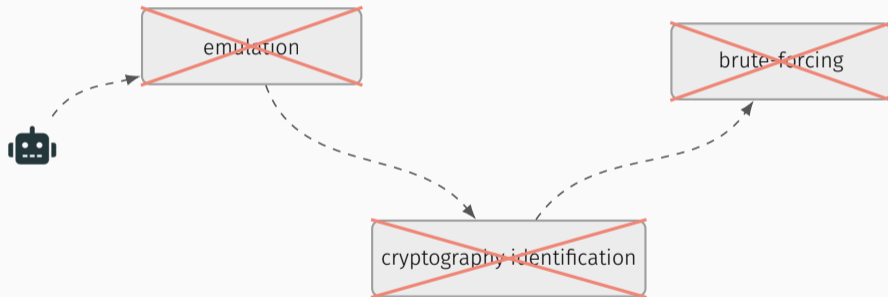
Most of the calls in this binary are indirect - recover their targets.



Agentic Recovery: False Starts



Most of the calls in this binary are indirect - recover their targets.



Agentic Recovery: False Starts



Most of the calls in this binary are indirect - recover their targets.

~~emulation~~

~~brute-forcing~~

required manual guidance

~~cryptology identification~~

Call-Graph Recovery Process



recover the
global table

identification,
boundaries

Call-Graph Recovery Process



recover the
global table

identification,
boundaries



propagate the
entries

const-prop the
slots

Call-Graph Recovery Process



recover the
global table

identification,
boundaries



propagate the
entries

const-prop the
slots



simplify MBA

verify with z3

Call-Graph Recovery Process



recover the
global table

identification,
boundaries



propagate the
entries

const-prop the
slots



simplify MBA

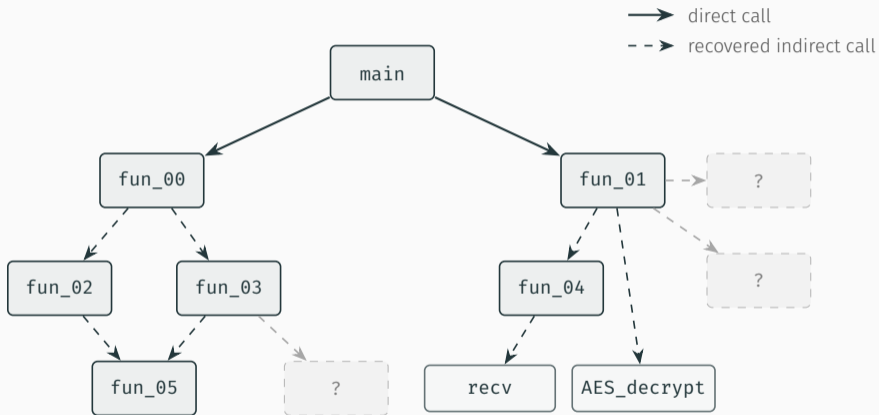
verify with z3



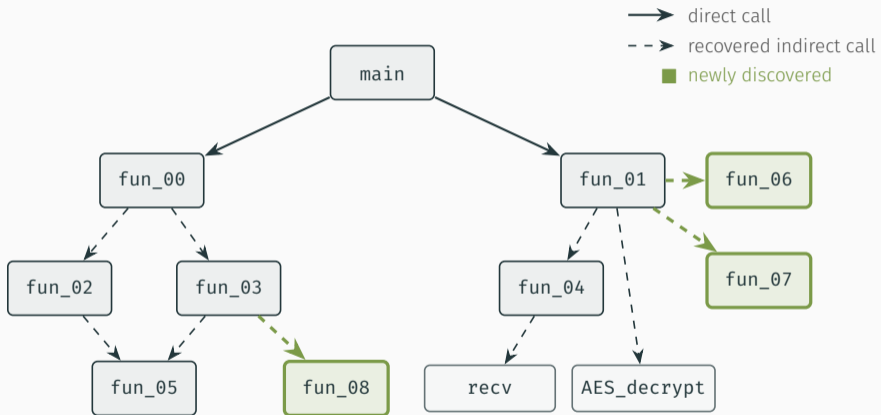
recover the
edges

patch calls

The Recovered Call Graph



The Recovered Call Graph





function boundaries
& more functions discovered



parameter detection
across the whole graph

`_main:`

```
0 @ 1001dbb3b int64_t rax = *___stack_chk_guard
1 @ 1001dbc22 int32_t var_60 = (((zx.d(&var_60) & 0x99a5cce8) + (zx.d(&var_60) | 0x665a3310) - &var_60)
1 @ 1001dbc22     ^ 0x1bcd0820) * 0x4b05c2c3 + 2
2 @ 1001dbc34 char rax_11 = (((data_100338778.d).b - (data_100338958.d).b) ^ 0x43) * 0x45
3 @ 1001dbc68 (-0x4ada058b + (&jump_table_100330c20)[zx.q((zx.d(rax_11)
3 @ 1001dbc68     ^ zx.d(*(zx.q(*(zx.q(rax_11) + &data_1002b8ea0)) ^ 0xeb) + &data_1002d6be0))) + 0x1d8]))(&var_60,
3 @ 1001dbc68     0, &data_1002b8ea0)
4 @ 1001dbc97 int32_t rcx_8 = 0
5 @ 1001dbc9e int32_t var_5c
6 @ 1001dbc9e rcx_8.b = var_5c == 0x3145c2fe
7 @ 1001dbcb9 switch (rcx_8)
```

`_main:`

```
0 @ 1001dbb3b int64_t rax = *___stack_chk_guard
1 @ 1001dbc22 int32_t var_60 = (((zx.d(&var_60) & 0x99a5cce8) + (zx.d(&var_60) | 0x665a3310) - &var_60)
1 @ 1001dbc22     ^ 0x1bcd0820) * 0x4b05c2c3 + 2
2 @ 1001dbc34 char rax_11 = (((data_100338778.d).b - (data_100338958.d).b) ^ 0x43) * 0x45
3 @ 1001dbc68 (-0x4ada058b +
3 @ 1001dbc68     ^ zx.d(*(z
3 @ 1001dbc68     0, &data_1002b8ea0)
4 @ 1001dbc97 int32_t rcx_8 = 0
5 @ 1001dbc9e int32_t var_5c
6 @ 1001dbc9e rcx_8.b = var_5c == 0x3145c2fe
7 @ 1001dbcb9 switch (rcx_8)
```

call hidden behind indirection

`_main:`

```
0 @ 1001dbb3b int64_t rax = *___stack_chk_guard
1 @ 1001dbbac int64_t var_b8
2 @ 1001dbbac char rax_3 = (&var_b8 * 0x8a + 0xf) ^ 0x43
3 @ 1001dbc65 sub_100058530(0x1f, 0, &data_1002b8ea0,
3 @ 1001dbc65     zx.q(*(zx.q((((data_100338778.d).b - (data_100338958.d).b) ^ 0x43) * 0x45) + &data_1002b8ea0))
3 @ 1001dbc65     ^ 0xeb,
3 @ 1001dbc65     -0x385c84ff + (&data_100330c20)[zx.q((zx.d(rax_3)
3 @ 1001dbc65     ^ zx.d(*(zx.q(*(zx.q(rax_3) + &data_1002b8ea0)) ^ 0x80) + &data_1002d6be0))) + 0x33d]])
4 @ 1001dbc97 int32_t rcx_6 = 0
5 @ 1001dbc9e int32_t var_5c
6 @ 1001dbc9e rcx_6.b = var_5c == 0x3145c2fe
7 @ 1001dbcb9 switch (rcx_6)
```

`_main:`

```
0 @ 1001dbb3b int64_t rax = *___stack_chk_guard
1 @ 1001dbbac int64_t var_b8
2 @ 1001dbbac char rax_3 = (&var_b8 * 0x8a + 0xf) ^ 0x43
3 @ 1001dbc65 sub_100058530(0x1f, 0, &data_1002b8ea0,
3 @ 1001dbc65     zx.q(*(zx.q((((q
3 @ 1001dbc65     ^ 0xeb,
3 @ 1001dbc65     -0x385c84ff + (&data_10000020)[zx.q((zx.d(rax_0)
3 @ 1001dbc65     ^ zx.d(*(zx.q(*(zx.q(rax_3) + &data_1002b8ea0))) ^ 0x80) + &data_1002d6be0))) + 0x33d]))
4 @ 1001dbc97 int32_t rcx_6 = 0
5 @ 1001dbc9e int32_t var_5c
6 @ 1001dbc9e rcx_6.b = var_5c == 0x3145c2fe
7 @ 1001dbcb9 switch (rcx_6)
```

patched call instruction

Summary: Global Call Obfuscation

- ✓ call graph hidden behind **indirect calls** through one global table
- ✗ agents get easily confused, we had to step in
- ✓ successfully broken with constant propagation and **MBA simplification**



Recap

What Worked

- ✓ **commercial VM** — full devirtualization, weeks → hours
- ✓ **anti-cheat** — ~1,000 VM-entries devirtualized in parallel
- ✓ **game DRM (older)** — VM architecture recovered, path devirtualized
- ✓ **deflattening** — agents write decompiler-based passes
- ✓ **call obfuscation** — recovered via constant propagation + MBA

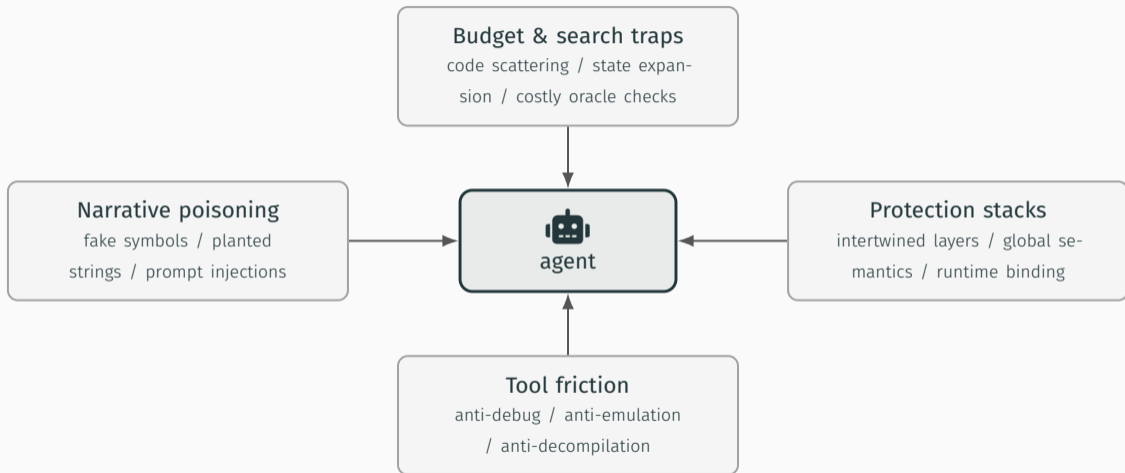
What Did Not Work

- ✗ commercial VM — lost without targeted guidance
- ✗ call obfuscation — agents get confused, we had to step in
- ✗ deflating — each pass stays sample-specific
- ✗ game DRM (newer) — stalls under emulation, meaning is runtime-bound



Anti-Agentic Obfuscation

Anti-Agentive Patterns





Wrap-Up

Takeaways

1. Many formerly tedious workflows become cheap to iterate.
2. They still require analyst guidance.
3. The agent becomes an orchestrator of tooling.
4. Defenses must account for agent-assisted attackers.

Agents change the economics of deobfuscation.

Summary

- agents are a force multiplier for deobfuscation
- obfuscation must scale towards agent-assisted attackers

Tim Blazytko



@mr_phrazer



<https://synthesis.to>

Nicolò Altamura



@nicolodev



<https://nicolo.dev>