

Breaking State-of-the-Art Binary Code Obfuscation via Program Synthesis

Black Hat Asia, Singapore

March 22, 2018

Tim Blazytko, @mr_phrazer
<http://synthesis.to>

Moritz Contag, @dwuid
<https://dwuid.com>

Chair for Systems Security
Ruhr-Universität Bochum
<firstname.lastname>@rub.de



Syntia: Synthesizing the Semantics of Obfuscated Code

**Tim Blazytko, Moritz Contag, Cornelius Aschermann,
and Thorsten Holz, *Ruhr-Universität Bochum***

<https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/blazytko>

**This paper is included in the Proceedings of the
26th USENIX Security Symposium
August 16–18, 2017 • Vancouver, BC, Canada**

ISBN 978-1-931971-40-9

- ❓ Obfuscated code, semantics?
- 🎓 Traditional deobfuscation techniques
- ➔ Orthogonal approach

Motivation

Prevent **Complicate** reverse engineering attempts.

- Intellectual Property
- Malicious Payloads
- Digital Rights Management

Motivation

Prevent **Complicate** reverse engineering attempts.

- Intellectual Property
- Malicious Payloads
- Digital Rights Management

“We achieved our goals. We were uncracked for **13 whole days.**”

– Martin Slater, 2K Australia, on *BioShock* (2007).

How to protect software?

Approaches

Abuse shortcomings of file parsers and other tools of the trade.

- `fld tbyte ptr [__bad_values]` crashing OllyDbg 1.10.
- Fake `SizeOfImage` crashing process dumpers.

Approaches

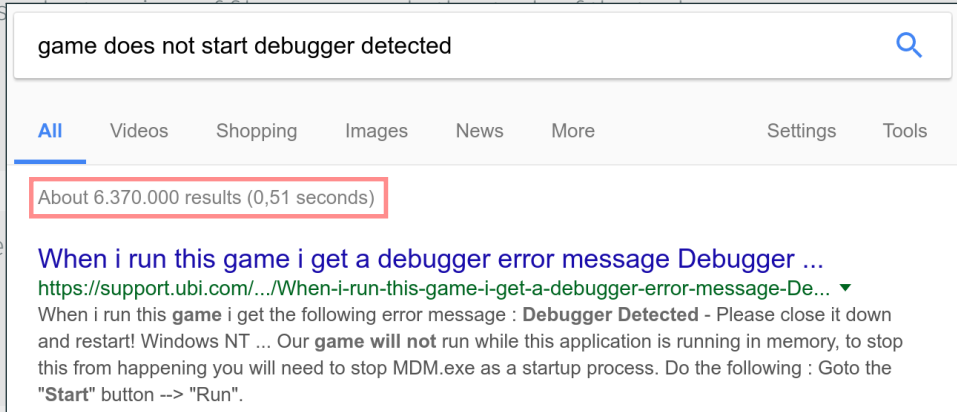
Abuse shortcomings of file parsers and other tools of the trade.

- `fld tbyte ptr [__bad_values]` crashing OllyDbg 1.10.
- Fake `SizeOfImage` crashing process dumpers.

Detect artifacts of the debugging process.

- `PEB.BeingDebugged` bit being set.
- `int 2D` and exception handling in debuggers.

Approaches



Requirements

1. We want the technique to be *semantics-preserving*.

Preserve the observable behavior of the application.

Requirements

1. We want the technique to be *semantics-preserving*.
2. We want to avoid external dependencies, focus on code only.

Assume white-box attack scenario.

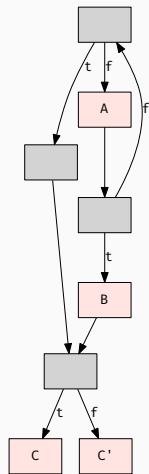
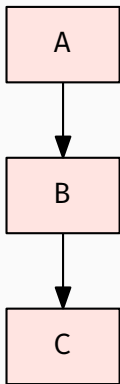
Requirements

1. We want the technique to be *semantics-preserving*.
2. We want to avoid external dependencies, focus on code only.
3. We want techniques where **effort(deploy) $\ll$ effort(attack)**.

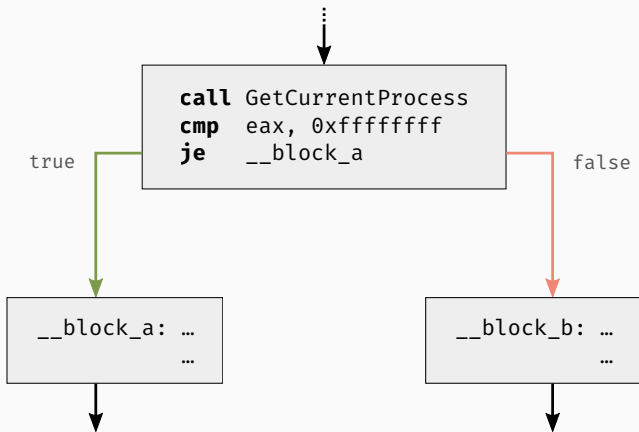
Anti-Debugging tricks are effort **1:1**.

Code Obfuscation Techniques

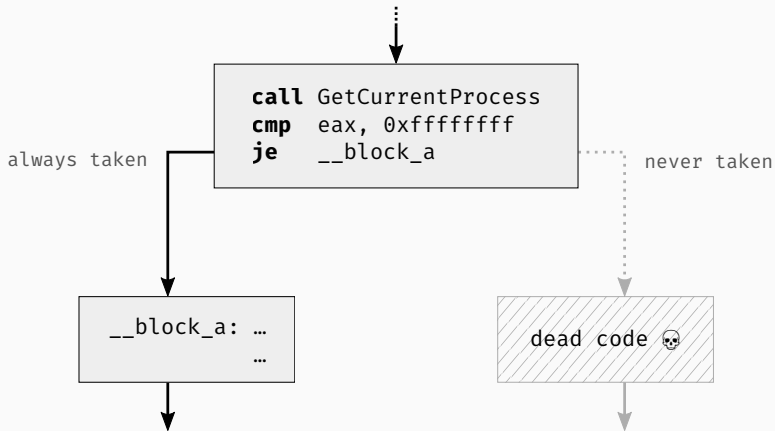
Opaque Predicates



Opaque Predicates

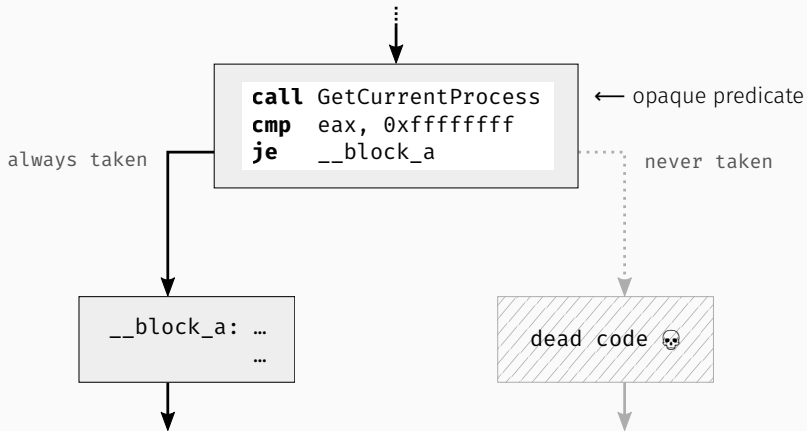


Opaque Predicates



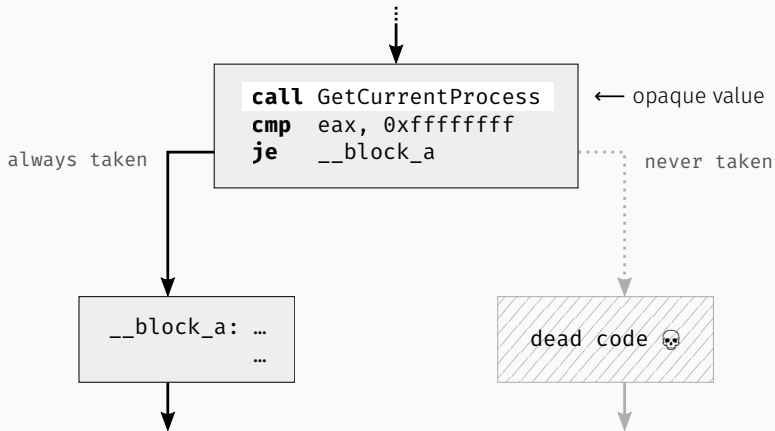
Opaque True Predicate

Opaque Predicates



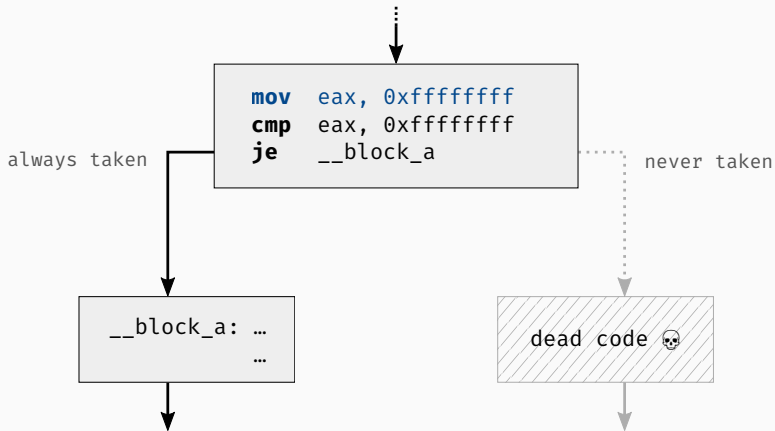
Opaque True Predicate

Opaque Predicates



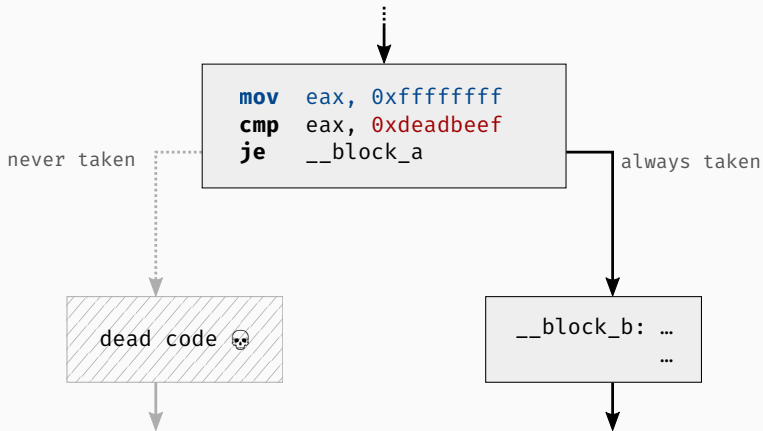
Opaque True Predicate

Opaque Predicates



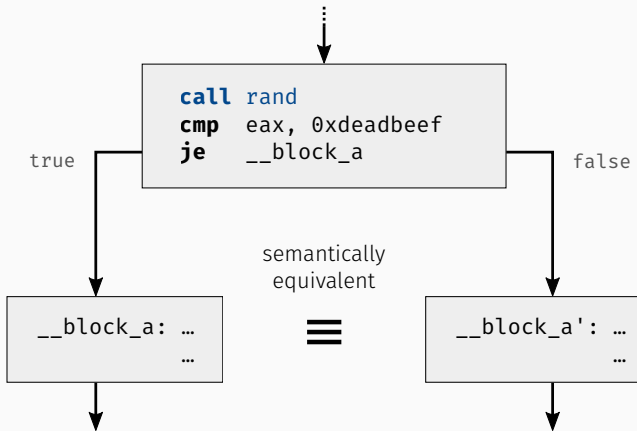
Opaque True Predicate

Opaque Predicates



Opaque False Predicate

Opaque Predicates



Random Opaque Predicate
duplicated block

Opaque Predicates

- ⊕ Increase in complexity (branch count, McCabe)
- ⊕ Can be built on hard problems (e.g., aliasing)
- ⊕ Forces analyst to **encode additional knowledge**
- ⊕ Hard to solve statically

⚠ Examples

- `GetCurrentProcess()` $\Rightarrow -1$
- `fldpi1` $\Rightarrow \text{st}(0) = \pi$
- $x^2 \geq 0 \quad \forall x$
- $x + 1 \neq x \quad \forall x$
- pointer *A must-alias* pointer B
- `checksum(code)` = `0x1c43b5cf`

Opaque Predicates

- ⊕ Increase in complexity (branch count, McCabe)
- ⊕ Can be built on hard problems (e.g., aliasing)
- ⊕ Forces analyst to encode additional knowledge
- ⊕ Hard to solve statically
- ⊖ Solved for free using **concrete execution traces**

⚠ Examples

- `GetCurrentProcess()` $\Rightarrow -1$
- `fldpi1` $\Rightarrow \text{st}(0) = \pi$
- $x^2 \geq 0 \quad \forall x$
- $x + 1 \neq x \quad \forall x$
- pointer *A must-alias* pointer B
- `checksum(code) = 0x1c43b5cf`

Code Obfuscation Techniques

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
```

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
```

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
```



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
```

Virtual Machines

```
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
```

Virtual Machines

```
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 65 2e de ad be ef
```



Virtual Machines

Core Components

VM Entry/Exit Context Switch: native context $\Leftrightarrow$ virtual context

VM Dispatcher Fetch–Decode–Execute loop

Handler Table Individual VM ISA instruction semantics

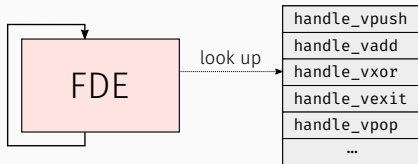
- **Entry** Copy native context (registers, flags) to VM context.
- **Exit** Copy VM context back to native context.
- Mapping from native to virtual registers is often 1:1.

Virtual Machines

Core Components

VM Entry/Exit	Context Switch: native context $\Leftrightarrow$ virtual context
VM Dispatcher	Fetch-Decode-Execute loop
Handler Table	Individual VM ISA instruction semantics

1. Fetch and decode instruction
2. Forward virtual instruction pointer
3. Look up handler for opcode in handler table
4. Invoke handler

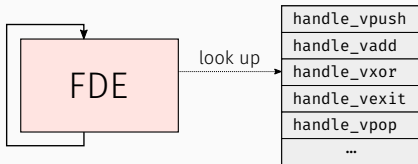


Virtual Machines

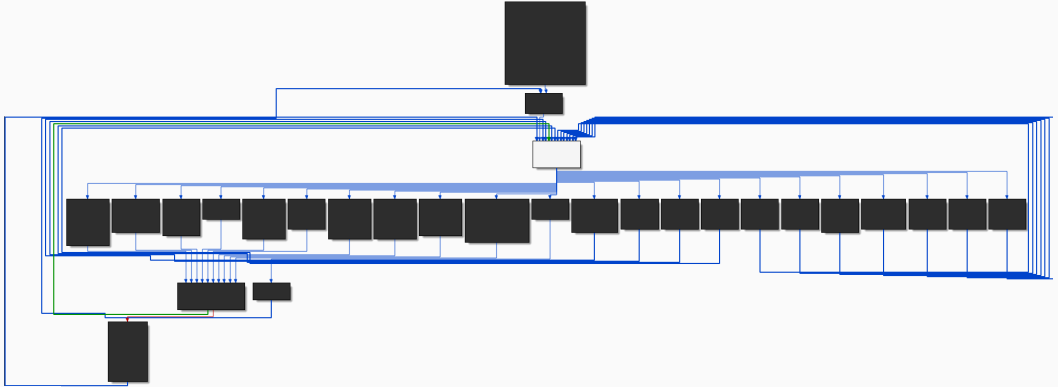
Core Components

VM Entry/Exit	Context Switch: native context $\Leftrightarrow$ virtual context
VM Dispatcher	Fetch-Decode-Execute loop
Handler Table	Individual VM ISA instruction semantics

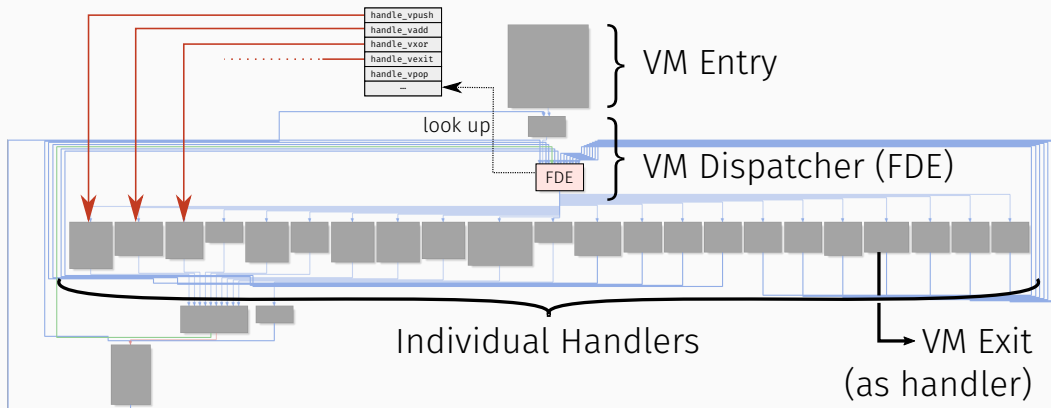
- Table of function pointers indexed by opcode
- One handler per virtual instruction
- Each handler decodes operands and updates VM context



Virtual Machines



Virtual Machines



Virtual Machines

```
__vm_dispatcher:  
    mov    bl, [rsi]  
    inc    rsi  
    movzx  rax, bl  
    jmp    __handler_table[rax * 8]
```

VM Dispatcher

`rsi` – virtual instruction pointer
`rbp` – VM context

Virtual Machines

```
__vm_dispatcher:
    mov    bl, [rsi]
    inc    rsi
    movzx  rax, bl
    jmp    __handler_table[rax * 8]
```

VM Dispatcher

`rsi` – virtual instruction pointer
`rbp` – VM context

```
__handle_vnor:
    mov    rcx, [rbp]
    mov    rbx, [rbp + 4]
    not    rcx
    not    rbx
    and    rcx, rbx
    mov    [rbp + 4], rcx
    pushf
    pop    [rbp]
    jmp    __vm_dispatcher
```

Handler performing **nor**
(with flag side-effects)

Virtual Machine Hardening

Hardening Technique #1 – Obfuscating individual VM components.

- Handlers are *conceptually simple*.

Hardening Technique #1 – Obfuscating individual VM components.

- Handlers are *conceptually simple*.
- Apply traditional code obfuscation transformations:
 - Substitution (`mov rax, rbx` $\Rightarrow$ `push rbx; pop rax`)
 - Opaque Predicates
 - Junk Code
 - ...

```
mov eax, dword [rbp]
mov ecx, dword [rbp+4]
cmp r11w, r13w
sub rbp, 4
not eax
clc
cmc
cmp rdx, 0x28b105fa
not ecx
cmp r12b, r9b
```

Hardening Technique #2 – Duplicating VM handlers.

- Handler table is typically indexed using one byte (= 256 entries).

Hardening Technique #2 – Duplicating VM handlers.

- Handler table is typically indexed using one byte (= 256 entries).
- **Idea:** *Duplicate* existing handlers to populate full table.
- Use traditional obfuscation techniques to impede *code similarity* analyses.

Goal: Increase workload of reverse engineer.

handle_vpush
handle_vadd
handle_vnor
handle_vpop

handle_vpush
handle_vadd
handle_vnor
handle_vpop



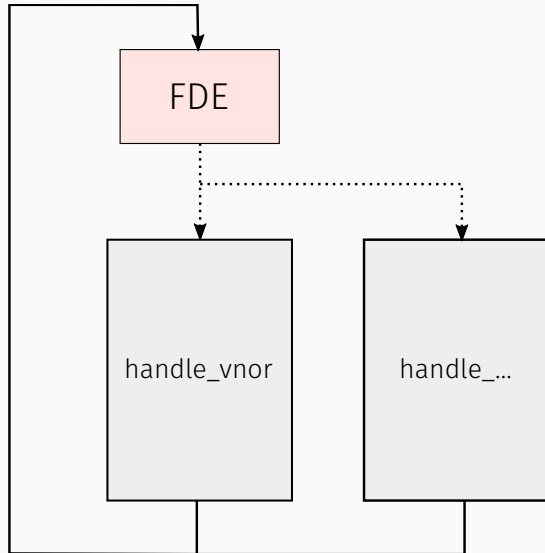
handle_vpush
handle_vadd
handle_vnor''
handle_vpop
handle_vadd'
handle_vnor
handle_vnor'
handle_vadd''

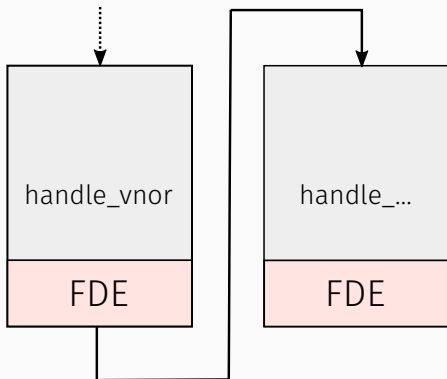
Hardening Technique #3 – No central VM dispatcher.

- A *central* VM dispatcher allows attacker to easily observe VM execution.
- **Idea:** Instead of branching to the central dispatcher, *inline* it into each handler.

Goal: No “single point of failure”.

(Themida, VMProtect Demo)





Threaded Code

James R. Bell
Digital Equipment Corporation

The concept of "threaded code" is presented as an alternative to machine language code. Hardware and software realizations of it are given. In software it is realized as interpretive code not needing an interpreter. Extensions and optimizations are mentioned.

Key Words and Phrases: interpreter, machine code, time tradeoff, space tradeoff, compiled code, subroutine calls, threaded code

CR Categories: 4.12, 4.13, 6.33

Fig. 2 Flow of control: interpretive code.

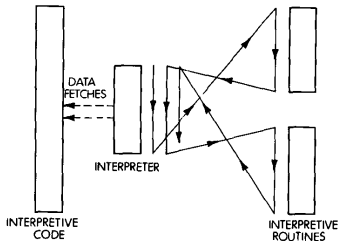
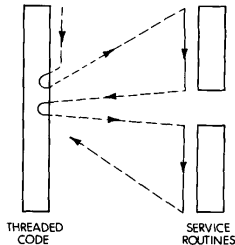


Fig. 3. Flow of control: threaded code.



Hardening Technique #4 – No explicit handler table.

- An *explicit* handler table easily reveals all VM handlers.

Hardening Technique #4 – No explicit handler table.

- An *explicit* handler table easily reveals all VM handlers.
- **Idea:** Instead of querying an explicit handler table, *encode* the next handler address in the VM instruction itself.

Goal: Hide location of handlers that have not been executed yet.

(VMProtect Full, SolidShield)

Hardening Technique #4 – No explicit handler table.

- An *explicit* handler table easily reveals all VM handlers.

- Idea



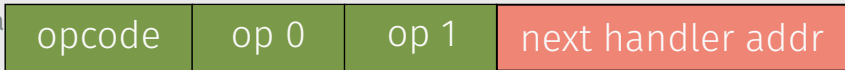
table,
the VM instruction itself.

Goal: Hide location of handlers that have not been executed yet.

(VMProtect Full, SolidShield)

Hardening Technique #4 – No explicit handler table.

- An *explicit* handler table easily reveals all VM handlers.
- Idea



Goal: Hide location of handlers that have not been executed yet.

(VMProtect Full, SolidShield)

SOFTWARE-PRACTICE AND EXPERIENCE, VOL. 11, 963-973 (1981)

Interpretation Techniques^{*}

PAUL KLINT

Mathematical Centre, P.O. Box 4079, 1009AB Amsterdam, The Netherlands

SUMMARY

The relative merits of implementing high level programming languages by means of interpretation or compilation are discussed. The properties and the applicability of interpretation techniques known as classical interpretation, **direct threaded code** and indirect threaded code are described and compared.

KEY WORDS

Interpretation versus compilation Interpretation techniques Instruction encoding Code generation Direct threaded code Indirect threaded code.

Hardening Technique #5 – Blinding VM bytecode.

- *Global analyses* on the bytecode possible, easy to patch instructions.

Hardening Technique #5 – Blinding VM bytecode.

- *Global analyses* on the bytecode possible, easy to patch instructions.
- **Idea:**
 - *Flow-sensitive* instruction decoding (“decryption” based on key register).
 - Custom decryption routine per handler, diversification.
 - Patching requires re-encryption of subsequent bytecode.

Goal: Hinder global analyses of bytecode and patching.

operand $\leftarrow [\mathbf{VIP} + 0]$

context $\leftarrow \text{semantics}(\text{context}, \text{operand})$

next_handler $\leftarrow [\mathbf{VIP} + 4]$

$\mathbf{VIP} \leftarrow \mathbf{VIP} + 8$

jmp *next_handler*

operand $\leftarrow [\mathbf{VIP} + 0]$

 *operand* $\leftarrow \text{unmangle}(\text{operand}, \mathbf{key})$

 *key* $\leftarrow \text{unmangle}'(\mathbf{key}, \text{operand})$

context $\leftarrow \text{semantics}(\text{context}, \text{operand})$

next_handler $\leftarrow [\mathbf{VIP} + 4]$

 *next_handler* $\leftarrow \text{unmangle}''(\text{next_handler}, \mathbf{key})$

 *key* $\leftarrow \text{unmangle}'''(\mathbf{key}, \text{next_handler})$

$\mathbf{VIP} \leftarrow \mathbf{VIP} + 8$

jmp *next_handler*

Code Obfuscation Techniques

Mixed Boolean-Arithmetic

What does this expression compute?

$$(x \oplus y) + 2 \cdot (x \wedge y)$$

What does this expression compute?

$$(x \oplus y) + 2 \cdot (x \wedge y)$$
$$= x + y$$

What does this expression compute?

$$(((x \oplus y) + ((x \wedge y) \ll 1)) \vee z) + (((x \oplus y) + ((x \wedge y) \ll 1)) \wedge z)$$

Mixed Boolean-Arithmetic

What does this expression compute?

$$\begin{aligned} &(((x \oplus y) + ((x \wedge y) \ll 1)) \vee z) + (((x \oplus y) + ((x \wedge y) \ll 1)) \wedge z) \\ &= x + y + z \end{aligned}$$

- Boolean identities?
- Arithmetic identities?
- Karnaugh-Veitch maps?

$$A \cdot 0 = 0$$

$$A + B = \overline{\overline{A} \cdot \overline{B}}$$

$$x^2 - y^2 = (x + y)(x - y)$$

		AB			
		00	01	11	10
CD	10	0	0	1	1
	11	0	0	1	1
	01	0	0	0	1
	00	0	1	1	1

Boolean-arithmetic algebra $BA[n]$

$(B^n, \wedge, \vee, \oplus, \neg, \leq, \geq, >, <, \leq^s, \geq^s, >^s, <^s, \neq, =, \gg^s, \gg, \ll, +, -, \cdot)$
is a Boolean-arithmetic algebra $BA[n]$, for $n > 0$, $B = \{0, 1\}$.

$BA[n]$ includes, amongst others, both:

- Boolean algebra $(B^n, \wedge, \vee, \neg)$,
- Integer modular ring $\mathbb{Z}/(2^n)$.

No techniques to simplify
such expressions easily!

Deobfuscation

Symbolic Execution

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
    and    rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```

Handler performing **nor**
(with flag side-effects)

Symbolic Execution

`__handle_vnor:`

- `mov rcx, [rbp]`
`mov rbx, [rbp + 4]`
`not rcx`
`not rbx`
`and rcx, rbx`
`mov [rbp + 4], rcx`
`pushf`
`pop [rbp]`
`jmp __vm_dispatcher`

`rcx ← [rbp]`

Handler performing **nor**
(with flag side-effects)

Symbolic Execution

```
__handle_vnor:  
    mov    rcx, [rbp]  
    • mov  rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
    and    rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```

rcx $\leftarrow$ [rbp]
rbx $\leftarrow$ [rbp + 4]

Handler performing **nor**
(with flag side-effects)

Symbolic Execution

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    • not   rcx  
      not   rbx  
    and    rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```

$rcx \leftarrow [rbp]$
 $rbx \leftarrow [rbp + 4]$
 $rcx \leftarrow \neg rcx = \neg [rbp]$

Handler performing **nor**
(with flag side-effects)

Symbolic Execution

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    • not    rbx  
    and    rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```

$rcx \leftarrow [rbp]$

$\textcolor{red}{rbx} \leftarrow [rbp + 4]$

$rcx \leftarrow \neg rcx = \neg [rbp]$

$rbx \leftarrow \neg \textcolor{red}{rbx} = \neg [rbp + 4]$

Handler performing **nor**
(with flag side-effects)

Symbolic Execution

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
    • and   rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```

```
rcx ← [rbp]  
rbx ← [rbp + 4]  
rcx ←  $\neg rcx = \neg [rbp]$   
rbx ←  $\neg rbx = \neg [rbp + 4]$   
rcx ←  $rcx \wedge rbx$   
      =  $(\neg [rbp]) \wedge (\neg [rbp + 4])$ 
```

Handler performing **nor**
(with flag side-effects)

Symbolic Execution

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
    • and   rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```

```
rcx ← [rbp]  
rbx ← [rbp + 4]  
rcx ←  $\neg$  rcx =  $\neg$  [rbp]  
rbx ←  $\neg$  rbx =  $\neg$  [rbp + 4]  
rcx ← rcx  $\wedge$  rbx  
      = ( $\neg$  [rbp])  $\wedge$  ( $\neg$  [rbp + 4])  
      = [rbp]  $\downarrow$  [rbp + 4]
```

Handler performing **nor**
(with flag side-effects)

Symbolic Execution

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
    and    rcx, rbx  
    • mov  [rbp + 4], rcx  
    pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```

Handler performing **nor**
(with flag side-effects)

```
rcx ← [rbp]  
rbx ← [rbp + 4]  
rcx ←  $\neg rcx = \neg [rbp]$   
rbx ←  $\neg rbx = \neg [rbp + 4]$   
rcx ←  $rcx \wedge rbx$   
           $= (\neg [rbp]) \wedge (\neg [rbp + 4])$   
           $= [rbp] \downarrow [rbp + 4]$   
[rbp + 4] ← rcx =  $[rbp] \downarrow [rbp + 4]$ 
```

Symbolic Execution

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
    and    rcx, rbx  
    mov    [rbp + 4], rcx  
• pushf  
    pop    [rbp]  
    jmp    __vm_dispatcher
```

```
rcx ← [rbp]  
rbx ← [rbp + 4]  
rcx ←  $\neg rcx = \neg [rbp]$   
rbx ←  $\neg rbx = \neg [rbp + 4]$   
rcx ←  $rcx \wedge rbx$   
       $= (\neg [rbp]) \wedge (\neg [rbp + 4])$   
       $= [rbp] \downarrow [rbp + 4]$   
[rbp + 4] ←  $rcx = [rbp] \downarrow [rbp + 4]$   
  
rsp ←  $rsp - 4$   
[rsp] ← flags
```

Handler performing **nor**
(with flag side-effects)

Symbolic Execution

```
__handle_vnor:  
    mov    rcx, [rbp]  
    mov    rbx, [rbp + 4]  
    not    rcx  
    not    rbx  
    and    rcx, rbx  
    mov    [rbp + 4], rcx  
    pushf  
    • pop   [rbp]  
    jmp    __vm_dispatcher
```

Handler performing **nor**
(with flag side-effects)

```
rcx ← [rbp]  
rbx ← [rbp + 4]  
rcx ←  $\neg rcx = \neg [rbp]$   
rbx ←  $\neg rbx = \neg [rbp + 4]$   
rcx ←  $rcx \wedge rbx$   
       $= (\neg [rbp]) \wedge (\neg [rbp + 4])$   
       $= [rbp] \downarrow [rbp + 4]$   
[rbp + 4] ←  $rcx = [rbp] \downarrow [rbp + 4]$   
  
rsp ←  $rsp - 4$   
[rsp] ← flags  
[rbp] ←  $[rsp] = flags$   
rsp ←  $rsp + 4$ 
```

Symbolic Execution

```
__handle_vnor:  
  mov    rcx, [rbp]  
  mov    rbx, [rbp + 4]  
  not    rcx  
  not    rbx  
  and    rcx, rbx  
  mov    [rbp + 4], rcx  
  pushf  
  pop    [rbp]  
  • jmp  __vm_dispatcher
```

Handler performing **nor**
(with flag side-effects)

```
rcx ← [rbp]  
rbx ← [rbp + 4]  
rcx ←  $\neg$  rcx =  $\neg$  [rbp]  
rbx ←  $\neg$  rbx =  $\neg$  [rbp + 4]
```

$[rbp + 4] \leftarrow ([rbp] \downarrow [rbp + 4])$

```
rcx ← rcx & rbx = [rbp] & [rbp + 4]  
[rbp + 4] ← rcx = [rbp] & [rbp + 4]
```

```
rsp ← rsp - 4  
[rsp] ← flags  
[rbp] ← [rsp] = flags  
rsp ← rsp + 4
```

Virtual Machine Handler

mov	eax, dword [rbp]	jmp	0xffffffffffff63380
mov	ecx, dword [rbp + 4]	dec	eax
cmp	r11w, r13w	stc	
sub	rbp, 4	ror	eax, 1
not	eax	jmp	0xffffffffffff2a70
clc		dec	eax
cmc		clc	
cmp	rdx, 0x28b105fa	bswap	eax
not	ecx	test	bp, 0x5124
cmp	r12b, r9b	neg	eax
cmc		test	dil, 0xe9
and	eax, ecx	cmp	bx, r14w
jmp	0xc239	cmc	
mov	word [rbp + 8], eax	push	rbx
pushfq		sub	bx, 0x49f8
movzx	eax, r10w	xor	dword [rsp], eax
and	ax, di	and	bh, 0xaf
pop	qword [rbp]	pop	rbx
sub	rsi, 4	movsxd	rax, eax
shld	rax, rdx, 0x1b	test	r13b, 0x94
xor	ah, 0x4d	add	rdi, rax
mov	eax, dword [rsi]	jmp	0xffffffffffffc67c7
cmp	ecx, r11d	lea	rax, [rsp + 0x140]
test	r10, 0x179708d5	cmp	rbp, rax
xor	eax, ebx	ja	0x6557b
		jmp	rdi

Virtual Machine Handler

[illegible]

[illegible]

Mixed Boolean-Arithmetic Expression

```
int mixed_boolean(int A, int B, int C) {
    int result;

    result = (((1438524315 + (((1438524315 + C) + 1438524315 * ((2956783114 - -1478456685 * C) |
    (-1478456685 * (1668620215 - A) - 2956783115)))) + A) - 1553572265)) + 1438524315 * ((2956783114 -
    -1478456685 * (((1438524315 + C) + 1438524315 * ((2956783114 - -1478456685 * C) | (-1478456685 *
    (1668620215 - A) - 2956783115)))) + A) - 1553572265)) | (-1478456685 * (1668620215 - B) -
    2956783115))) - ((1438524315 + (1668620215 - (((1438524315 + C) + 1438524315 * ((2956783114 -
    -1478456685 * C) | (-1478456685 * (1668620215 - A) - 2956783115)))) + A) - 1553572265))) +
    1438524315 * ((2956783114 - -1478456685 * (1668620215 - (((1438524315 + C) + 1438524315 *
    ((2956783114 - -1478456685 * C) | (-1478456685 * (1668620215 - A) - 2956783115)))) + A) -
    1553572265))) | (-1478456685 * B - 2956783115)))) + 1553572265;

    return -1478456685 * result - 2956783115;
}
```

Mixed Boolean-Arithmetic Expression

[illegible]

Mixed Boolean-Arithmetic Expression

Symbolic Execution

- ⊕ Captures full semantics of executed code
- ⊕ Computer algebra system, some degree of simplification
- ⊖ Usability decreases with increasing *syntactic* complexity
 - Artificial complexity (substitution, ...)
 - Algebraic complexity (MBA)

Symbolic Execution

- ⊕ Captures full semantics of executed code
- ⊕ Computer algebra system, some degree of simplification
- ⊖ Usability decreases with increasing *syntactic* complexity
 - Artificial complexity (substitution, ...)
 - Algebraic complexity (MBA)

What if we could reason about *semantics* only instead of *syntax*?

Program Synthesis

Program Synthesis: A Semantic Approach

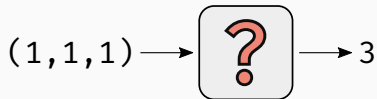
We use f as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$

Program Synthesis: A Semantic Approach

We use f as a black-box:

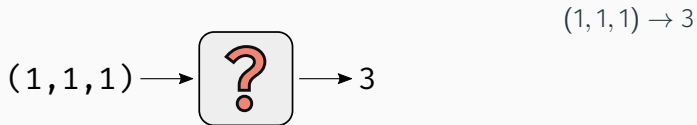
$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$



Program Synthesis: A Semantic Approach

We use f as a black-box:

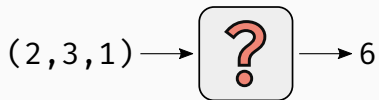
$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$



Program Synthesis: A Semantic Approach

We use f as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$

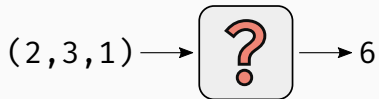


$$(1, 1, 1) \rightarrow 3$$

Program Synthesis: A Semantic Approach

We use f as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$



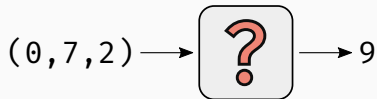
$$(1, 1, 1) \rightarrow 3$$

$$(2, 3, 1) \rightarrow 6$$

Program Synthesis: A Semantic Approach

We use f as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$



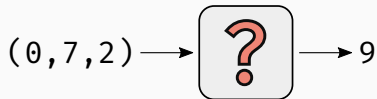
$$(1, 1, 1) \rightarrow 3$$

$$(2, 3, 1) \rightarrow 6$$

Program Synthesis: A Semantic Approach

We use f as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$



$$(1, 1, 1) \rightarrow 3$$

$$(2, 3, 1) \rightarrow 6$$

$$(0, 7, 2) \rightarrow 9$$

Program Synthesis: A Semantic Approach

We use f as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$

$$(1, 1, 1) \rightarrow 3$$

$$(2, 3, 1) \rightarrow 6$$

$$(0, 7, 2) \rightarrow 9$$

We **learn** a function that has the same I/O behavior:

Program Synthesis: A Semantic Approach

We use f as a black-box:

$$f(x, y, z) := (((x \oplus y) + ((x \wedge y) \cdot 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \cdot 2)) \wedge z)$$

$$(1, 1, 1) \rightarrow 3$$

$$(2, 3, 1) \rightarrow 6$$

$$(0, 7, 2) \rightarrow 9$$

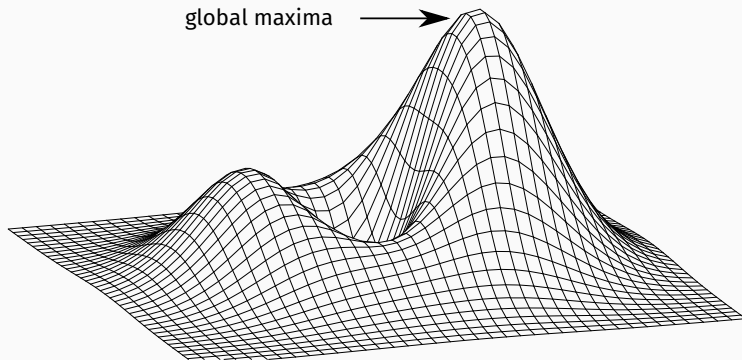
We **learn** a function that has the same I/O behavior:

$$h(x, y, z) := x + y + z$$

How to synthesize programs?

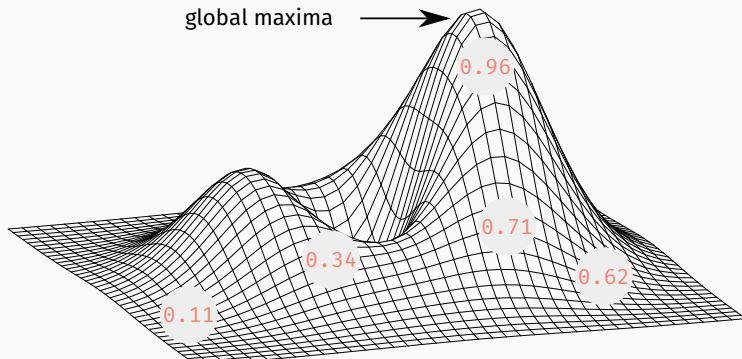
Stochastic Program Synthesis

- probabilistic optimization problem



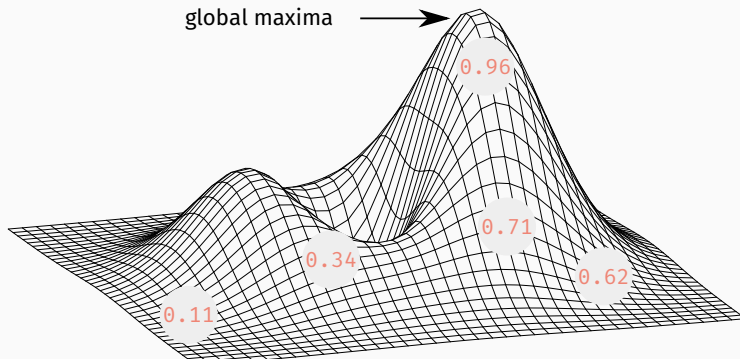
Stochastic Program Synthesis

- probabilistic optimization problem



Stochastic Program Synthesis

- probabilistic optimization problem
- based on Monte Carlo Tree Search (MCTS)



Let's synthesize: $a + b \bmod 8$

$$U \rightarrow U + U \mid U * U \mid a \mid b$$

$$U \rightarrow U + U \mid U * U \mid a \mid b$$

- non-terminal symbol: U

$$U \rightarrow U + U \mid U * U \mid a \mid b$$

- non-terminal symbol: U
- input variables: $\{a, b\}$

Program Generation

$$U \rightarrow U + U \mid U * U \mid a \mid b$$

- non-terminal symbol: U
- input variables: $\{a, b\}$
- candidate programs: $a, b, a * b, a + b, \dots$

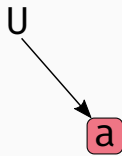
Program Generation

$$U \rightarrow U + U \mid U * U \mid a \mid b$$

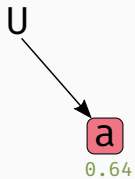
- non-terminal symbol: U
- input variables: $\{a, b\}$
- candidate programs: $a, b, a * b, a + b, \dots$
- intermediate programs: $U + U, U * U, U + b, \dots$



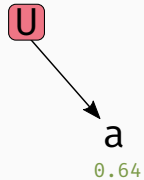
Monte Carlo Tree Search



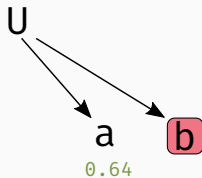
Monte Carlo Tree Search



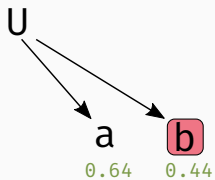
Monte Carlo Tree Search



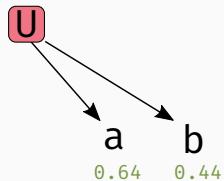
Monte Carlo Tree Search



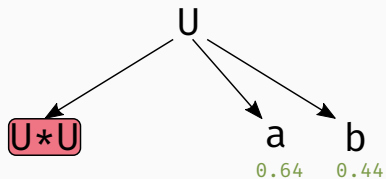
Monte Carlo Tree Search



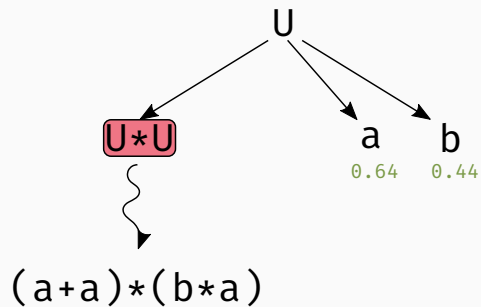
Monte Carlo Tree Search



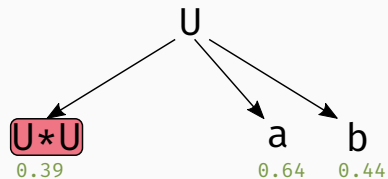
Monte Carlo Tree Search



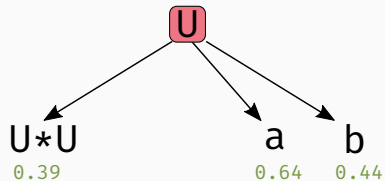
Monte Carlo Tree Search



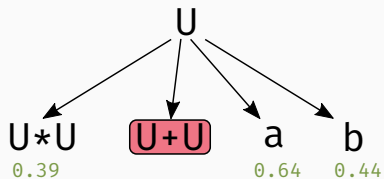
Monte Carlo Tree Search



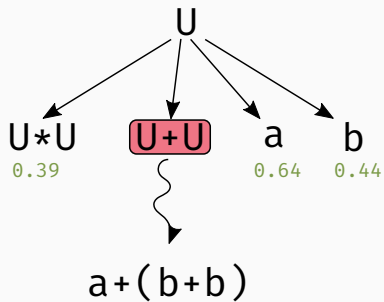
Monte Carlo Tree Search



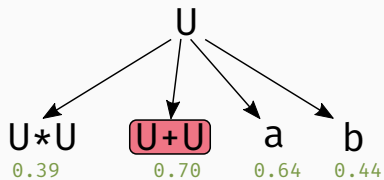
Monte Carlo Tree Search



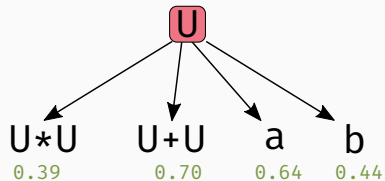
Monte Carlo Tree Search



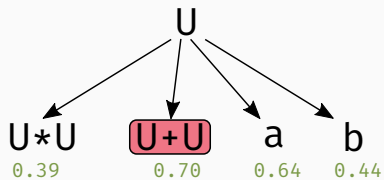
Monte Carlo Tree Search



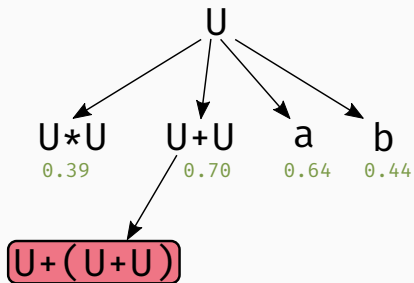
Monte Carlo Tree Search



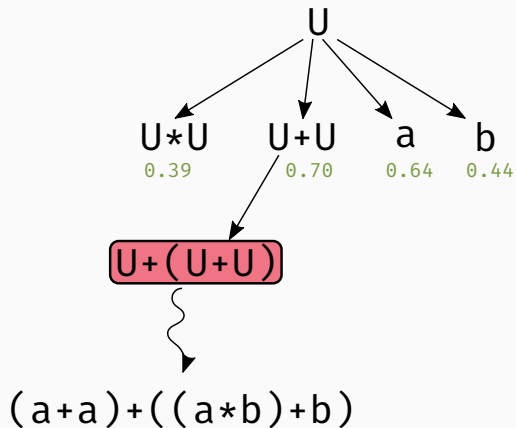
Monte Carlo Tree Search



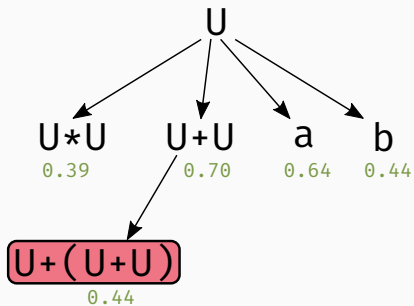
Monte Carlo Tree Search



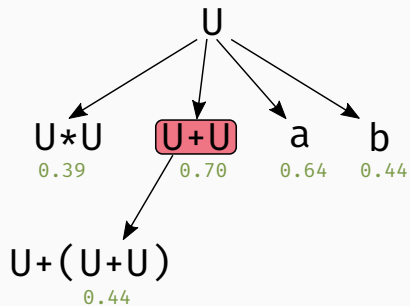
Monte Carlo Tree Search



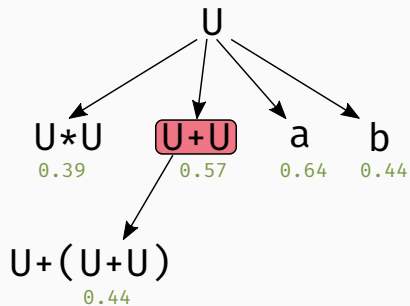
Monte Carlo Tree Search



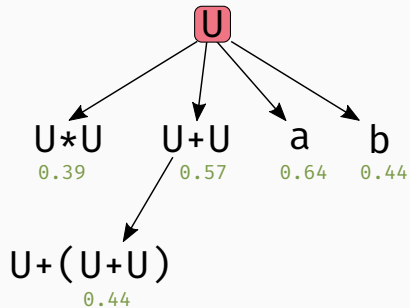
Monte Carlo Tree Search



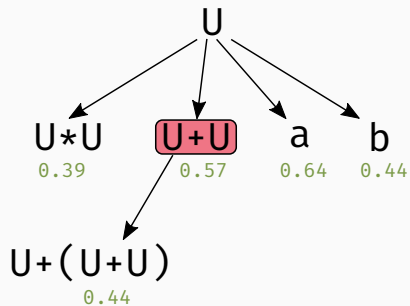
Monte Carlo Tree Search



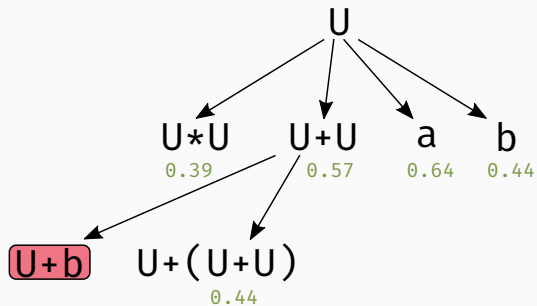
Monte Carlo Tree Search



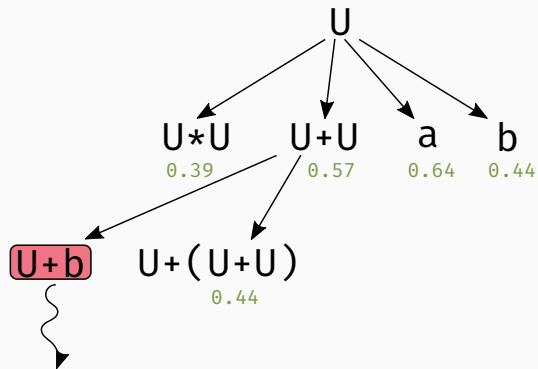
Monte Carlo Tree Search



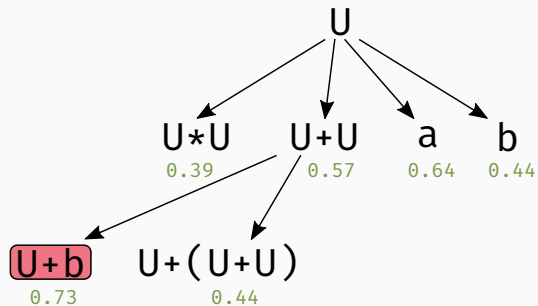
Monte Carlo Tree Search



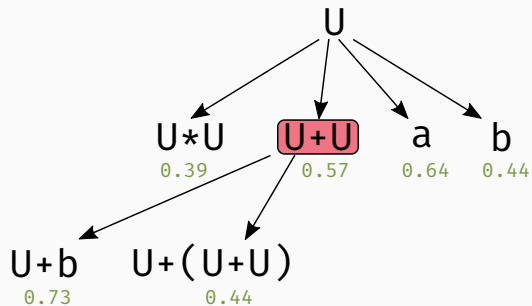
Monte Carlo Tree Search



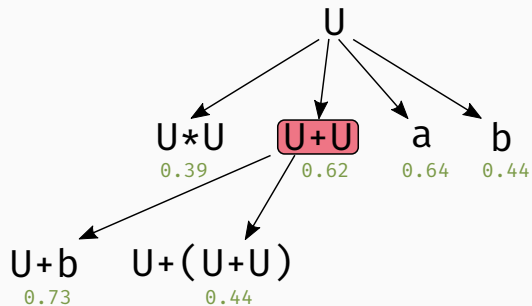
Monte Carlo Tree Search



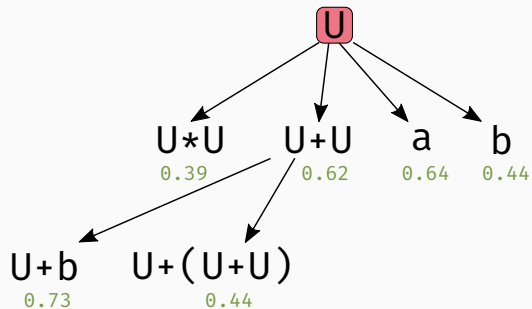
Monte Carlo Tree Search



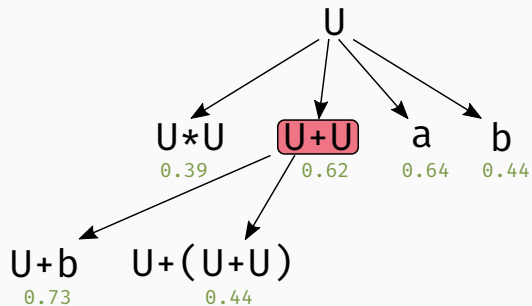
Monte Carlo Tree Search



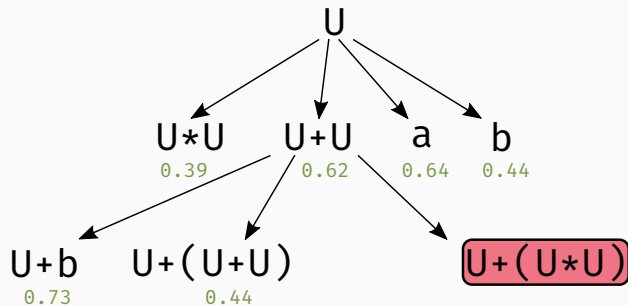
Monte Carlo Tree Search



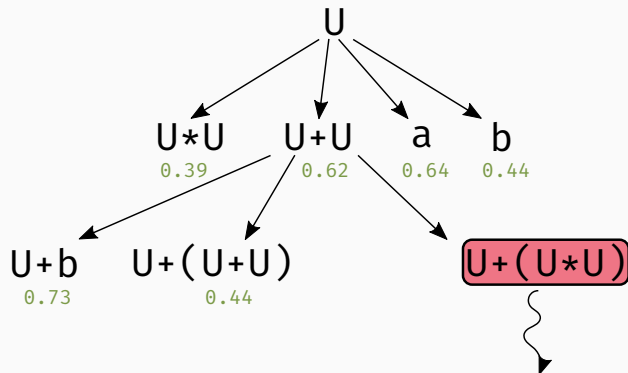
Monte Carlo Tree Search



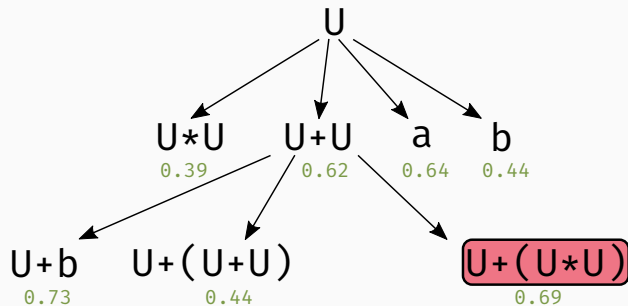
Monte Carlo Tree Search



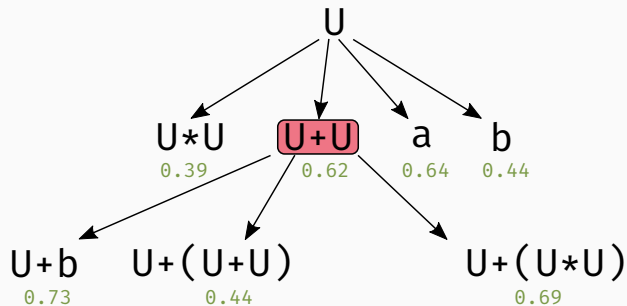
Monte Carlo Tree Search



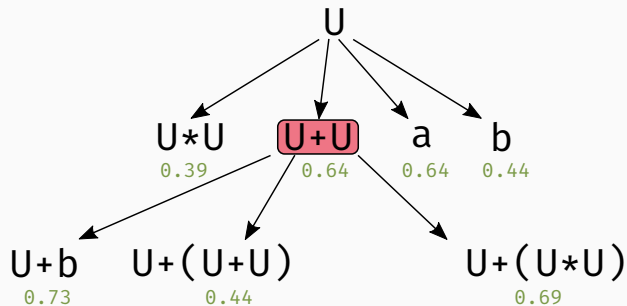
Monte Carlo Tree Search



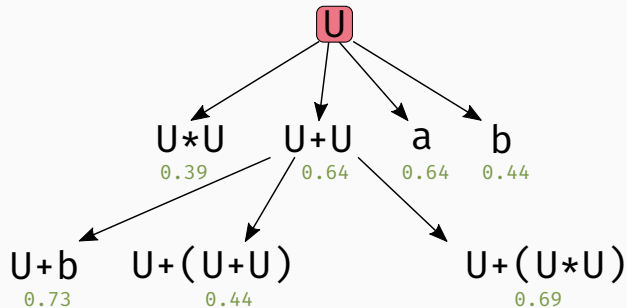
Monte Carlo Tree Search



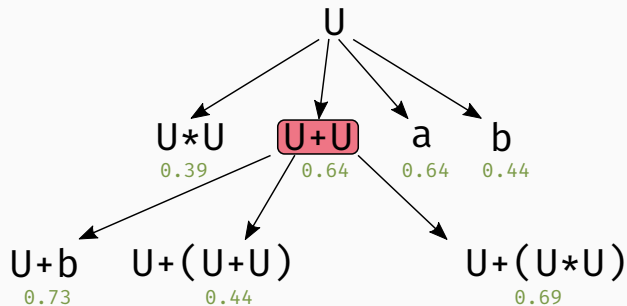
Monte Carlo Tree Search



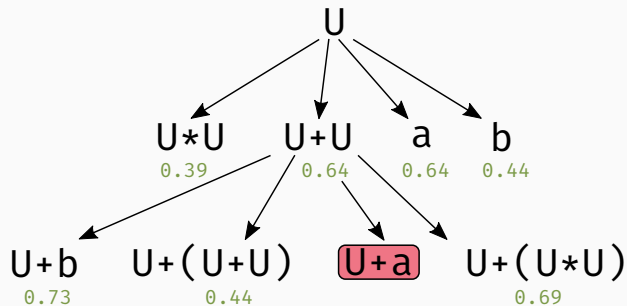
Monte Carlo Tree Search



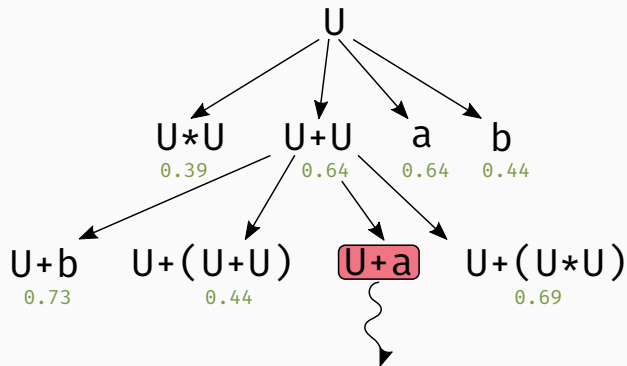
Monte Carlo Tree Search



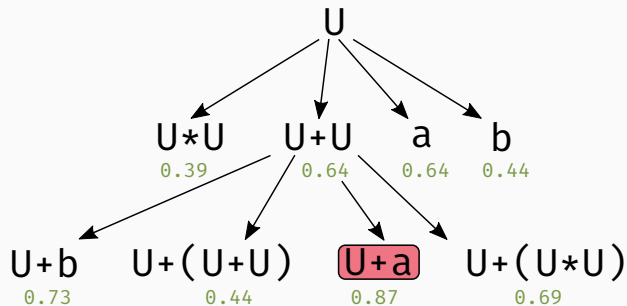
Monte Carlo Tree Search



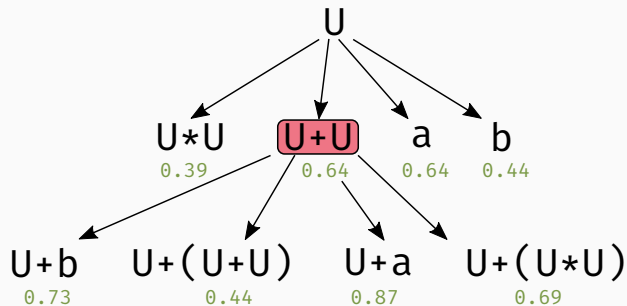
Monte Carlo Tree Search



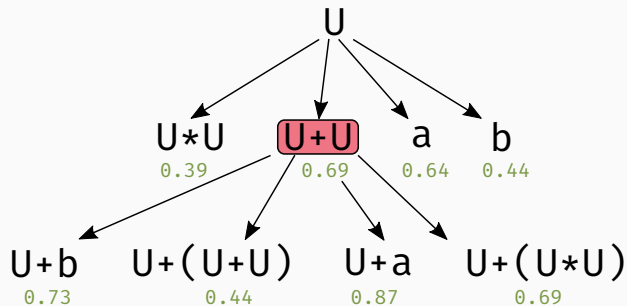
Monte Carlo Tree Search



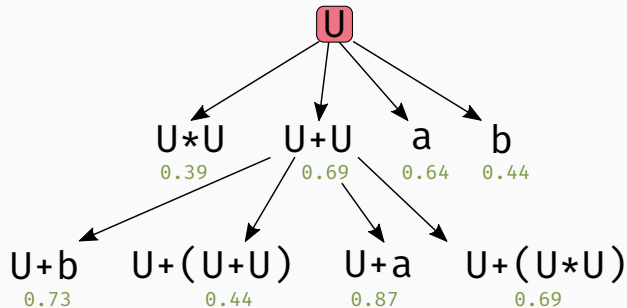
Monte Carlo Tree Search



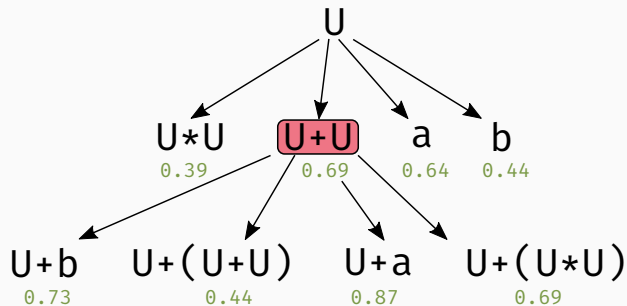
Monte Carlo Tree Search



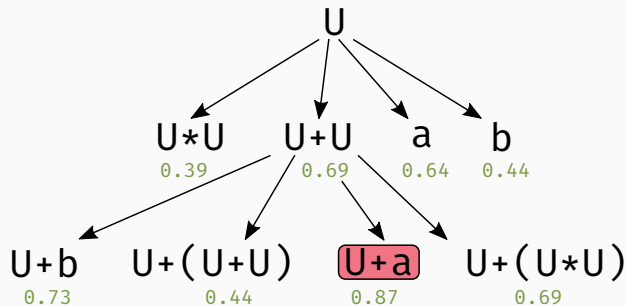
Monte Carlo Tree Search



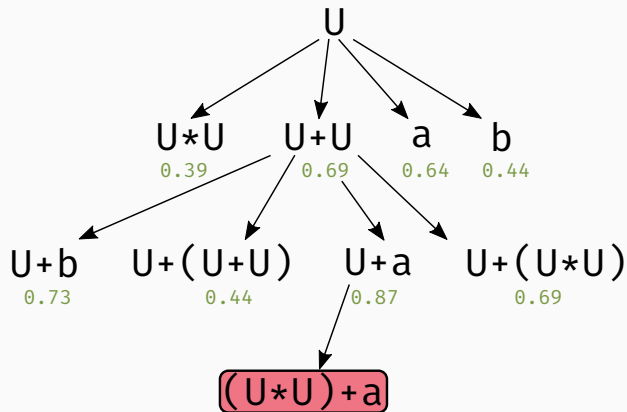
Monte Carlo Tree Search



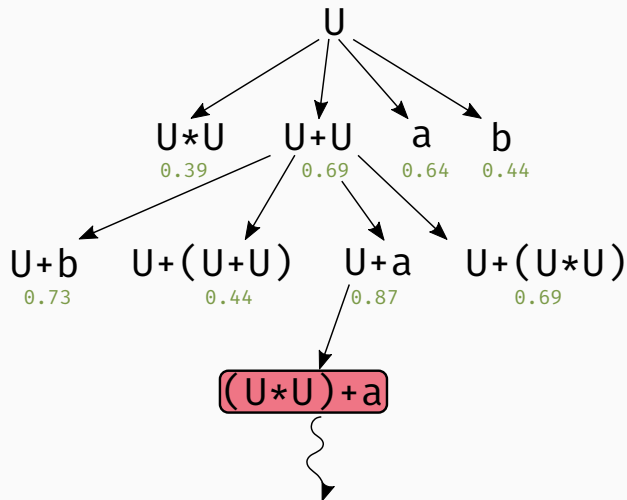
Monte Carlo Tree Search



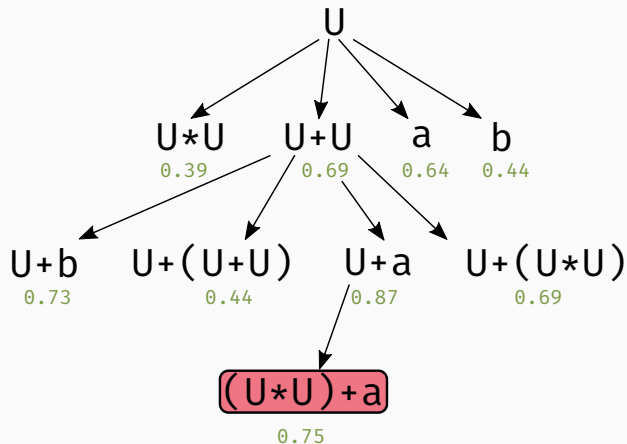
Monte Carlo Tree Search



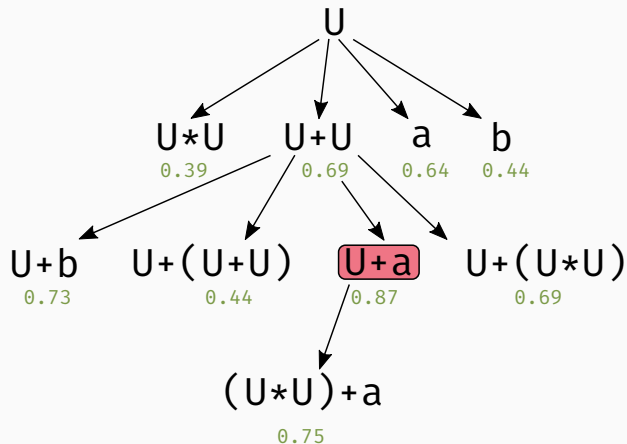
Monte Carlo Tree Search



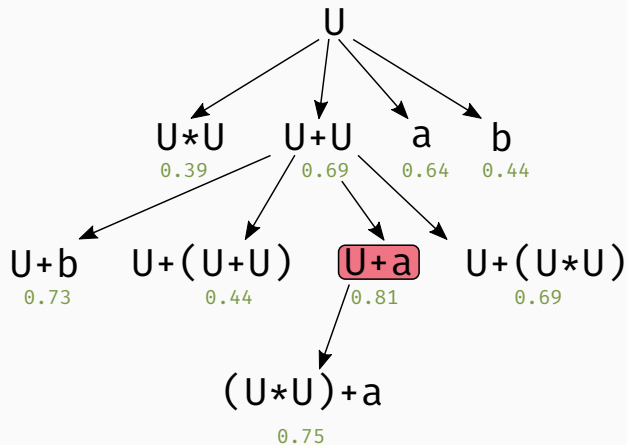
Monte Carlo Tree Search



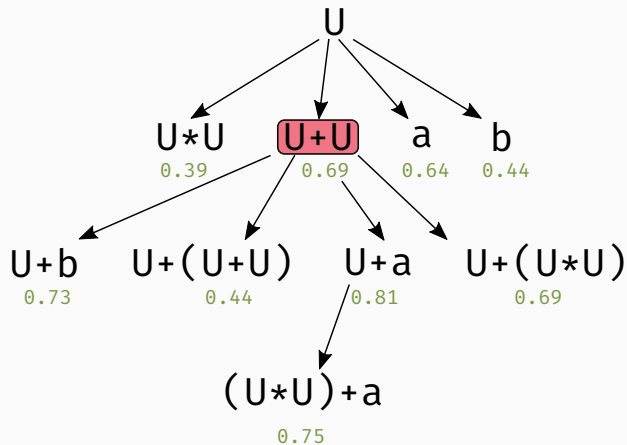
Monte Carlo Tree Search



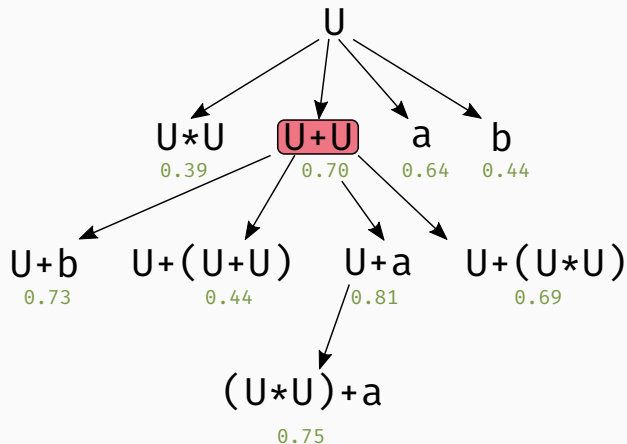
Monte Carlo Tree Search



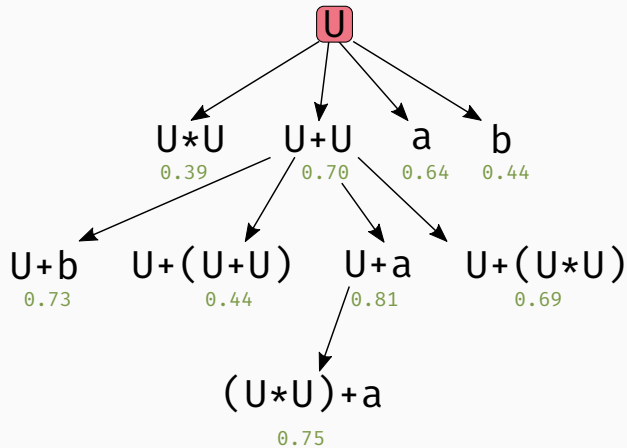
Monte Carlo Tree Search



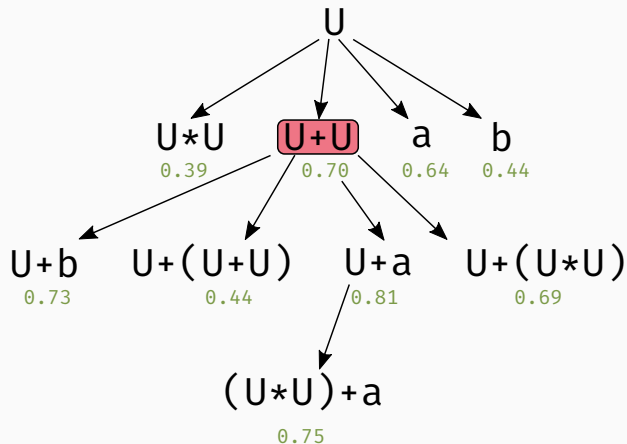
Monte Carlo Tree Search



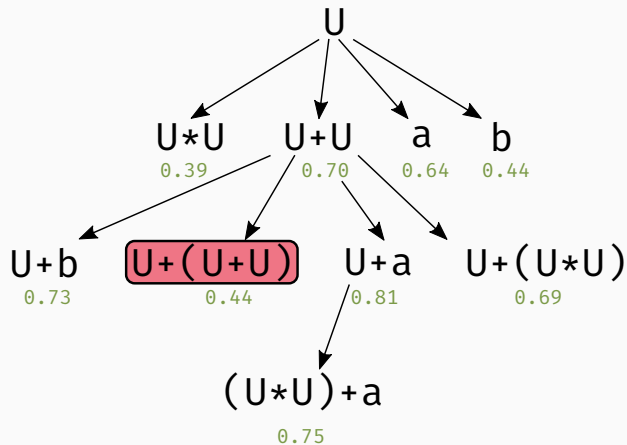
Monte Carlo Tree Search



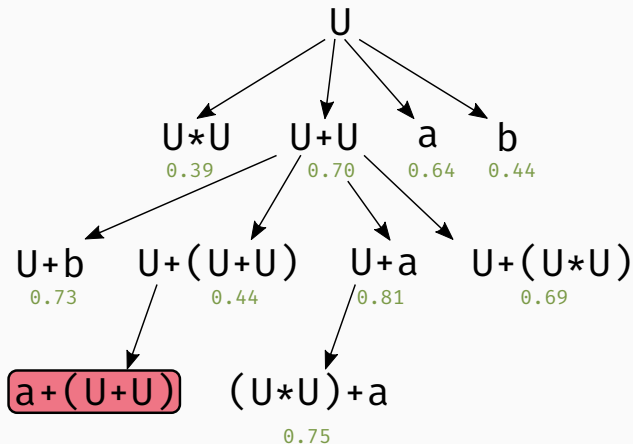
Monte Carlo Tree Search



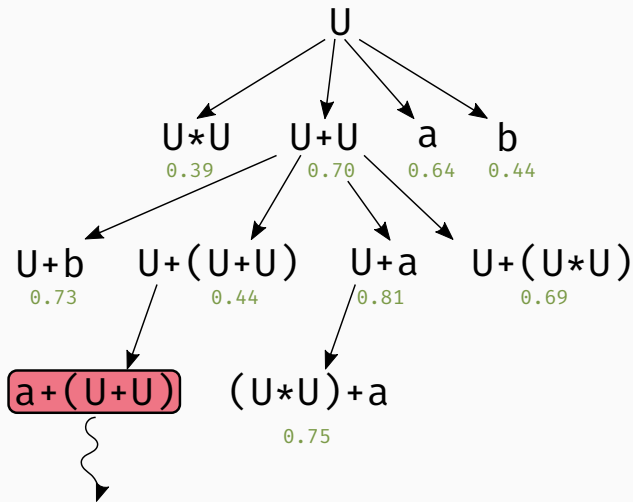
Monte Carlo Tree Search



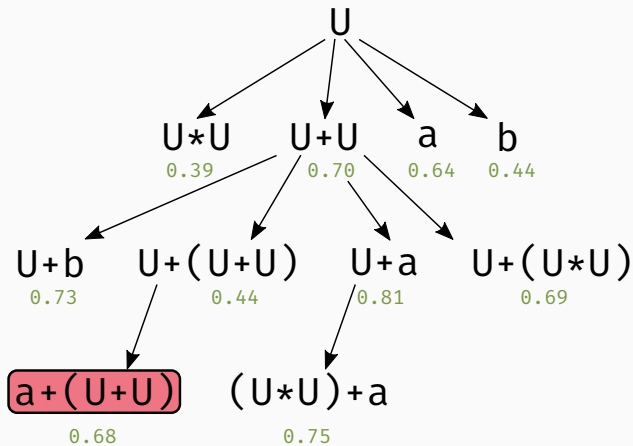
Monte Carlo Tree Search



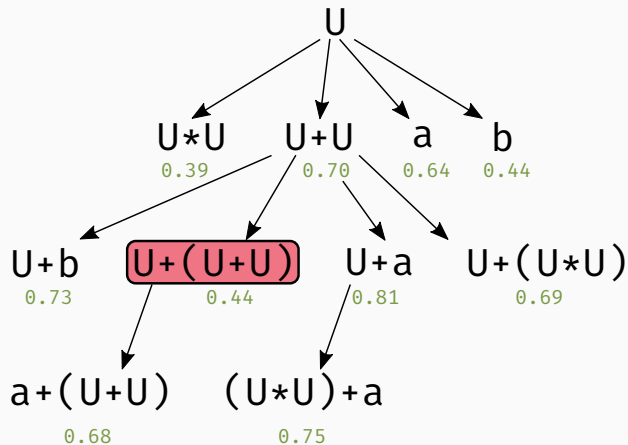
Monte Carlo Tree Search



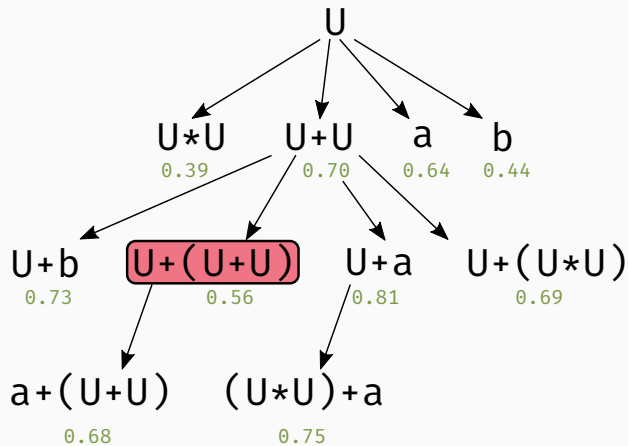
Monte Carlo Tree Search



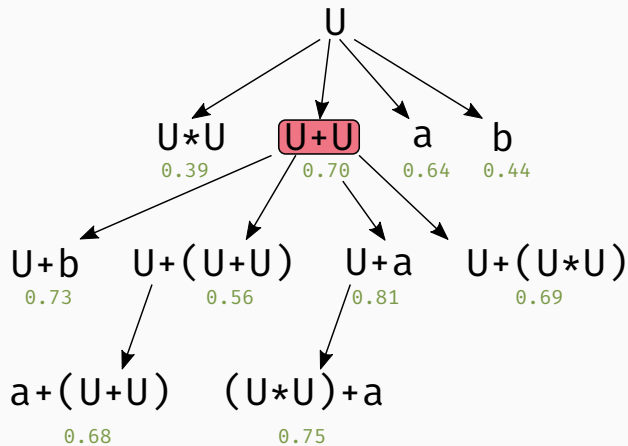
Monte Carlo Tree Search



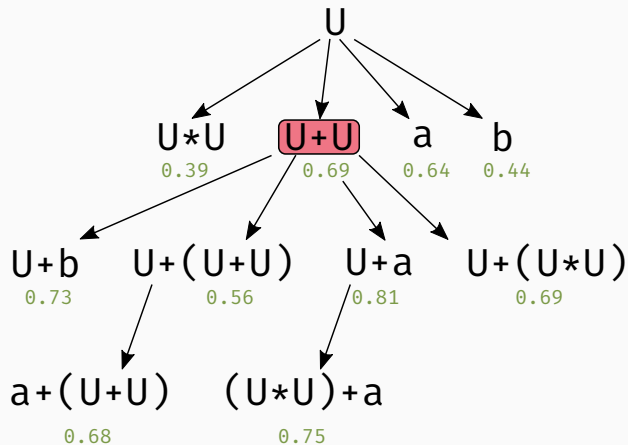
Monte Carlo Tree Search



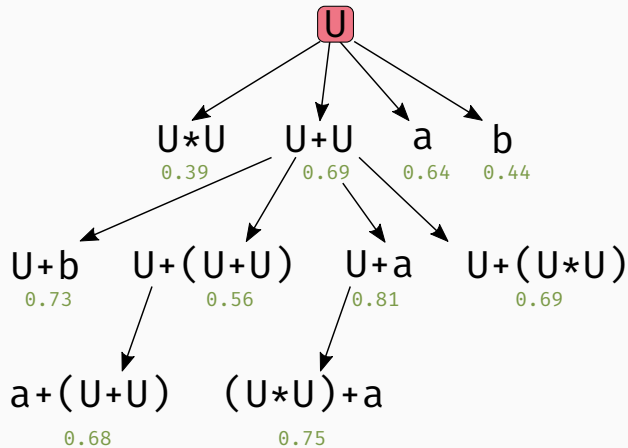
Monte Carlo Tree Search



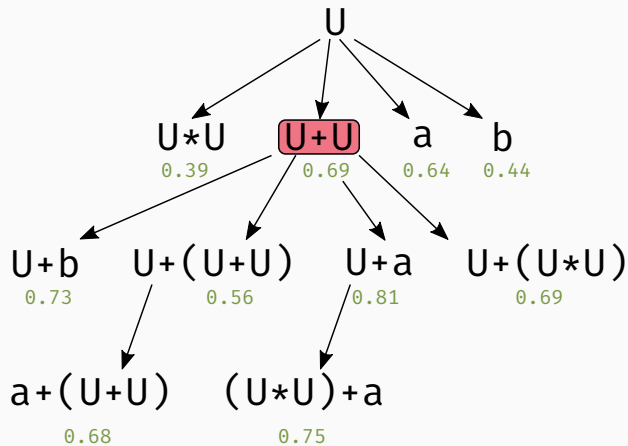
Monte Carlo Tree Search



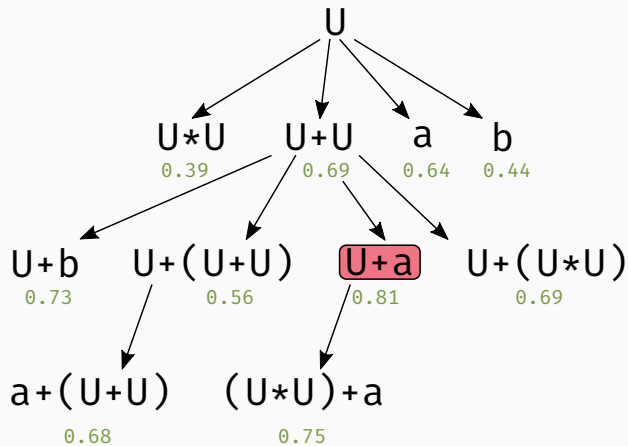
Monte Carlo Tree Search



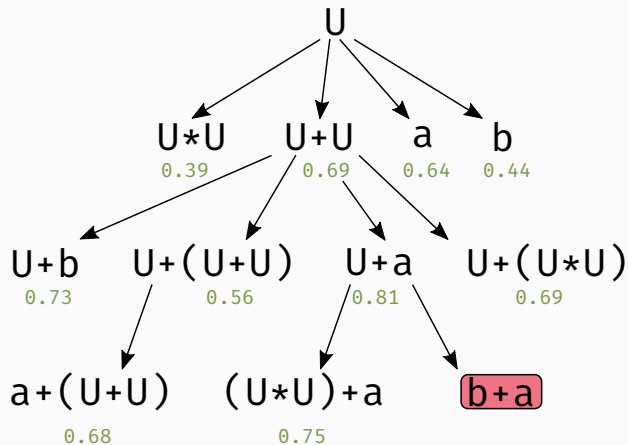
Monte Carlo Tree Search



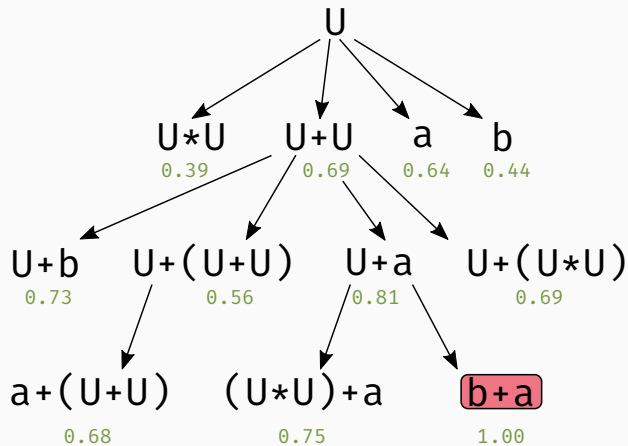
Monte Carlo Tree Search



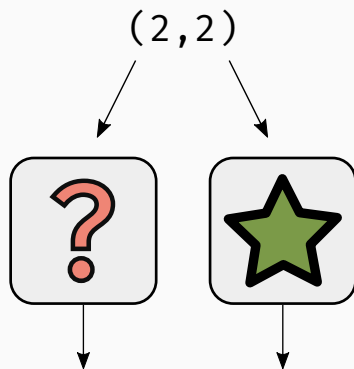
Monte Carlo Tree Search



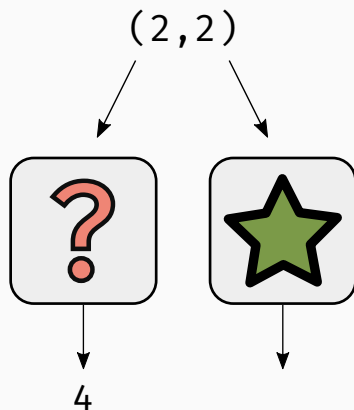
Monte Carlo Tree Search



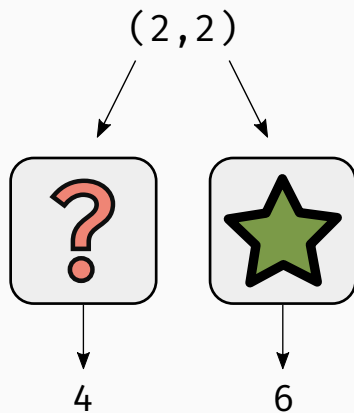
Score Calculation



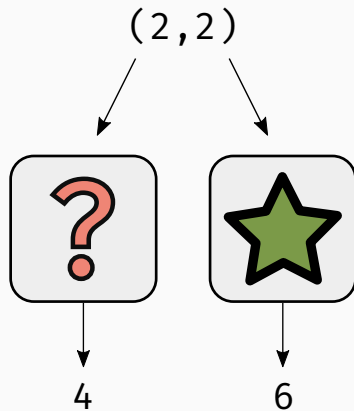
Score Calculation



Score Calculation

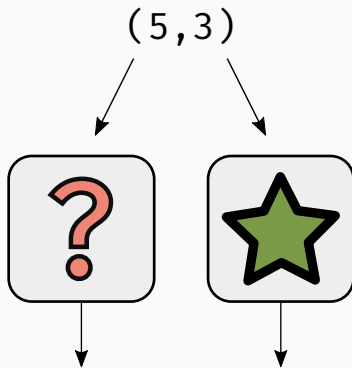


Score Calculation



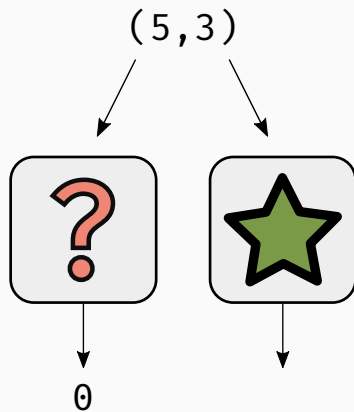
$$\text{similarity}(4, 6) = 0.78$$

Score Calculation



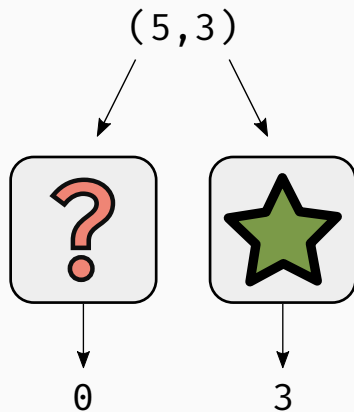
$$\text{similarity}(4, 6) = 0.78$$

Score Calculation



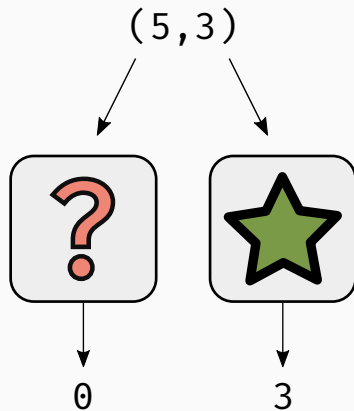
$$\text{similarity}(4, 6) = 0.78$$

Score Calculation



$$\text{similarity}(4, 6) = 0.78$$

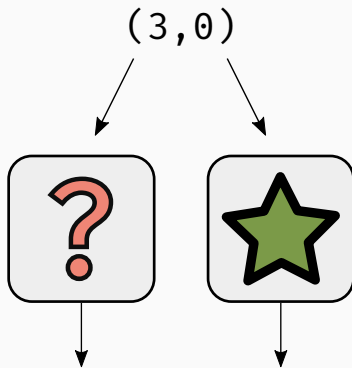
Score Calculation



$$\text{similarity}(4, 6) = 0.78$$

$$\text{similarity}(0, 3) = 0.33$$

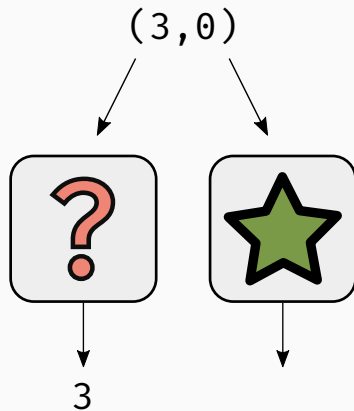
Score Calculation



$$\text{similarity}(4, 6) = 0.78$$

$$\text{similarity}(0, 3) = 0.33$$

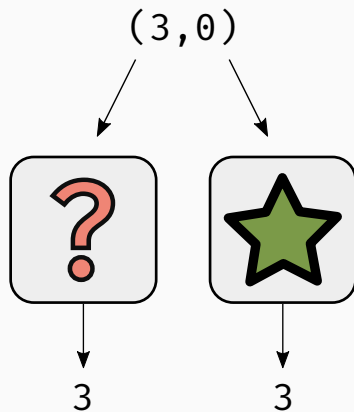
Score Calculation



$$\text{similarity}(4, 6) = 0.78$$

$$\text{similarity}(0, 3) = 0.33$$

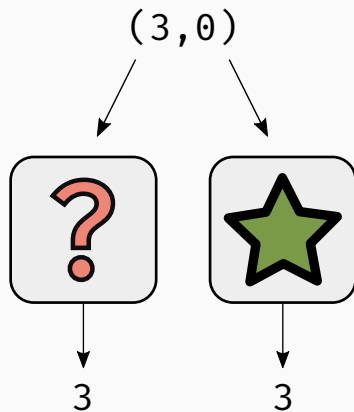
Score Calculation



$$\text{similarity}(4, 6) = 0.78$$

$$\text{similarity}(0, 3) = 0.33$$

Score Calculation

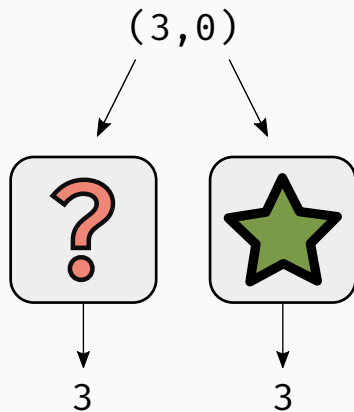


$$\text{similarity}(4, 6) = 0.78$$

$$\text{similarity}(0, 3) = 0.33$$

$$\text{similarity}(3, 3) = 1.0$$

Score Calculation



$$\text{similarity}(4, 6) = 0.78$$

$$\text{similarity}(0, 3) = 0.33$$

$$\text{similarity}(3, 3) = 1.0$$

average score: 0.70

111101111001000010001100100000000

111000100001100111101011000000000

Let's compare:

111	1011110010000100011001	00000000
111	0001000011001111010110	00000000

Are they in the same range?

Output Similarity: $\text{similarity}(O, O')$

1	1	1	1	0	1	1	1	1	0	0	1	0	0	0	0	1	0	0	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	1	0	0	0	0	1	1	0	0	1	1	1	1	0	1	0	1	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0

How many bits are different?

Output Similarity: $\text{similarity}(O, O')$

11110111100100001000110010000000
00010101011101101010000110000000
11100010000110011110101100000000

How close are they numerically?

DEMO

How to synthesize obfuscated code?

Obtaining Code



static disassembly

Obtaining Code



static disassembly

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

memory dump

Obtaining Code



static disassembly

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

memory dump

```
mov r15, 0x200      mov r15, rdx
xor r15, 0x800      xor r10d, dword ptr [r12]
mov rbx, rbp        sub r15, 0x800
add rbx, 0xc0       or r15, 0x400
mov rbx, qword ptr [rbx] mov rsi, 0x200
mov r13, 1         mov r14, rbp
mov rcx, 0         sub rsi, rsi
mov r15, rbp       mov rdi, rbp
add r15, 0xc0      mov r8, 0x400
or rcx, 0x88       sub rsi, r9
add rbx, 0xb       sub r8, rsi
mov r15, qword ptr [r15] add r14, 0
or r12, 0xffffffff80000000 add rsi, rax
sub rcx, 0x78      and r8, 0x88
movzx r10, word ptr [rbx] xor rsi, r14
xor r12, r13       mov rsi, rbp
add r12, 0xffff    add rdi, 0xc0
add r15, 0         sub r8, rdi
mov r8, rbp       add r8, 0x78
sub rcx, 0x10      add rsl, 4
or r12, r12       mov rcx, 0x200
or rcx, 0x800     mov rdi, qword ptr [rdi]
movzx r11, word ptr [r15] dword ptr [rsi], 0x254
xor rcx, 0x800    xor rcx, 0xf0
mov r12, r15      add rcx, r10
add r8, 0         add rdi, 6
xor r12, 0xf0     mov r8, 0x400
mov rbx, 0x58     mov ax, word ptr [rdi]
add r11, rbp      mov r8, 1
```

instruction trace

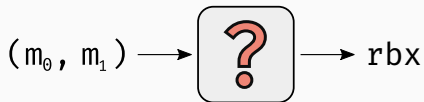
Learning Code Semantics

```
__handle_vnor:  
mov    rcx, [rbp]  
mov    rbx, [rbp + 4]  
not    rcx  
not    rbx  
and    rcx, rbx  
mov    [rbp + 4], rcx  
pushf  
pop    [rbp]  
jmp    __vm_dispatcher
```

Handler performing `nor`
(with flag side-effects)

Learning Code Semantics

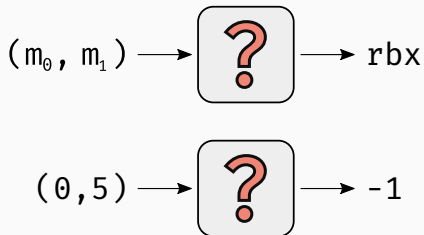
```
__handle_vnor:  
mov    rcx, [rbp]  
mov    rbx, [rbp + 4]  
not    rcx  
• not  rbx  
and    rcx, rbx  
mov    [rbp + 4], rcx  
pushf  
pop    [rbp]  
jmp    __vm_dispatcher
```



Handler performing `nor`
(with flag side-effects)

Learning Code Semantics

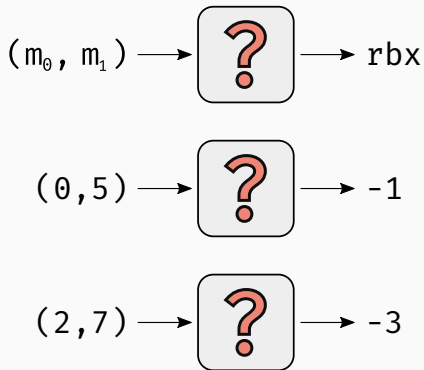
```
__handle_vnor:  
mov    rcx, [rbp]  
mov    rbx, [rbp + 4]  
not    rcx  
• not  rbx  
and    rcx, rbx  
mov    [rbp + 4], rcx  
pushf  
pop    [rbp]  
jmp    __vm_dispatcher
```



Handler performing `nor`
(with flag side-effects)

Learning Code Semantics

```
__handle_vnor:  
mov    rcx, [rbp]  
mov    rbx, [rbp + 4]  
not    rcx  
• not  rbx  
and    rcx, rbx  
mov    [rbp + 4], rcx  
pushf  
pop    [rbp]  
jmp    __vm_dispatcher
```

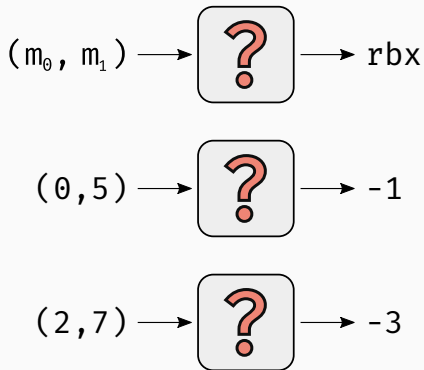


Handler performing `nor`
(with flag side-effects)

Learning Code Semantics

```
__handle_vnor:  
mov    rcx, [rbp]  
mov    rbx, [rbp + 4]  
not    rcx  
• not  rbx  
and    rcx, rbx  
mov    [rbp + 4], rcx  
pushf  
pop    [rbp]  
jmp    __vm_dispatcher
```

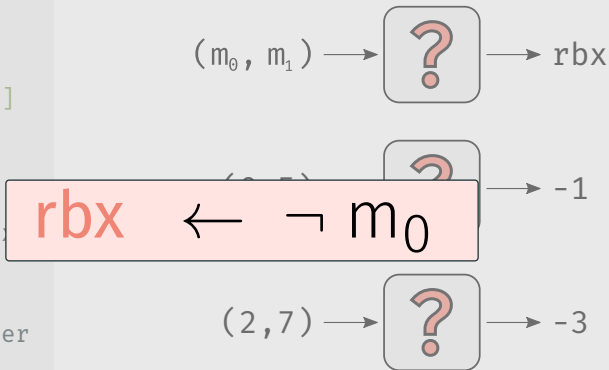
Handler performing `nor`
(with flag side-effects)



...

Learning Code Semantics

```
__handle_vnor:  
mov    rcx, [rbp]  
mov    rbx, [rbp + 4]  
not    rcx  
• not  rbx  
and    rcx, rbx  
mov    [rbp + 4], rcx  
pushf  
pop    [rbp]  
jmp    __vm_dispatcher
```

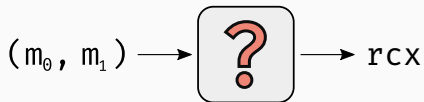


...

Handler performing `nor`
(with flag side-effects)

Learning Code Semantics

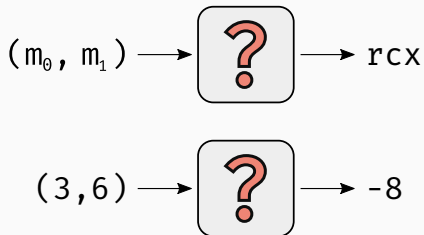
```
__handle_vnor:  
mov    rcx, [rbp]  
mov    rbx, [rbp + 4]  
not    rcx  
not    rbx  
• and   rcx, rbx  
mov    [rbp + 4], rcx  
pushf  
pop    [rbp]  
jmp    __vm_dispatcher
```



Handler performing `nor`
(with flag side-effects)

Learning Code Semantics

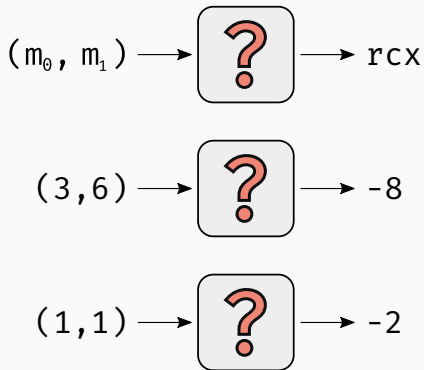
```
__handle_vnor:  
mov    rcx, [rbp]  
mov    rbx, [rbp + 4]  
not    rcx  
not    rbx  
• and   rcx, rbx  
mov    [rbp + 4], rcx  
pushf  
pop    [rbp]  
jmp    __vm_dispatcher
```



Handler performing `nor`
(with flag side-effects)

Learning Code Semantics

```
__handle_vnor:  
mov    rcx, [rbp]  
mov    rbx, [rbp + 4]  
not    rcx  
not    rbx  
• and   rcx, rbx  
mov    [rbp + 4], rcx  
pushf  
pop    [rbp]  
jmp    __vm_dispatcher
```

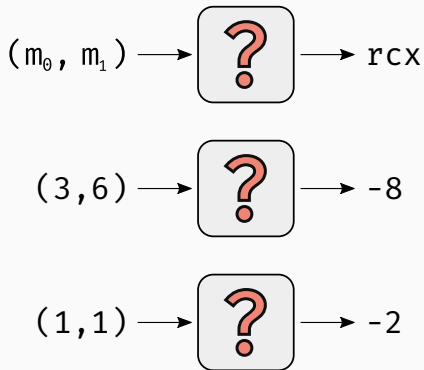


Handler performing `nor`
(with flag side-effects)

Learning Code Semantics

```
__handle_vnor:  
mov    rcx, [rbp]  
mov    rbx, [rbp + 4]  
not    rcx  
not    rbx  
• and   rcx, rbx  
mov    [rbp + 4], rcx  
pushf  
pop    [rbp]  
jmp    __vm_dispatcher
```

Handler performing `nor`
(with flag side-effects)



...

Learning Code Semantics

```
__handle_vnor:  
mov    rcx, [rbp]  
mov    rbx, [rbp + 4]  
not    rcx  
not    rbx  
• and   rcx, rbx  
mov    [rbp + 8], rcx  
pushf  
pop    [rbp]  
jmp    __vm_dispatcher
```

rcx $\leftarrow \neg (m_0 \vee m_1)$

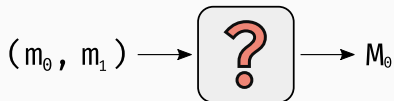


...

Handler performing `nor`
(with flag side-effects)

Learning Code Semantics

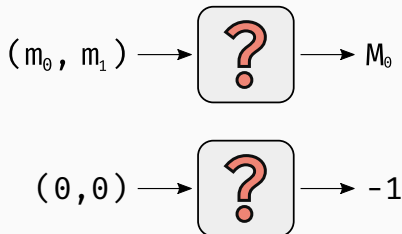
```
__handle_vnor:  
mov    rcx, [rbp]  
mov    rbx, [rbp + 4]  
not    rcx  
not    rbx  
and    rcx, rbx  
• mov  [rbp + 4], rcx  
pushf  
pop    [rbp]  
jmp    __vm_dispatcher
```



Handler performing **nor**
(with flag side-effects)

Learning Code Semantics

```
__handle_vnor:  
mov    rcx, [rbp]  
mov    rbx, [rbp + 4]  
not    rcx  
not    rbx  
and    rcx, rbx  
• mov  [rbp + 4], rcx  
pushf  
pop    [rbp]  
jmp    __vm_dispatcher
```

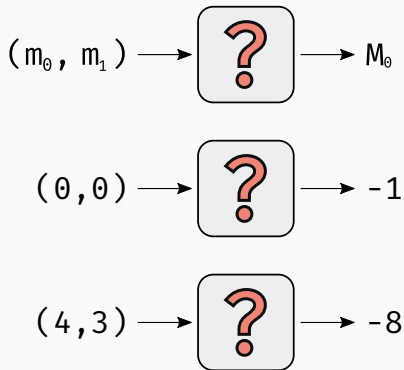


Handler performing `nor`
(with flag side-effects)

Learning Code Semantics

```
__handle_vnor:  
mov    rcx, [rbp]  
mov    rbx, [rbp + 4]  
not    rcx  
not    rbx  
and    rcx, rbx  
• mov  [rbp + 4], rcx  
pushf  
pop    [rbp]  
jmp    __vm_dispatcher
```

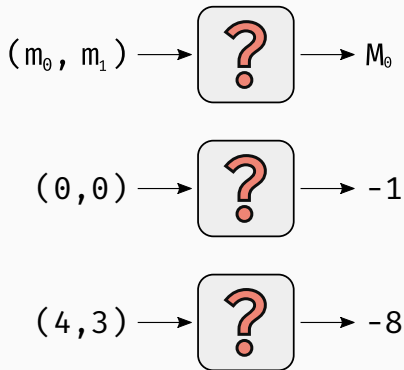
Handler performing `nor`
(with flag side-effects)



Learning Code Semantics

```
__handle_vnor:  
mov    rcx, [rbp]  
mov    rbx, [rbp + 4]  
not    rcx  
not    rbx  
and    rcx, rbx  
• mov  [rbp + 4], rcx  
pushf  
pop    [rbp]  
jmp    __vm_dispatcher
```

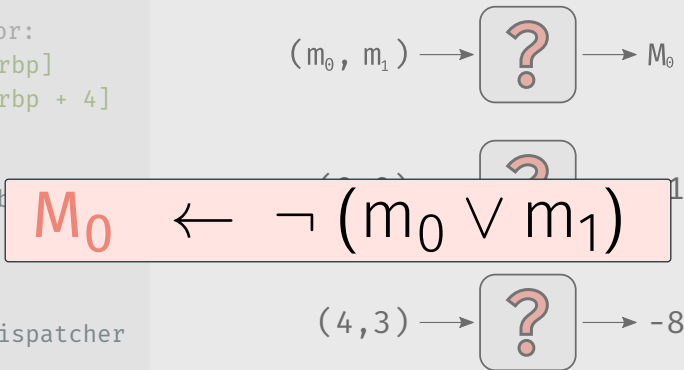
Handler performing `nor`
(with flag side-effects)



...

Learning Code Semantics

```
__handle_vnor:  
mov    rcx, [rbp]  
mov    rbx, [rbp + 4]  
not    rcx  
not    rbx  
and    rcx, rbx  
• mov  [rbp + 8], rcx  
pushf  
pop    [rbp]  
jmp    __vm_dispatcher
```



Handler performing `nor`
(with flag side-effects)

Learning Code Semantics

```
__handle_vnor:  
mov    rcx, [rbp]  
mov    rbx, [rbp + 4]  
not    rcx  
• not  rbx  
• and  rcx, rbx  
• mov  [rbp + 4], rcx  
pushf  
pop    [rbp]  
jmp    __vm_dispatcher
```

Handler performing **nor**
(with flag side-effects)

$rbx \leftarrow \neg m_0$

$rcx \leftarrow \neg (m_0 \vee m_1)$

$M_0 \leftarrow \neg (m_0 \vee m_1)$

WinDbg



Valgrind



Unicorn

<your tool here>

x64dbg

DynamoRIO

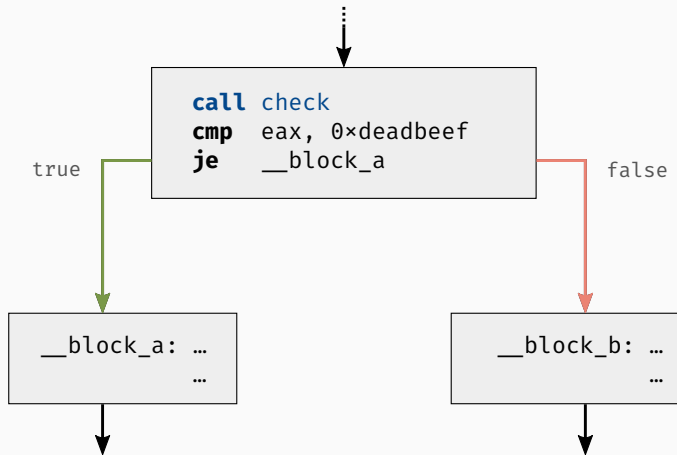
Miasm



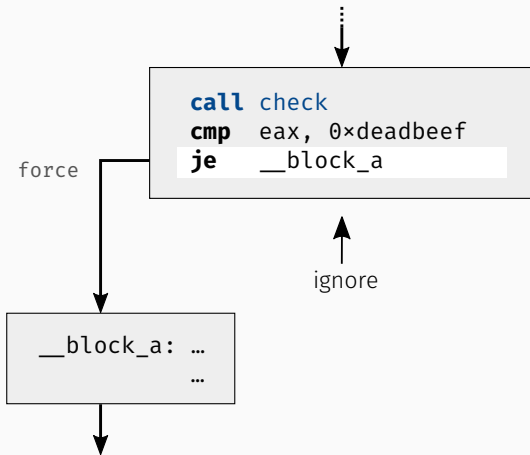
angr

Metasm

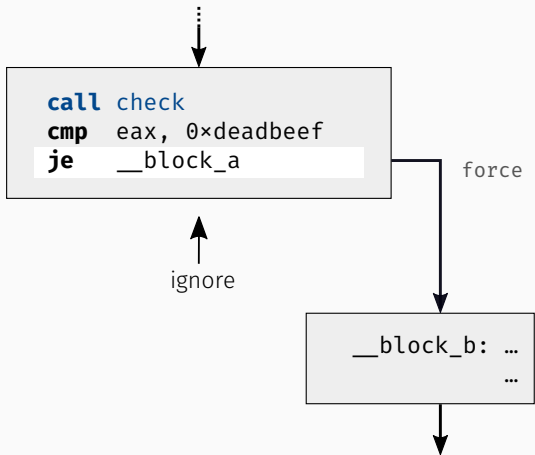
Instruction Trace: Forced Execution



Instruction Trace: Forced Execution



Instruction Trace: Forced Execution



- program synthesis framework for code deobfuscation
- written in Python
- random I/O sampling for assembly code
- MCTS-based program synthesis

<https://github.com/RUB-SysSec/syntia>

DEMO

Breaking Virtual Machine Obfuscation

Reminder: Virtual Machine Hardening

Hardening Technique #1 – Obfuscating individual VM components.

Hardening Technique #2 – Duplicating VM handlers.

Hardening Technique #3 – No central VM dispatcher.

Hardening Technique #4 – No explicit handler table.

Hardening Technique #5 – Blinding VM bytecode.

#1: Obfuscating Individual VM Components

```
mov r15, 0x200
xor r15, 0x800
mov rbp, sub
add rbx, 0xc0
mov rbx, qword ptr [rbx]
mov r13, 1
mov rcx, 0
mov r15, rbp
add r15, 0xc0
or rcx, 0x88
or add rbx, 0xb
mov r15, qword ptr [r15]
or r12, 0xffffffff80000000
sub rcx, 0x78
movzx r10, word ptr [rbx]
xor r12, r13
add r12, 0xffff
add r15, 0
mov r8, rbp
sub rcx, 0x10
or r12, r12
or rcx, 0x800
movzx r11, word ptr [r15]
xor rcx, 0x800
mov r12, r15
add r8, 0
xor r12, 0xf0
mov rbx, 0x58
add r11, rbp
xor rbx, 0x800
and r12, 0x20
add rbx, 0x800
mov r11, qword ptr [r11]
add rbx, 1
and r12, r9
mov rdx, 1
xor r10d, dword ptr [r8]
sub r9, r11
pushfq
xor rbx, 0xf0
xor rbx, 0x800
and rdx, r8
mov r12, rbp
xor rdx, 0x20
sub rbx, 4
add r11, 0x2549b044
or rbx, 0x78
and rdx, r10
mov rax, 0
add r12, 0x42

mov r15, rdx
xor r15d, dword ptr [r12]
sub r15, 0x800
or rdx, 0x400
mov rsi, 0x200
mov r14, rbp
mov rsi, rsi
mov r14, rbp
mov rsi, rsi
mov r15, rbp
mov r15, 0xc0
mov rcx, qword ptr [rcx]
add rsi, r9
movzx r10, word ptr [rcx]
mov r9, rbp
add r9, 0
xor r10d, dword ptr [r9]
and rdi, 0xffffffff80000000
sub r13, 0xf0
mov rsi, 0
sub r13, 0x20
mov rbx, rbp
or r13, 0x88
and rcx, 8
mov r8, 0x58
mov rbx, 0xc0
mov rcx, qword ptr [rbx]
sub rcx, 0x20
add rdi, 0x80
sub r13, 0x10
add rbx, 8
mov si, word ptr [rbx]
or r9, 0xffff
sub r9, 1
mov r9, rbp
add r13, 0x58
add r9, 0
sub r13, 0x80
mov r15, r13
or rcx, r12
xor esi, dword ptr [r9]
mov r10, rbp
mov r10d, 0xcc
sub r8, 0x20
xor esi, dword ptr [r10]
xor r13, 0x90
add rdi, 0x10
mov r14, rsi
mov rdx, rbp
add rdx, 0
add dword ptr [rdx], esi
xor r12, 1
mov r13, r15

or r14, r14
mov rax, rbp
and rcx, r13
add rcx, 4
sub r8, 0x80000000
add r13, 0xffff
and rcx, 0x20
mov r10, rbp
add r13, r15
add r10, 0x89
word ptr [r10], si
xor rdx, r11
mov rsi, rbp
sub rdx, rbx
and rax, 0x40
or rbx, 0xf0
add rsi, 0x5a
mov r8, rcx
movzx rsi, word ptr [rsi]
mov rax, 0x200
mov r14, rbp
and rax, rdx
and rcx, 0x20
mov r14, 0x89
or rax, 0x40
xor si, 0x7a28
add rdx, 0x78
add rdx, 0x20
movzx r14, word ptr [r14]
mov r14, 0x58
mov rsi, rbp
xor rax, rdx
add r8, 0x80
mov r15, rsi
add r14, rbp
add r8, r15
mov rbx, 0
and rdx, 0x10
add qword ptr [r14], qword ptr [rsi], r14
pushfq
xor r11, r14
add r15, r14
mov r13, 0x12
mov r8, 0
mov r14, 0x88
and r13, 0x40
add r13, 1
mov rdx, rbp

mov r14, 0x200
add rdx, 0xc0
r11, r14
or r15, 0x88
or rdx, qword ptr [rdx]
add rdx, 0xa
add r11, 0x78
mov r8b, byte ptr [rdx]
cnp r8b, 0
add sub
mov rdx, rbp
or r11, 0x40
and r15, 1
xor r11, 0x10
xor rdx, 0xc0
or r14, 4
mov r15, 0x12
mov rdx, qword ptr [rdx]
sub r11, r8
add rdx, 4
or r11, 0x80
mov r8w, word ptr [rdx]
mov r14, r8
add r8, rbp
xor r13, 4
pop r10
mov qword ptr [r8], r10
jmp 0x4ae
xor rsi, 0x88
xor rbx, 0xffffffff80000000
add rsi, 0x78
mov r10b, 0x68
mov r9, 0x12
or rbx, r10
and r15, 0x78
mov r14, rbp
or r9, 8
add r14, 0x29
xor rbx, rdi
xor r15, 0x3f
or byte ptr [r14], r10b
mov rax, 0x58
mov r8, rbp
sub rsi, 0x78
add r8, 0x127
mov rdi, rbx
xor rdx, 0x88
mov r8, qword ptr [r8]
xor rsi, 1
mov rax, rbp

add r15, 0x3f
or r15, 0xffffffff80000000
add rsi, r9
add rax, 0xc0
add rdi, r14
or rsi, 1
mov rax, qword ptr [rax]
and rdi, 0x7fffffff
add rax, 2
add rsi, 4
or rbx, rsi
movzx rax, word ptr [rax]
mov r9, rbp
mov r13, 0x200
mov r10, 0x58
add r9, 0
or r10, 0x20
add eax, dword ptr [r9]
xor r10, 0x40
add eax, 0x3f505c07
add r15, 0x88
mov r12, rbp
or rdi, 0x90
add r12, 0
or rbx, 0x80
add rdi, 0xf0
mov r13, 0x400
add dword ptr [r12], eax
and rsi, r8
or r10, 8
and rbx, 0x20
and rax, 0xffff
mov r11, 0
add r13, r8
or rbx, 1
shl rax, 3
add r8, rax
or rbx, r15
sub r15, 0x10
xor r11, r13
mov rbx, qword ptr [r8]
mov rdx, rbp
sub r13, 0x80
add rdx, 0xc0
add qword ptr [rdx], 0xd
jmp rbx
```

#1: Obfuscating Individual VM Components

```

mov r15, 0x200
xor r15, 0x800
mov rbp, sub
add rbx, 0xc0
mov rbx, qword ptr [rbx]
mov r13, 1
mov rcx, 0
mov r15, rbp
add r15, 0xc0
or rcx, 0x88
or add rbx, 0xb
mov r15, qword ptr [r15]
or r12, 0xffffffff80000000
sub rcx, 0x78
movzx r10, word ptr [rbx]
xor r12, r13
add r12, 0xffff
add r15, 0
mov r8, rbp
sub rcx, 0x10
or r12, r12
or rcx, 0x800
movzx r11, word ptr [r15]
xor rcx, 0x800
mov r12, r15
add r8, 0
xor r12, 0xf0
mov rbx, 0x58
add r11, rbp
xor rbx, 0x800
and r12, 0x20
add rbx, 0x800
mov r11, qword ptr [r11]
add rbx, 1
and r12, r9
mov rdx, 1
xor r10d, dword ptr [r8]
sub r9, r11
pushfq
xor rbx, 0xf0
xor rbx, 0x800
and rdx, r8
mov r12, rbp
xor rdx, 0x20
sub rbx, 4
add r11, 0x2549b044
or rbx, 0x78
and rdx, r10
mov rax, 0
add r12, 0x42

mov r15, rdx
xor r15, dword ptr [r12]
or r15, 0x800
or rdx, 0x400
mov r14, rbp
sub r14, rbp
mov r15, rax
mov r14, rbp
pop r9
mov rcx, rbp
add rcx, 0xc0
mov rcx, qword ptr [rcx]
add r10, word ptr [rcx]
mov r9, rbp
mov add r9, 0
xor r10d, dword ptr [r9]
and rdi, 0xffffffff80000000
sub r13, 0xf0
mov r15, 0
sub r13, 0x20
mov rbx, rbp
or r13, 0x88
and rcx, 8
mov r8, 0x58

add r8, 1
or r8, 0x78
mov word ptr [rbx], r10w
add r15, rax
sub r15, rax
pop r9
mov rcx, rbp
add rcx, 0xc0
mov rcx, qword ptr [rcx]
add r10, word ptr [rcx]
mov r9, rbp
mov add r9, 0
xor r10d, dword ptr [r9]
and rdi, 0xffffffff80000000
sub r13, 0xf0
mov r15, 0
sub r13, 0x20
mov rbx, rbp
or r13, 0x88
and rcx, 8
mov r8, 0x58

add rbx, 8
si, word ptr [rbx]
mov r9, 0xffff
sub r9, 1
mov r9, rbp
mov r13, 0x58
add r9, 0
sub r13, 0x80
mov r15, r13
or rcx, r12
xor esi, dword ptr [r9]
mov r10, rbp
mov r10, 0xcc
sub r15, 0x20
xor esi, dword ptr [r10]
xor r13, 0x90
add rdi, 0x10
mov r14, rsi
mov rdx, rbp
add rdx, 0
add dword ptr [rdx], esi
xor r12, 1
mov r13, r15

mov r14, r14
xor rax, rbp
and rcx, r13
add rcx, 4
mov r8, 0x80000000
add r13, 0xffff
and rcx, 0x20
mov r10, rbp
add r13, r15
je r10, 0x89
mov word ptr [r10], si
xor rdx, r11
sub rsi, rbp
mov rdx, rbp
and rax, 0x40
or rbx, 0xf0
add rsi, 0x5a
mov r8, rcx
movzx rsi, word ptr [rsi]
mov rax, 0x200
mov rbp
mov rdx
add r13, 0x20
xor r13, 0x13
xor r14, r10
jmp 0x4a
xor rsi, 0x80
xor rbx, 0xffffffff80000000
add rsi, 0x78
mov r10b, 0x68
mov r9, 0x12
or rbx, r10
and r15, 0x78
mov r14, rbp
or r9, 8
add r14, 0x29
xor rbx, rdi
xor r15, 0x3f
or byte ptr [r14], r10b
mov rax, 0x58
mov r8, rbp
sub rsi, 0x78
add r8, 0x127
mov rdi, rbx
xor r15, 0x3f
mov r8, qword ptr [r8]
xor rsi, 1
mov rax, rbp

add r15, 0x3f
or r15, 0xffffffff80000000
xor rsi, r9
add rax, 0xc0
rdi, r14
or rsi, 1
mov rax, qword ptr [rax]
and rdi, 0x7fffffff
add rax, 2
rdi, 4
or rbx, rsi
movzx rax, word ptr [rax]
mov r9, rbp
mov r13, 0x200
mov r10, 0x58
add r9, 0
or r10, 0x20
add eax, dword ptr [r9]
xor r10, 0x40
add eax, 0x3f505c07
add r15, 0x88
mov r12, rbp
or rdi, 0x90
add r12, 0
or rbx, 0x80
add rdi, 0xf0
mov r13, 0x400
add dword ptr [r12], eax
and rsi, r8
or r10, 8
and rbx, 0x20
and rax, 0xffff
mov r11, 0
add r13, r8
or rbx, 1
shl rax, 3
add r8, rax
or rbx, r15
sub r15, 0x10
r11, r13
mov rbx, qword ptr [r8]
mov rdx, rbp
sub r13, 0x80
add rdx, 0xc0
add qword ptr [rdx], 0xd
jmp rbx

```

$$u64 \text{ res} = M_{13} + M_{14} \cdot r10$$

#2: Duplicating VM Handlers

?
?
?
?
?
?
?
?
?
...

#2: Duplicating VM Handlers

?
vm_add64
vm_xor32
?
vm_sub16
vm_shl16
vm_add8
?
vm_add64
...

#2: Duplicating VM Handlers

?
vm_add64
vm_xor32
?
vm_sub16
vm_shl16
vm_add8
?
vm_add64
...

#5: Blinding VM Bytecode

```
mov r15, 0x200
xor r15, 0x800
mov rbx, rbp
add rbx, 0xc0
mov rbx, qword ptr [rbx]
mov r13, 1
mov rcx, 0
mov r15, rbp
add r15, 0xc0
or rcx, 0x88
mov rbx, rbp
add r15, qword ptr [r15]
or r12, 0xffffffff80000000
sub rcx, 0x78
movzx r10, word ptr [rbx]
xor r12, r13
add r12, 0xffff
add r15, 0
mov r8, rbp
sub rcx, 0x10
or r12, r12
or rcx, 0x800
movzx r11, word ptr [r15]
xor rcx, 0x800
mov r12, r15
add r12, 0
xor r12, 0xf0
mov rbx, 0x58
add r11, rbp
xor rbx, 0x800
and r12, 0x20
add rbx, 0x800
mov r11, qword ptr [r11]
add rbx, 1
and r12, r9
mov rdx, 1
xor r10d, dword ptr [r8]
sub r9, r11
pushfq
xor rbx, 0xf0
xor rbx, 0x800
xor rdx, r8
mov r12, rbp
xor rdx, 0x20
sub rbx, 4
add r11, 0x2549b044
or rbx, 0x78
and rdx, r10
mov rax, 0
add r12, 0x42

mov r15, rdx
xor r10d, dword ptr [r12]
sub r15, 0x800
or rdx, 0x400
mov rsi, 0x200
add r14, rbp
sub rsi, r1
mov r15, rbp
mov r8, 0x400
sub rsi, r9
sub r15, rsi
add r15, qword ptr [r15]
and rsi, rax
and r8, 0x88
xor r1, r14
add r1, rbp
mov rdi, 0xc0
add rcx, qword ptr [rcx]
add rcx, 8
movzx r10, word ptr [rcx]
mov r9, rbp
add r9, 0
xor r10d, dword ptr [r9]
and rdi, 0xffffffff80000000
sub r13, 0xf0
mov rsi, 0
sub r13, 0x20
mov rbx, rbp
or r13, 0x88
and rcx, 8
mov r8, 0x58
xor rbx, 0xc0
mov rbx, qword ptr [rbx]
sub rcx, 0x20
add rdi, 0x80
mov r13, 0x10
add rbx, 8
mov si, word ptr [rbx]
or r9, 0xffff
sub r9, 1
mov r9, rbp
mov r12, 0x58
add r9, 0
sub r13, 0x80
mov r15, r13
or rcx, r12
xor esi, dword ptr [r9]
mov r10, rbp
add r10, 0xcc
sub r15, 0x20
xor esi, qword ptr [r10]
or rcx, 4
or rbx, rbp
mov rcx, 4
or rcx, 0x400
sub rax, rbp
or rcx, 0x80
add rcx, 0x80
mov rbx, 0x5a

add r8, 1
or r8, 0x78
word ptr [rbx], r10w
mov r15, rax
sub r15, rax
pop r9
mov rcx, rbp
and rcx, 0xc0
mov rcx, qword ptr [rcx]
add rcx, 8
movzx r10, word ptr [rcx]
mov r9, rbp
add r9, 0
xor r10d, dword ptr [r9]
and rdi, 0xffffffff80000000
sub r13, 0xf0
mov rsi, 0
sub r13, 0x20
mov rbx, rbp
or r13, 0x88
and rcx, 8
mov r8, 0x58
xor rbx, 0xc0
mov rbx, qword ptr [rbx]
sub rcx, 0x20
add rdi, 0x80
mov r13, 0x10
add rbx, 8
mov si, word ptr [rbx]
or r9, 0xffff
sub r9, 1
mov r9, rbp
mov r12, 0x58
add r9, 0
sub r13, 0x80
mov r15, r13
or rcx, r12
xor esi, dword ptr [r9]
mov r10, rbp
add r10, 0xcc
sub r15, 0x20
xor esi, qword ptr [r10]
or rcx, 4
or rbx, rbp
mov rcx, 4
or rcx, 0x400
sub rax, rbp
or rcx, 0x80
add rcx, 0x80
mov rbx, 0x5a

or r14, r14
mov rax, rbp
and rcx, r13
add r15, 4
sub r13, 0x80000000
add r13, 0xffff
and rcx, 0x20
mov r10, rbp
add r10, r8
add r13, r15
add r10, 0x89
word ptr [r10], si
xor rdx, r11
mov rsi, rbp
xor rdx, rbx
and rax, 0x40
or rbx, 0xf0
add rsi, 0x5a
mov r8, rcx
movzx rsi, word ptr [rsi]
mov rax, 0x200
mov r14, rbp
and rax, rdx
and rcx, 0x20
add r14, 0x89
or rax, 0x40
mov si, 0x7a28
xor rdx, 0x78
add rdx, 0x20
movzx r14, word ptr [r14]
mov rcx, 0x58
add rsi, rbp
xor rax, rdx
or r8, 0x80
mov r15, rsi
add r14, rbp
add r8, r15
mov rbx, 0
and rdx, 0x10
mov r14, qword ptr [r14]
pushfq
xor r11, r14
add r15, r14
mov r13, 0x12
mov r8, 0
and r14, 0x88
and r13, 0x40
add r13, 1
mov rdx, rbp

mov r14, 0x200
rdx, 0xc0
add r11, r14
or r15, 0x80
mov rdx, qword ptr [rdx]
add rdx, 0xa
add r11, 0x78
mov r8b, byte ptr [rdx]
cmp r8b, 0
je 0x49e
mov rdx, rbp
mov r11, 0x40
add r15, 1
xor r11, 0x10
add rdx, 0xc0
or r14, 4
mov r15, 0x12
mov rdx, qword ptr [rdx]
mov r11, r8
add rdx, 4
or r11, 0x80
mov r8w, word ptr [rdx]
mov r14, r8
add r8, rbp
xor r13, 4
xor r10, qword ptr [r8], r10
jmp 0x4ae
xor rsi, 0x88
xor rbx, 0xffffffff80000000
add rsi, 0x78
mov r10b, 0x68
mov r9, 0x12
or rbx, r10
mov r15, 0x78
mov r14, rbp
or r9, 8
add r14, 0x29
xor rbx, rdi
and r11, 0x3f
or byte ptr [r14], r10b
mov rax, 0x58
mov r8, rbp
sub rsi, 0x78
add r8, 0x127
mov rdi, rbx
xor rbx, 0x3f
mov r8, qword ptr [r8]
xor rsi, 1
mov rax, rbp

add r15, 0x3f
or r15, 0xffffffff80000000
xor rsi, r9
add rax, 0xc0
add r13, r14
or rsi, 1
or rax, qword ptr [rax]
and rdi, 0x7fffffff
add rax, 2
sub rsi, 4
or rbx, rsi
movzx rax, word ptr [rax]
mov r9, rbp
mov r13, 0x200
mov r10, 0x58
add r9, 0
or r10, 0x20
add eax, dword ptr [r9]
xor r10, 0x40
add eax, 0x3f505c07
add r15, 0x88
mov r12, rbp
or rdi, 0x90
add r12, 0
or rbx, 0x80
add rdi, 0xf0
mov r13, 0x400
add dword ptr [r12], eax
and rsi, r8
or r10, 8
and rbx, 0x20
and rax, 0xffff
mov r11, 0
add r13, r8
or rbx, 1
shl rax, 3
add r8, rax
or rbx, r15
sub r15, 0x10
or r11, r13
mov rbx, qword ptr [r8]
mov rdx, 1
sub r13, 0x80
add rdx, 0xc0
add qword ptr [rdx], 0xd
jmp rbx
```

#5: Blinding VM Bytecode

mov r15, 0x200	mov r15, rdx	add r8, 1	or r14, r14	mov r14, 0x200	add r15, 0x3f
xor r15, 0x800	xor r10d, dword ptr [r12]	or r8, 0x78	mov rax, rbp	add rdx, 0xc0	or r15, 0xffffffff80000000
mov rbx, rbp	sub r15, 0x800	add word ptr [rbx], r10w	and rcx, r13	add r11, r14	or rsi, r9
add rbx, 0xc0	or rdx, 0x400	mov r15, rax	or rax, 4	add r15, 0x80	add rax, 0xc0
mov rbx, qword ptr [rbx]	mov rsi, 0x200	sub r15, rax	sub r8, 0x80000000	mov rdx, qword ptr [rdx]	add rdi, r14
mov r13, 1	mov r14, rbp	pop r9	add r13, 0xffff	mov rdx, 0xa	or rsi, 1
mov rcx, 0	sub rsi, rbp	mov rcx, rbp	and rcx, 0x20	add r11, 0x78	mov rax, qword ptr [rax]
mov r15, rbp	mov r8, 0x400			mov r8b, byte ptr [rdx]	and rdi, 0x7fffffff
add r15, 0xc0	sub rsi, r9			cmp r8b, 0	add rax, 2
or rcx, 0x88	mov r8, 0x400			je 0x4ae	sub rsi, 4
add rbx, 0xb	sub r8, rsi			mov rdx, rbp	movzxb rax, word ptr [rax]
mov r15, qword ptr [r15]	add r14, 0			mov rdx, rbp	mov r9, rbp
or r12, 0xffffffff80000000	add rsi, rax			sub r11, rbp	mov r13, 0x200
sub rcx, 0x78	and r8, 0x88			and r15, 1	mov r10, 0x58
movzx r10, word ptr [rbx]	and rsi, r14			xor r11, 0x10	add r9, 0
xor r12, r13	mov rdi, rbp			add rdx, 0xc0	or r10, 0x20
add r12, 0xffff	add rdi, 0xc0			or r14, 4	add eax, dword ptr [r9]
add r15, 0	sub r8, rdi			mov r15, 0x12	xor r10, 0x40
mov r8, rbp	add r8, 0x78			mov rdx, qword ptr [rdx]	add eax, 0x3f505c07
sub rcx, 0x10	add rsi, 4			add rdx, 4	add r15, 0x88
or r12, r12	mov rcx, 0x200			or r11, 0x80	mov r12, rbp
or rcx, 0x800	rdi, qword ptr [rdi]			mov r8w, word ptr [rdx]	or rdi, 0x90
movzx r11, word ptr [r15]	add dword ptr [rsi], 0x254			mov r14, r8	add r12, 0
xor rcx, 0x800	xor rcx, 0xf0			add r8, rbp	or rbx, 0x80
mov r12, r15	add rcx, r10			xor r13, 4	add rdi, 0xf0
add r8, 0	add rdi, 6			pop r10	mov r13, 0x400
xor r12, 0xf0	rdi, 0x400			mov qword ptr [r8], r10	add dword ptr [r12], eax
mov rbx, 0x58	mov ax, word ptr [rdi]			jmp 0x4ae	
add r11, rbp	mov r8, 1			xor rsi, 0x88	and rsi, r8
xor rbx, 0x800	mov rsi, rbp			xor rbx, 0xffffffff80000000	or r10, 8
and r12, 0x20	and rcx, 8			add rsi, 0x78	and rbx, 0x20
add rbx, 0x800	sub rcx, 1			mov r10b, 0x68	and rax, 0xffff
mov r11, qword ptr [r11]	mov rcx, rdi			mov r9, 0x12	mov r11, 0
add rbx, 1	rdi, 0x29			or rbx, r10	add r13, r8
and r12, r9	or rdx, 8			mov r14, rbp	or rbx, 0
mov rdx, 1	mov r8, rsi			shl rax, 3	add r8, rax
xor r10d, dword ptr [r8]	add rcx, 4			add r14, 0x29	or rbx, r15
sub r9, r11	mov r13b, byte ptr [rsi]			xor rbx, rdi	sub r15, 0x10
pushfq	cmp r13b, 0xd2			and r15, 0x3f	or r11, r13
xor rbx, 0xf0	jbe 0x204			or byte [r14], r10b	mov rbx, qword ptr [r8]
xor rbx, 0x800	and r8, r13			mov rdx, 0x58	mov rdx, 1
rdx, r8	or rcx, r13			mov r8, rbp	sub r13, 0x80
mov r12, rbp	or rcx, 4			sub rsi, 0x78	add rdx, 0xc0
xor rdx, 0x20	mov or rcx, 4			add r8, 0x127	add qword ptr [rdx], 0xd
sub rbx, 4	or rcx, 0x400			rdi, rbx	jmp rbx
add r11, 0x2549b044	add rax, rbp			xor rdx, 0x3f	
or rbx, 0x78	or rcx, 0x80			mov r8, qword ptr [r8]	
and rdx, r10	add rax, 0			rst, 1	
mov rax, 0	add rbx, 0x5a			mov r13, r15	
add r12, 0x42				mov rdx, rbp	

```

mov r9, rbp
...
add r9, 0
...
add eax, dword ptr [r9]
...
add eax, 0x3f505c07
...
mov r12, rbp
...
add r12, 0
add dword ptr [r12], eax

```

, si

[rsi]

[r14]

ptr [r14]

[rsi], r14

pushfq

xor r11, r14

add r15, r14

mov r13, 0x12

mov r8, 0

and r14, 0x88

and r13, 0x40

add r13, 1

mov rdx, rbp

#5: Blinding VM Bytecode

```
mov r15, 0x200
xor r15, 0x800
mov rbx, rbp
add rbx, 0xc0
mov r13, qword ptr [rbx]
mov r13, 1
mov rcx, 0
mov r15, rbp
add r15, 0xc0
or rcx, 0x88
add rbx, 0xb
mov r15, qword ptr [r15]
or r12, 0xffffffff80000000
sub rcx, 0x78
movzx r10, word ptr [rbx]
xor r12, r13
add r12, 0xffff
add r15, 0
mov r8, rbp
sub rcx, 0x10
or r12, r12
or rcx, 0x800
movzx r11, word ptr [r15]
xor rcx, 0x800
mov r12, r15
add r8, 0
xor r12, 0xf0
mov rbx, 0x58
add r11, rbp
xor rbx, 0x800
and r12, 0x20
add rbx, 0x800
mov r11, qword ptr [r11]
add rbx, 1
add r12, r9
mov rdx, 1
xor r10d, dword ptr [r8]
sub r9, r11
pushfq
xor rbx, 0xf0
xor rbx, 0x800
rdx, r8
mov r12, rbp
xor rdx, 0x20
sub rbx, 4
add r11, 0x2549b044
or rbx, 0x78
and rdx, r10
mov rax, 0
add r12, 0x42

mov r15, rdx
xor r10d, dword ptr [r12]
sub r15, 0x800
or rdx, 0x400
mov r15, rax
sub r15, rax
pop r9
mov rcx, rbp

add r8, 1
or r8, 0x78
add word ptr [rbx], r10w
mov r15, rax
sub r15, rax
pop r9
mov rcx, rbp

or r14, r14
rax, rbp
and rcx, r13
add rax, 4
sub r8, 0x80000000
add r13, 0xffff
and rcx, 0x20

mov r14, 0x200
rdx, 0xc0
add r11, r14
or r15, 0x88
mov rdx, qword ptr [rdx]
add rdx, 0xa
add r11, 0x78
mov r8b, byte ptr [rdx]
cnp r8b, 0
je 0x40e
mov rdx, rbp
and r11, 0x40
and r15, 1
xor r11, 0x10
add rdx, 0xc0
or r14, 4
mov r15, 0x12
mov rdx, qword ptr [rdx]
sub r11, r8
add rdx, 4
or r11, 0x80
mov r8w, word ptr [rdx]
mov r14, r8
add r8, rbp

add r15, 0x3f
or r15, 0xffffffff80000000
and r15, r9
or r15, r9
or rax, 0xc0
rdi, r14
or r15, 1
or rax, qword ptr [rax]
rdi, 0x7fffffff
add rax, 2
sub r15, 4
or rbx, rsi
movzx rax, word ptr [rax]
mov r9, rbp
mov r13, 0x200
mov r10, 0x58
add r9, 0
or r10, 0x20
add eax, dword ptr [r9]
xor r10, 0x40
add eax, 0x3f505c07
add r15, 0x88
mov r12, rbp
or rdi, 0x90
add r12, 0
xor r15, 0x80
l, 0xf0
rdi, 0x400
qword ptr [r12], eax

mov r12, 0
add r12, 0
add dword ptr [r12], eax

mov r12, rbp
and rcx, 8
sub rcx, 1
mov rcx, rdi
rdi, 0x29
add rcx, 8
mov r8, rsi
add rcx, 4
mov r13b, byte ptr [rsi]
mov r13b, 0xd2
add r10, 0xc0
and r8, r13
xor r13, r13
or rcx, 4
or rbx, rbp
or rcx, 4
or rcx, 0x400
add rax, rbp
or rcx, 0x80
add rcx, 0x80
add rbx, 0x5a

mov r9, 0
sub r13, 0x80
mov r15, r13
add r15, rsi
or rcx, r12
xor esi, dword ptr [r9]
mov r10, rbp
add r10, 0xcc
sub r15, 0x20
add esi, dword ptr [r10]
xor r13, 0x90
add rdi, 0x10
mov r14, rsi
mov rdx, rbp
add rdx, 0
add dword ptr [rdx], esi
xor r12, 1
mov r13, r15

mov r11, r14
add r15, r14
mov r13, 0x12
mov r8, 0
and r14, 0x88
and r13, 0x40
add r13, 1
mov rdx, rbp

xor rax, 0x1111111111111111
add rsi, 0x78
mov r10b, 0x68
mov r9, 0x12
or rbx, r10
and r15, 0x78
mov r14, rbp
or r9, 8
add r14, 0x29
xor rbx, rdi
and r15, 0x3f
byte ptr [r14], r10b
mov rax, 0x58
mov r8, rbp
sub rsi, 0x78
add r8, 0x127
mov rdi, rbx
xor rbx, 0x3f
mov r8, qword ptr [r8]
rst, 1
xor rax, rbp

or r10, 8
rbx, 0x20
and rax, 0xffff
mov r11, 0
add r13, r8
or rbx, r10
shl rax, 3
add r8, rax
or rbx, r15
sub r15, 0x10
or r11, r13
mov rbx, qword ptr [r8]
mov rdx, rbp
sub r13, 0x80
add rdx, 0xc0
add qword ptr [rdx], 0xd
jmp rbx
```

No influence on underlying code's semantics

#3: No Central VM Dispatcher

mov	r15, 0x200	mov	r15, rdx	add	r8, 1	mov	r14, r14	mov	r14, 0x200	add	r15, 0x3f
mov	r15, 0x800	mov	r10d, dword ptr [r12]	or	r8, 0x78	mov	r14, r14	add	r15, 0xfffffff800000000	or	r15, 0xfffffff800000000
mov	rbx, rbp	sub	r15, 0x800	or	word ptr [rbx], r10w	and	rcx, r13	add	r11, r14	and	rsi, r9
add	rbx, 0xc0	or	rdx, 0x400	mov	r15, rax	and	rax, 4	add	r15, 0x88	add	rax, 0xc0
mov	rbx, qword ptr [rbx]	mov	rsi, 0x200	sub	r15, rax	sub	r8, -0x80000000	mov	rdx, qword ptr [rdx]	add	rdi, r14
mov	r13, 1	mov	r14, rbp	pop	r9	add	r13, 0xffff	add	rdx, 0xa	or	rsi, 1
mov	rcx, 0	sub	rsi, rsi	mov	rcx, rbp	and	rcx, 0x20	add	r11, 0x78	mov	rax, qword ptr [rax]
mov	r15, rbp	mov	rdi, rbp	mov	r10, rbp	mov	r10, rbp	add	rdi, byte ptr [rdx]	and	rdi, 0x7fffffff
add	r15, 0xc0	mov	rcx, 0x400	mov	rcx, qword ptr [rcx]	add	r13, r15	cmp	rb8, 0	add	rax, 2
or	rcx, 0x88	sub	rsi, r9	add	rcx, 8	add	r14, r8	je	0x49e	sub	rsi, 4
add	rbx, 0xb	sub	r8, rsi	movzx	r10, word ptr [rcx]	add	r10, 0x89	mov	rdx, rbp	or	rbx, rsi
mov	r15, qword ptr [r15]	add	r14, 0	mov	r9, rbp	xor	word ptr [r10], si	or	r11, 0x40	movzx	rax, word ptr [rax]
r12, 0xfffffff800000000	add	rsi, rax	add	r9, 0	xor	rdx, r11	mov	r15, 1	mov	r9, rbp	
sub	rcx, 0x78	and	r8, 0x88	xor	r10d, dword ptr [r9]	mov	rsi, rbp	xor	r11, 0x10	mov	r10, 0x200
movzx	r10, word ptr [rbx]	xor	rsi, r14	and	rdi, 0xfffffff800000000	sub	rdx, 0xc0	add	rdx, 0xc0	mov	r13, 0x58
r12, 0	add	rsi, r13	sub	r13, rbp	or	r14, r8	add	r14, 4	mov	r9, add	
add	r12, 0xffff	mov	rdi, 0xc0	mov	rsi, 0	or	rbx, 0xf0	mov	r15, 0x12	or	r10, 0x20
add	r15, 0	sub	r8, rdi	sub	r13, 0x20	or	rsi, 0x5a	mov	rdx, qword ptr [rdx]	add	eax, dword ptr [r9]
mov	r8, rbp	add	r8, 0x78	mov	rbx, rbp	mov	r8, rcx	sub	r11, r8	xor	r10, 0x40
sub	rcx, 0x10	add	rsi, 4	or	r13, 0x88	movzx	rsi, word ptr [rsi]	add	rdx, 4	add	eax, 0x3f505c07
or	r12, r12	mov	rcx, 0x200	and	rcx, 8	mov	rax, 0x200	or	r11, 0x80	add	r15, 0x88
or	rcx, 0x800	mov	rdi, qword ptr [rdi]	mov	rbx, 0x58	mov	r14, rbp	mov	rbw, word ptr [rdx]	mov	r12, rbp
movzx	r11, word ptr [r15]	add	dword ptr [rsi], 0x2549b044	add	rcx, 0xc0	and	rax, rdx	mov	r14, r8	or	rdi, 0x90
or	rcx, 0x80	add	rcx, 0x80	sub	rcx, 0x20	add	rcx, qword ptr [rbx]	mov	r8, 0	r12, add	
add	r12, r15	add	rcx, r10	add	rcx, 0x20	add	r14, 0x89	xor	r13, 4	or	rbx, 0x0
add	r8, 0	add	rdi, 6	add	rdi, 0x80	or	rax, 0x40	pop	r10	add	rdi, 0xf0
xor	r12, 0xf0	mov	r8, 0x400	sub	r13, 0x10	xor	si, 0x7a28	mov	qword ptr [r8], r10	mov	r13, 0x400
mov	rbx, 0x58	mov	ax, word ptr [rdi]	add	rbx, 8	add	rdx, 0x78	jmp	0x4ae	add	dword ptr [r12], eax
add	r11, rbp	mov	r8, 1	mov	si, word ptr [rbx]	add	rdx, 0x20	xor	rsi, 0x88	and	rsi, r8
xor	rbx, 0x800	and	rsi, rbp	sub	or	r9, 0xffff	movzx	r14, word ptr [r14]	xor	rbx, 0xfffffff800000000	
add	r12, 0x20	mov	rcx, 8	or	rcx, 8	or	rcx, 0x58	add	rsi, 0x78	and	rbx, 0x20
add	rbx, 0x800	add	rcx, 1	mov	r9, rbp	add	rsi, rbp	mov	r10b, 0x58	and	rax, 0xffff
mov	r11, qword ptr [r11]	mov	rcx, rdi	mov	r12, 0x58	xor	rax, rdx	mov	r9, 0x12	mov	r1, 0
add	rbx, 1	add	rsi, 0x29	add	r9, 0	add	r8, 0x80	or	rbx, r10	add	r13, r8
add	r12, r9	or	rcx, 8	sub	r13, 0x80	mov	r15, rsi	or	r15, 0x78	or	rbx, 1
mov	rdx, 1	mov	r8, rsi	mov	r15, r13	add	r14, rbp	and	r14, rbp	shl	rax, 3
sub	r10d, dword ptr [r8]	add	rcx, 4	or	rcx, r12						

#3: No Central VM Dispatcher

```
mov r15, 0x200
xor r15, 0x800
mov rbx, rbp
add rbx, 0xc0
mov rbx, qword ptr [rbx]
mov r13, 1
mov rcx, 0
mov r15, rbp
add r15, 0xc0
or rcx, 0x88
add rbx, 0xb
mov r15, qword ptr [r15]
or r12, 0xffffffff80000000
sub rcx, 0x78
movzx r10, word ptr [rbx]
xor r12, r13
add r12, 0xffff
add r15, 0
mov r8, rbp
sub rcx, 0x10
or r12, r12
or rcx, 0x800
movzx r11, word ptr [r15]
xor rcx, 0x800
mov r12, r15
xor r8, 0
mov r12, 0xf0
mov rbx, 0x58
add r11, rbp
xor rbx, 0x800
and r12, 0x20
add rbx, 0x800
mov r11, qword ptr [r11]
add rbx, 1
and r12, r9
mov rdx, 1
xor r10d, dword ptr [r8]
sub r9, r11
pushfq
xor rbx, 0xf0
xor rbx, 0x800
rdx, r8
mov r12, r9
xor rdx, 0x20
sub rbx, 4
add r11, 0x2549b044
or rbx, 0x78
and rdx, r10
mov rax, 0
add r12, 0x42

mov r15, rdx
xor r10d, dword ptr [r12]
sub r15, 0x800
or rdx, 0x400
mov rsi, 0x200
mov r14, rbp
rdi, rsi
rdi, rbp
mov r8, 0x400
sub rsi, r9
sub r8, rsi
mov r15, qword ptr [r15]
add rsi, rax
and r8, 0x88
xor rsi, r14
rdi, rbp
add rsi, 0xc0
mov r8, rdi
rdi, 0x78
add rsi, 4
rcx, 0x200
rdi, qword ptr [rdi]
dword ptr [rsi], 0x2549
xor rcx, 0xf0
xor rcx, r10
add rdi, 6
rdi, 0
mov r8, 0x400
ax, word ptr [rdi]
mov r8, 1
rdi, rbp
mov rcx, 8
rdi, 1
rcx, 1
mov rcx, rdi
rdi, 0x29
add rsi, 8
or r8, rsi
add rcx, 4
mov r13b, byte ptr [rsi]
mov r13b, 0xd2
xor r8, r13
rdi, r13
xor rcx, 4
or rbx, rbp
mov rcx, 4
or rcx, 0x400
rdi, rbp
add rcx, 0x80
add rcx, 0x80
rbx, 0x5a

add r8, 1
or r8, 0x78
add word ptr [rbx], r10w
mov r15, rax
sub r15, rax
pop r9
rcx, rbp

or r14, r14
rax, rbp
and rcx, r13
add rax, 4
sub r8, 0x80000000
add r13, 0xffff
and rcx, 0x20

mov r14, 0x200
rdx, 0xc0
add r11, r14
or r15, 0x80
mov rdx, qword ptr [rdx]
add rdx, 0xa
add r11, 0x78
mov r8b, byte ptr [rdx]
cmp r8b, 0
je 0x49e
mov rdx, rbp
sub r11, 0x40
and r15, 1
xor r11, 0x10
add rdx, 0xc0
or r14, 4
mov r15, 0x12
mov rdx, qword ptr [rdx]
sub r11, r8
add rax, 4
or r11, 0x80
mov r8w, word ptr [rdx]
mov r14, r8
add r8, rbp
xor r13, 4
pop r10
mov qword ptr [r8], r10
jmp 0x4ae
xor rsi, 0x88
xor rbx, 0xffffffff80000000
add rsi, 0x78
mov r10b, 0x68
mov r9, 0x12
or rbx, r10
and r15, 0x78
mov r14, rbp
or r9, 8
add r14, 0x29
xor rbx, rdi
and r15, 0x3f
or byte ptr [r14], r10b
mov rax, 0x58
mov r8, rbp
sub rsi, 0x78
add r8, 0x127
mov rdi, rbx
xor rbx, 0x3f
mov r8, qword ptr [r8]
xor rsi, 1
mov rax, rbp

add r15, 0x3f
or r15, 0xffffffff80000000
or rsi, r9
add rax, 0xc0
add rdi, r14
or rsi, 1
mov rax, qword ptr [rax]
and rdi, 0x7fffffff
add rax, 2
sub rsi, 4
or rbx, rsi
movzx r9, word ptr [rax]
mov r13, 0x200
r10, 0x58
add r9, 0
or r10, 0x20
add eax, dword ptr [r9]
xor r10, 0x40
xor eax, 0x3f505c07
add r15, 0x88
mov r12, rbp
or rdi, 0x90
add r12, 0
or rbx, 0x80
rdi, 0xf0
mov r13, 0x400
add dword ptr [r12], eax
and rsi, r8
or r10, 8
xor rbx, 0x20
and rax, 0xffff
r11, 0
add r13, r8
or rbx, 1
shl rax, 3
add r8, rax
or rbx, r15
sub r15, 0x10
or r11, r13
mov rbx, qword ptr [r8]
mov rdx, rbp
sub r13, 0x80
rdx, 0xc0
add qword ptr [rdx], 0xd
jmp rbx
```

```
or rbx, 1
add r8, rax
or rbx, r15
sub r15, 0x10
or r11, r13
mov rbx, qword ptr [r8]
mov rdx, rbp
sub r13, 0x80
add rdx, 0xc0
add qword ptr [rdx], 0xd
jmp rbx
```

, si

[rsi]

[r14]

[r14]

[rsi], r14

pushfq

xor r11, r14

mov r15, r14

mov r13, 0x12

mov r8, 0

and r14, 0x88

and r13, 0x40

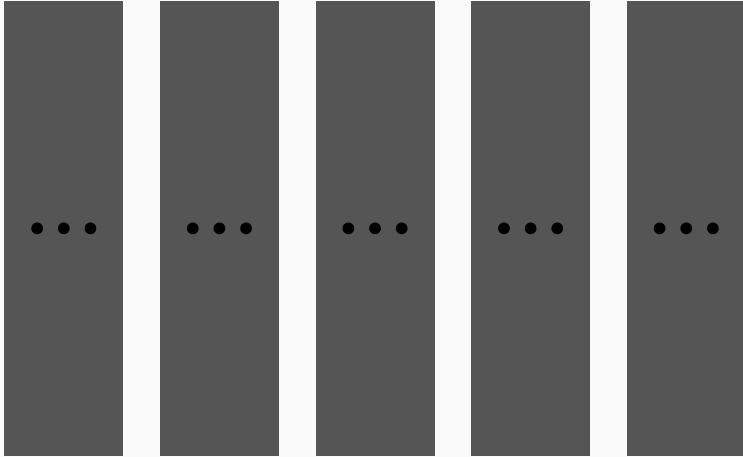
add r13, 1

mov rdx, rbp

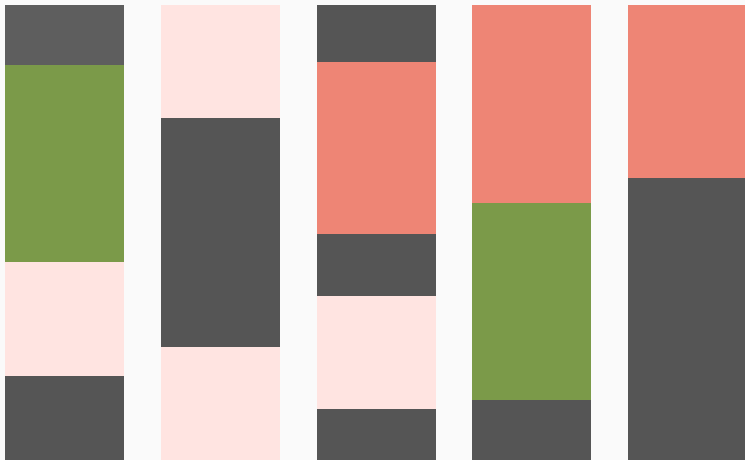
#3: No Central VM Dispatcher

Split at indirect control-flow transfers

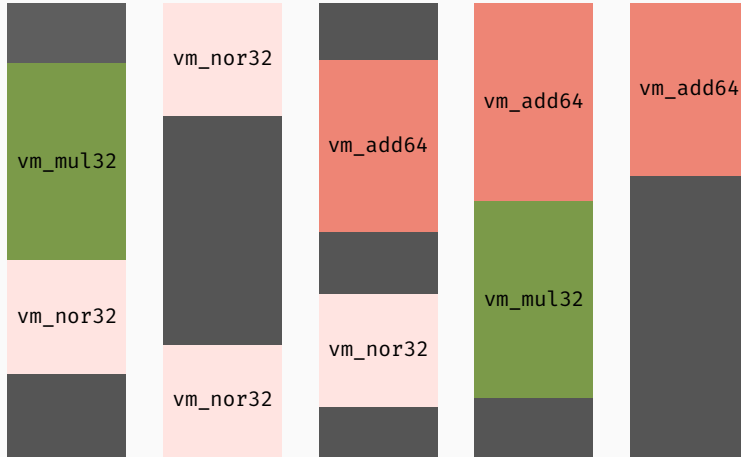
#4: No Explicit Handler Table



#4: No Explicit Handler Table



#4: No Explicit Handler Table



Conclusion

1. syntactic complexity insignificant

1. syntactic complexity insignificant
2. semantic complexity low within specified boundaries

1. syntactic complexity insignificant
2. semantic complexity low within specified boundaries
3. learn underlying code's semantics despite obfuscation

1. syntactic complexity insignificant
2. semantic complexity low within specified boundaries
3. learn underlying code's semantics despite obfuscation

Program Synthesis as an orthogonal approach to traditional techniques

Limitations

Implementation Shortcomings

choosing *meaningful* code window boundaries

$$(x \oplus y) + 2 \cdot (x \wedge y) \quad \text{vs.} \quad (x \oplus y) + 2$$

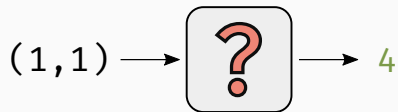
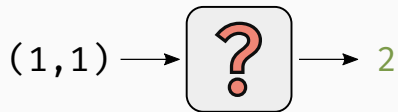
constants

$$x + 15324326921$$

control-flow operations

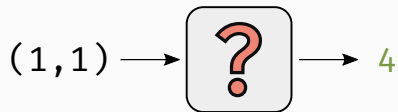
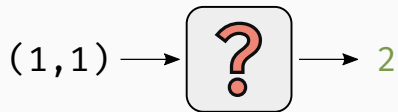
$$x \ ? \ y \ : \ z$$

Limitations



non-determinism

Limitations

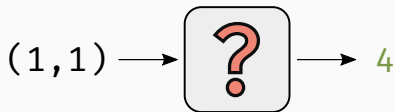
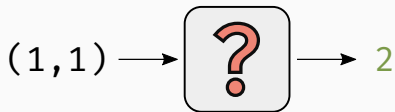


non-determinism

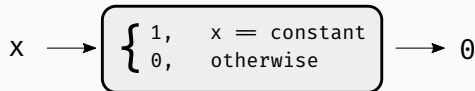


semantic complexity

Limitations



non-determinism



point functions



semantic complexity

Do try it at home!

[Code](#) [Issues 1](#) [Pull requests 0](#) [Projects 0](#) [Insights](#)

Branch: **master** [syntia](#) / **samples** / [Create new file](#) [Find file](#) [History](#)

 **mrphrazer** added MBA samples from tigress Latest commit 91a5c16 7 days ago

..

 info	added VM handler samples for vmprotect and themida	7 days ago
 mba/tigress	added MBA samples from tigress	7 days ago
 themida/tiger_white	added VM handler samples for vmprotect and themida	7 days ago
 vmprotect	added VM handler samples for vmprotect and themida	7 days ago
 tigress_mba_trace.bin	initial commit	15 days ago
 vmprotect_add16_trace.bin	initial commit	15 days ago

- obfuscation techniques (opaque predicates, VM, MBA)
- symbolic execution for syntactic deobfuscation
- program synthesis for semantic deobfuscation

<https://github.com/RUB-SysSec/syntia>