Loki: Hardening Code Obfuscation Against Automated Attacks

Usenix Security 2022, Boston

August 12, 2022

Moritz Schloegel[*], Tim Blazytko[*], Moritz Contag[*], Cornelius Aschermann[*], Julius Basler[*], Thorsten Holz[†], Ali Abbasi[†]

[*] *Ruhr-Universität Bochum*

[†] *CISPA Helmholtz Center for Information Security*

- VM-based obfuscation
- Automated attacks on VMs
- **Hardening code obfuscation**

Motivation

~~Prevent~~ **Complicate** reverse engineering attempts.

- intellectual property
- Digital Rights Management (DRM)
- abuse prevention, e.g., Google Botguard
- malicious payloads

VM-based Obfuscation

```
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
```

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  r0
   vpop r1
   vld  r2
   vld  r1
   vadd r1
   vld  r2
   vpop r0
   vld  r1
   vpop r2
   vldi #1
   vld  r3
   vsub r3
   vld  #0
   veq  r3
   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
     69 6e 67 20 74 6f 20
        65 2e de ad be ef
```

BUT: interpreter knows how to run obufscated code

BUT: interpreter knows how to run obufscated code

$\Rightarrow$ attack VM interpreter

Breaking Virtual Machine Obfuscation

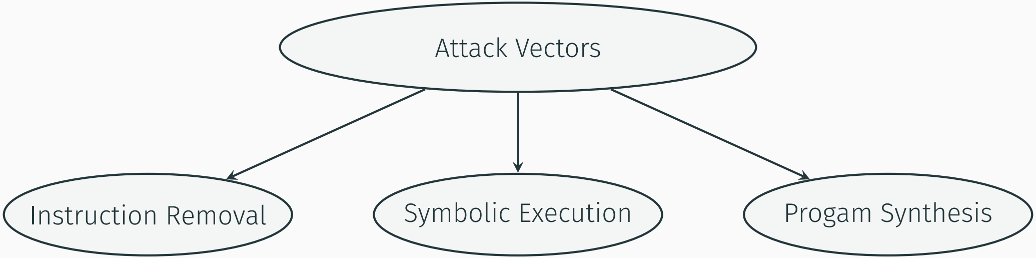
- locate VM interpreter components
- **simplify handlers** with program analysis techniques
- reconstruct original code

- locate VM interpreter components
- **simplify handlers** with program analysis techniques
- reconstruct original code

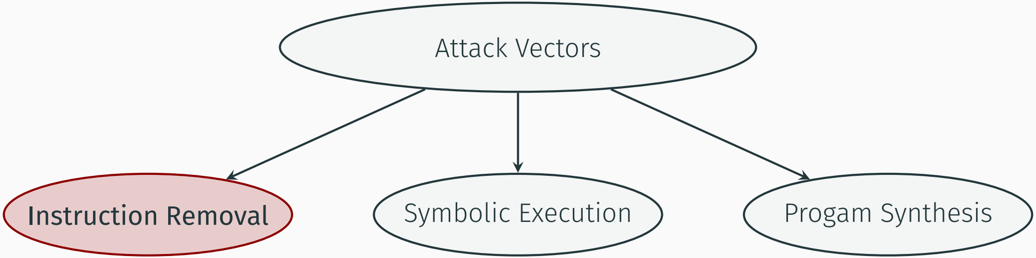
our focus

Automated Attacks on VMs

VM Attack Landscape



VM Attack Landscape



Compiler Optimizations

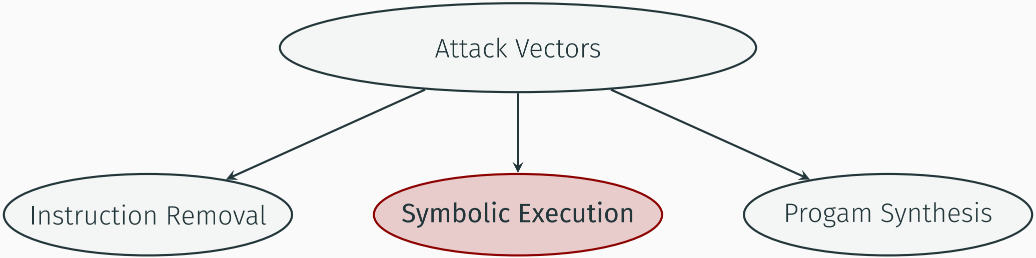
```
mov eax, 0xdead
mov eax, 0x1234
not eax
push eax
mov eax, 0x5678
mov ecx, ecx
add eax, 0x1111
add ecx, 0x0
mov edx, eax
pop eax
not eax
ret
```

Compiler Optimizations

```
×
mov eax, 0x1234
×
×
×
×
×
×
mov edx, 0x6789
×
×
ret
```

- dead code elimination
- constant folding
- constant propagation
- peephole optimization
- ...

VM Attack Landscape



Symbolic Execution: A Syntactic Approach

```
__handler_vdouble:
  not   rcx
  not   rcx
  add   rcx, rcx
  jmp   __vm_dispatcher
```

Handler doubling content of `rcx`

Symbolic Execution: A Syntactic Approach

```
__handler_vdouble:
• not   rcx
  not   rcx
  add   rcx, rcx
  jmp   __vm_dispatcher
```

Handler doubling content of `rcx`

$\mathrm{rcx} \leftarrow \neg \mathrm{rcx}$

Symbolic Execution: A Syntactic Approach

```
__handler_vdouble:
  not   rcx
• not   rcx
  add   rcx, rcx
  jmp   __vm_dispatcher
```

Handler doubling content of `rcx`

$$\mathrm{rcx} \leftarrow \neg \mathrm{rcx}$$

$$\mathrm{rcx} \leftarrow \neg(\neg \mathrm{rcx})$$

Symbolic Execution: A Syntactic Approach

```
__handler_vdouble:
  not   rcx
• not   rcx
  add   rcx, rcx
  jmp   __vm_dispatcher
```

Handler doubling content of `rcx`

$$\text{rcx} \leftarrow \neg\text{rcx}$$

$$\text{rcx} \leftarrow \neg(\neg\text{rcx}) = \text{rcx}$$

Symbolic Execution: A Syntactic Approach

```
__handler_vdouble:
  not   rcx
  not   rcx
• add   rcx, rcx
  jmp   __vm_dispatcher
```

Handler doubling content of `rcx`

$$\mathrm{rcx} \leftarrow \neg \mathrm{rcx}$$
$$\mathrm{rcx} \leftarrow \neg(\neg \mathrm{rcx}) = \mathrm{rcx}$$
$$\mathrm{rcx} \leftarrow \mathrm{rcx} + \mathrm{rcx}$$

Symbolic Execution: A Syntactic Approach

```
__handler_vdouble:
  not   rcx
  not   rcx
• add   rcx, rcx
  jmp   __vm_dispatcher
```

Handler doubling content of `rcx`

$$\text{rcx} \leftarrow \neg\text{rcx}$$

$$\text{rcx} \leftarrow \neg(\neg\text{rcx}) = \text{rcx}$$

$$\text{rcx} \leftarrow \text{rcx} + \text{rcx} = \text{rcx} \ll 1$$

Symbolic Execution: A Syntactic Approach

```
__handler_vdouble:
  not   rcx
  not   rcx
  add   rcx, rcx
• jmp   __vm_dispatcher
```

Handler doubling content of `rcx`

$$\mathrm{rcx} \leftarrow \neg\mathrm{rcx}$$

$$\mathrm{rcx} \leftarrow \neg(\neg\mathrm{rcx}) = \mathrm{rcx}$$

$$\mathrm{rcx} \leftarrow \mathrm{rcx} + \mathrm{rcx} = \mathrm{rcx} \ll 1$$

Symbolic Execution: A Syntactic Approach

```
__handler_vdouble:
  not   rcx
  not   rcx
  add   rcx, rcx
  jmp   __vm_dispatcher
```

Handler doubling content of `rcx`

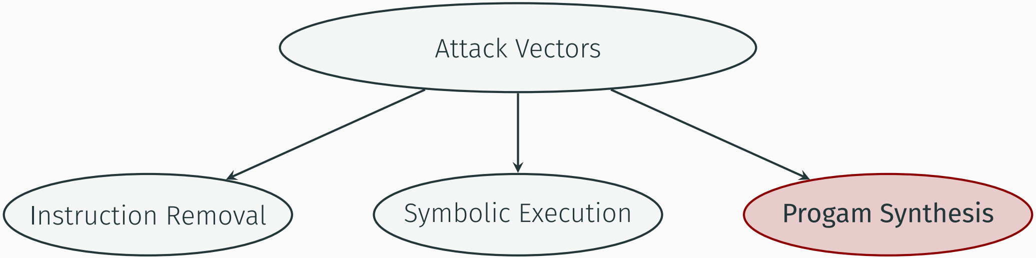
$\text{rcx} \leftarrow \neg\text{rcx}$

$\text{rcx} \leftarrow \neg(\neg\text{rcx}) = \text{rcx}$

$\text{rcx} = \text{rcx} \ll 1$

$$\text{rcx} \leftarrow \text{rcx} \ll 1$$

VM Attack Landscape



Program Synthesis: A Semantic Approach

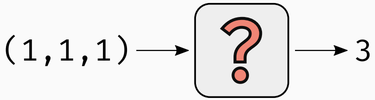
We use handler f as a black-box:

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

Program Synthesis: A Semantic Approach

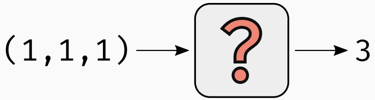
We use handler f as a black-box:

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



We use handler f as a black-box:

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

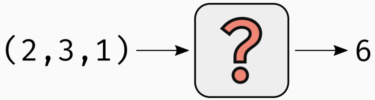


$(1,1,1) \rightarrow 3$

Program Synthesis: A Semantic Approach

We use handler f as a black-box:

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

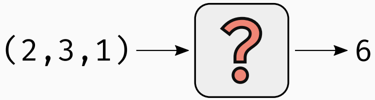


$(1, 1, 1) \rightarrow 3$

Program Synthesis: A Semantic Approach

We use handler f as a black-box:

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



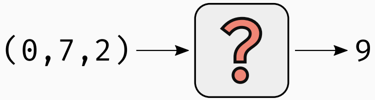
$(1,1,1) \rightarrow 3$

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

Program Synthesis: A Semantic Approach

We use handler f as a black-box:

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



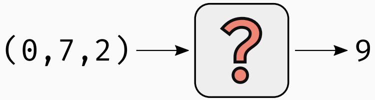
$(1, 1, 1) \rightarrow 3$

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

Program Synthesis: A Semantic Approach

We use handler 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 handler 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 h that has the same I/O behavior.

We use handler f as a black-box:

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

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

$\to 3$

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

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

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

Loki: Hardening Code Obfuscation

❶ Merging core semantics

❷ Complex, target-specific expressions

❸ Mixed Boolean-Arithmetic

❶ Merging Core Semantics

$$h(x, y, c) := x + y \qquad g(x, y, c) := x - y << c$$

❶ Merging Core Semantics

$$\overbrace{h(x, y, c) := x + y}^{\textit{Handler}} \qquad g(x, y, c) := x - y << c$$

➊ Merging Core Semantics

$$h(x, y, c) := \underbrace{x + y}_{\textit{Core Semantics}}$$

$$g(x, y, c) := x - y << c$$

❶ Merging Core Semantics

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

$$g(x, y, c) := x - y << c$$

$$f(x, y, c, k) := \begin{cases} x + y & \text{if } k == 0 \\ x - y << c & \text{if } k == 1 \end{cases}$$

❶ Merging Core Semantics

$$h(x, y, c) := x + y \qquad\qquad g(x, y, c) := x - y << c$$

$$f(x, y, c, k) := \begin{cases} x + y & \text{if } k == 0 \\ x - y << c & \text{if } k == 1 \end{cases}$$

$h(x, y, c) := x + y$ $g(x, y, c) := x - y << c$

Key-dependent core semantics

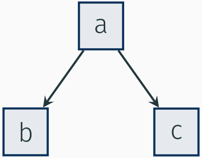
$$f(x, y, c, k) := \begin{cases} x + y & \text{if } k == 0 \\ x - y << c & \text{if } k == 1 \end{cases}$$

Polynomial Encodings and Branch-free Code

$$f(x, y, c, k) := \begin{cases} x + y & \text{if } k == 0 \\ x - y << c & \text{if } k == 1 \end{cases}$$

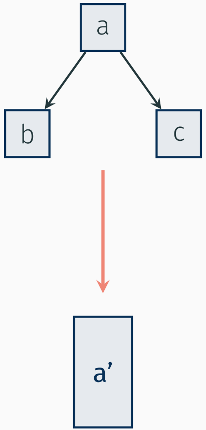
Polynomial Encodings and Branch-free Code

$$f(x, y, c, k) := \begin{cases} x + y & \text{if } k == 0 \\ x - y << c & \text{if } k == 1 \end{cases}$$



Polynomial Encodings and Branch-free Code

$$f(x, y, c, k) := \begin{cases} x + y & \text{if } k == 0 \\ x - y << c & \text{if } k == 1 \end{cases}$$

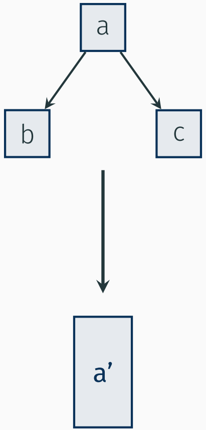


Polynomial Encodings and Branch-free Code

$$f(x, y, c, k) := \begin{cases} x + y & \text{if } k == 0 \\ x - y << c & \text{if } k == 1 \end{cases}$$

$\Updownarrow$ *equal*

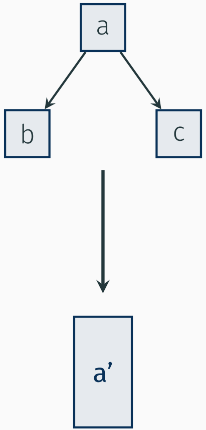
$$\begin{aligned} f(x, y, c, k) := \quad & (k == 0) \cdot x + y \\ + \quad & (k == 1) \cdot x - y << c \end{aligned}$$



$$f(x, y, c, k) := \begin{cases} x + y & \text{if } k == 0 \\ x - y << c & \text{if } k == 1 \end{cases}$$

$$\Updownarrow$$

$$f(x, y, c, k) := \quad (k == 0) \cdot x + y \quad + \quad (k == 1) \cdot x - y << c$$



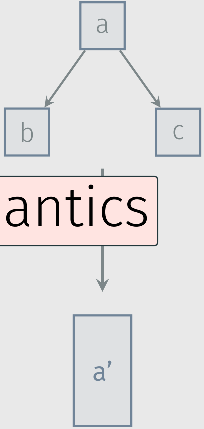
Polynomial Encodings and Branch-free Code

$$f(x, y, c, k) := \begin{cases} x + y & \text{if } k == 0 \\ x - y << c & \text{if } k == 1 \end{cases}$$

$$\Updownarrow$$

Interlocking of core semantics

$$f(x, y, c, k) := \quad (k == 0) \cdot x + y \quad + \quad (k == 1) \cdot x - y << c$$



Hardening Key Selection

$$
\begin{aligned}
f(x, y, c, k) := \quad & (k == 0) \cdot x + y \\
+ \quad & (k == 1) \cdot x - y << c
\end{aligned}
$$

Hardening Key Selection

$$f(x, y, c, k) := (n \mod k == 0) \cdot x + y + (pf(k)) \cdot x - y << c$$

Hardening Key Selection

Factorization

$$f(x, y, c, k) := (n \mod k == 0) \cdot x + y$$
$$+ (pf(k)) \cdot x - y << c$$

Hardening Key Selection

$$
\begin{aligned}
f(x, y, c, k) := \quad & (n \mod k == 0) & \cdot \quad & x + y \\
+ \quad & (pf(k)) & \cdot \quad & x - y << c
\end{aligned}
$$

Hardening Key Selection

$$
\begin{aligned}
f(x, y, c, k) := \quad & (n \mod k == 0) && \cdot \quad x + y \\
+ \quad & (pf(k)) && \cdot \quad x - y << c
\end{aligned}
$$

↑

Point Function pf(k)

$$f(x, y, c, k) := \quad (n \mod k == 0) \quad \cdot \quad x + y$$

$$\ldots y << c$$

Prevent static attacks

↑

Point Function pf(k)

❷ Semantically Complex Arithmetic Operations

__v_add
__v_mul
__v_add
__v_add

→

__v_add_mul_add_add

❷ Semantically Complex Arithmetic Operations

$$
\begin{aligned}
f(x, y, c, k) := \quad & (n_1 \mod k == 0) \cdot x + y \\
+ \quad & pf(k) \cdot x - y << c
\end{aligned}
$$

❷ Semantically Complex Arithmetic Operations

$$f(x, y, c, k) := (n_1 \mod k == 0) \cdot x + y + (x + x)$$
$$+ \; pf(k) \cdot x - y \cdot (x + y)$$

❷ Semantically Complex Arithmetic Operations

$$f(x, y, c, k) := (n_1 \mod k == 0) \cdot \overbrace{x + y + (x + x)}^{\textit{target specific}} + pf(k) \cdot x - y \cdot (x + y)$$

target specific

$f(x, y, c, k) \ldots x + x)$

$+ \quad pf(k) \quad \cdot \quad x - y \cdot (x + y)$

Thwarts program synthesis

❸ Syntactically Complex Expressions

$$\begin{aligned} f(x, y, c, k) := \quad & (n_1 \mod k == 0) \cdot x + y + (x + x) \\ + \quad & pf(k) \cdot x - y \cdot (x + y) \end{aligned}$$

❸ Syntactically Complex Expressions

$$
\begin{aligned}
f(x, y, c, k) := \quad & (n_1 \mod k == 0) \cdot ((x \oplus y) + 2 \cdot (x \wedge y)) + (x \ll 1) \\
+ \quad & pf(k) \cdot (x + \neg y + 1) \cdot ((x \oplus y) + 2 \cdot (x \wedge y))
\end{aligned}
$$

$f(x, y, c, k) :=$ … $)) + (x \ll 1)$

… $y) + 2 \cdot (x \wedge y))$

Prevent symbolic execution

$$x - y \cdot (x + y)$$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

   $\ldots$

50) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

$$x - y \cdot (x + y)$$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

$\dots$

50) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

Mixed Boolean-Arithmetic Expressions

$x - y \cdot (x + y)$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

$\cdots$

50) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

Mixed Boolean-Arithmetic Expressions

$$x - y \cdot (x + y)$$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

$\dots$

50) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

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

Mixed Boolean-Arithmetic Expressions

$$x - y \cdot (x + y)$$

$$\Big\Updownarrow \textit{equal}$$

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

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

$\dots$

50) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

$$x - y \cdot (x + y)$$

Rewriting rules:

$$
\begin{aligned}
&1) \quad x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y) \\
&2) \quad x \oplus y \rightarrow (x \vee y) - (x \wedge y) \\
&\qquad \ldots \\
&50) \quad x \wedge y \rightarrow (\neg x \vee y) - \neg x
\end{aligned}
$$

$$\boxed{x - y \cdot ((x \oplus y) + 2 \cdot (x \wedge y))}$$

final expression

Traditional Approach

Mixed Boolean-Arithmetic Expressions

$$x - y \cdot (x + y)$$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

$\dots$

50) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

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

final expression

Mixed Boolean-Arithmetic Expressions

$$x - y \cdot (x + y)$$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

$\ldots$

850,000) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

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

final expression

$$x - y \cdot (x + y)$$

Rewriting rules:

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

Lookup table w/ *all* identities

850,000) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

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

final expression

Mixed Boolean-Arithmetic Expressions

$$x - y \cdot (x + y)$$

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \wedge y)$
2) $x \oplus y \rightarrow (x \vee y) - (x \wedge y)$

$\ldots$

850,000) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

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

~~*final expression*~~

Mixed Boolean-Arithmetic Expressions

$$x - y \cdot (x + y)$$

$$x - y \cdot ((x \oplus y) + 2 \cdot (x \land y))$$

~~*final expression*~~

Rewriting rules:

1) $x + y \rightarrow (x \oplus y) + 2 \cdot (x \land y)$
2) $x \oplus y \rightarrow (x \lor y) - (x \land y)$

$\cdots$

850,000) $x \land y \rightarrow (\neg x \lor y) - \neg x$

Recursive Approach

$$x - y \cdot (x + y)$$

Rewriting rules:

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

2) … $) - (x \wedge y)$

850,000) $x \wedge y \rightarrow (\neg x \vee y) - \neg x$

Recursive Rewriting

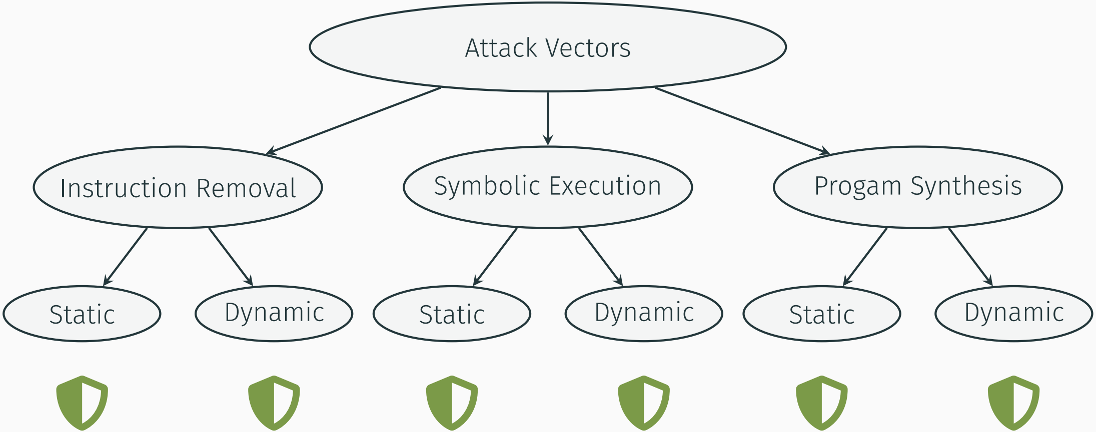
$$x - y \cdot ((x \oplus y) + 2 \cdot (x \wedge y))$$

~~*final expression*~~

Recursive Approach

Putting it all together

Putting it all together



Takeaways

- Automated techniques can simplify VM-based obfuscation
- Merging core semantics increases complexity
- Key encodings stall static attackers
- Complex, target-specific expressions thwart program synthesis
- Recursive rewriting of MBAs thwarts symbolic execution

Takeaways

- Automated techniques can simplify VM-based obfuscation
- Merging core semantics increases complexity
- Key encodings stall static attackers
- Complex, target-specific expressions thwart program synthesis
- Recursive rewriting of MBAs thwarts symbolic execution

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

Artifact: `https://github.com/RUB-SysSec/loki`

Contact

Moritz Schloegel

@m_u00d8

moritz-schloegel

Tim Blazytko

@mr_phrazer

https://synthesis.to

We are always open for questions, discussion, or collaborations!