

# Analyzing Memory Accesses in Obfuscated x86 Executables

Michael Venable  
Mohamed R. Choucane  
Md. Enamul Karim  
Arun Lakhota (Presenter)

DIMVA 2005  
Wien

# Talk Contents

---

Motivation

Previous work

VSA (Balakrishnan and Reps, 2004)

Abstract Stack Graph (Lakhotia and Kumar 2004)

VSA+ASG

Domain

Interpretation example

Application

Detect Call obfuscations

Implementation

Future and Conclusions

# Motivation

---

- Past project: detecting malicious behavior
  - Match models of malicious behavior
    - models typically involved *system calls*
- *Typical application of accepted techniques*
  - implemented prototype using IDA Pro
  - standard application of formal theory  
(model checking)
  - tried against common virus (Win32.Evol)

# Result: FAILURE

---

- Our original tool didn't stand a chance
  - our technology wasn't even tested
  - virus writer fouled up the *entire analysis pipeline*
- Success out of failure:
  - showed us where a major battle lies
  - refocus on hardening analysis techniques

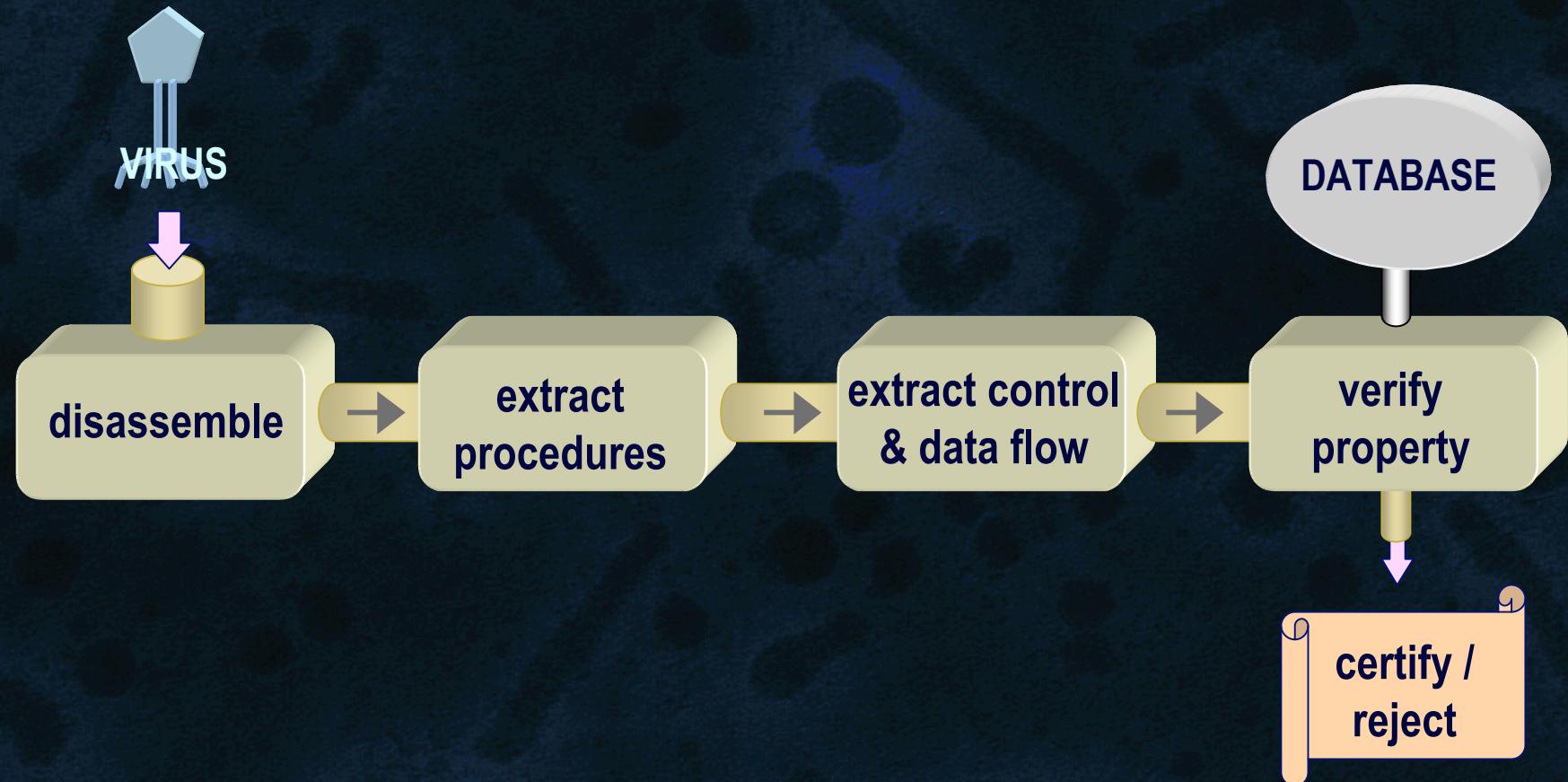
# Code analysis: benign/adversarial

---

- Half a century of program analysis
  - compilers, optimizers, checkers, refactoring tools
  - read in programs, analyze, visualize, transform
- Mostly for *friendly* programs
  - yet these are often used to battle malware
  - not prepared to face adversarial code
    - **soft targets**
    - **fragile infrastructures**
  - would we send Mouseketeers to battlefield?

# Typical analysis pipelines

---



# Problem: Not hardened

---



# Attack: Disassembly



	ORIG BYTES	ASSEMBLY
401063:	5d	pop %ebp
401064:	c3	ret
401065:	55	push %ebp
401066:	89 e5	mov %esp,%ebp
401068:	83 ec 08	sub \$0x8,%esp
40106b:	eb 05	jmp 0x401072
40106c:	e8 ee ff ff ff e8	movl 0\$0018ffff,%esi
401073:	e9 ffffffcfffffc	jmp 0\$000000000077
401078:	43 45 fc 00 00 00 00	imovl %ebp,0xfffffff(%ebp)
401079:	81 7d fc e7 03 00 00	cldpl \$0x3e7,0xfffffff(%ebp)

bad disassembly  
(no jump target)

jump over junk

malicious func

# Attack: Extract procedures



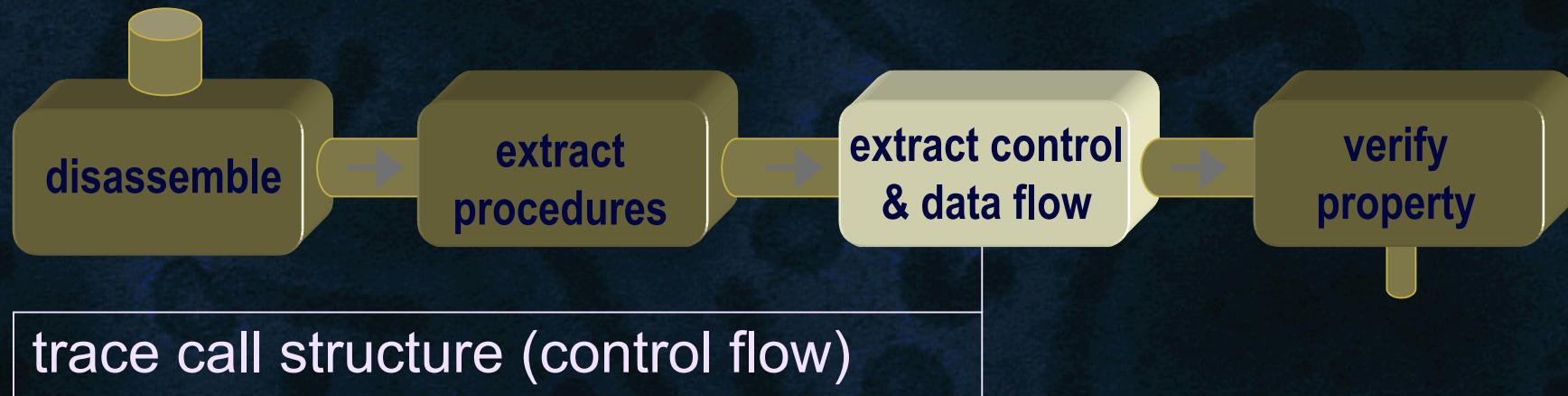
```
401063: 5d  
401064: c3  
401065 <_malicious>:  
401065: 55  
401066: 89 e5  
401068: 83 ec 08  
40106b: eb 05  
40106d: e8 ee ff ff ff  
401072: e8 e9 ff ff ff  
401077: c7 45 fc 00 00 00 00
```

```
pop %ebp  
ret  
  
push %ebp  
mov %esp,%ebp  
sub $0x8,%esp  
jmp 401072 <_malicious+0xd>  
call 401060 <_sendLotsOfEmail>  
call 401060 <_sendLotsOfEmail>  
movl $0x0,0xffffffffc(%ebp)
```

malicious func

jmp 401072 <\_malicious+0xd>

# Attack: Extract CF & DF



```
401063: 5d  
401064: c3  
401065 <_malicious:>  
401065: 55  
401066: 89 e5  
401068: 83 ec 08  
40106b: ff36578 10 40 00  
40106d: ff835e6ff 10f40 00  
401072: e8 e9 ff ff ff  
401078: c7 45 fc 00 00 00 00
```

L0: call F  
L1: →  
pop %ebp  
ret

L0: push L1  
push F  
L1: ret

instr. substitution

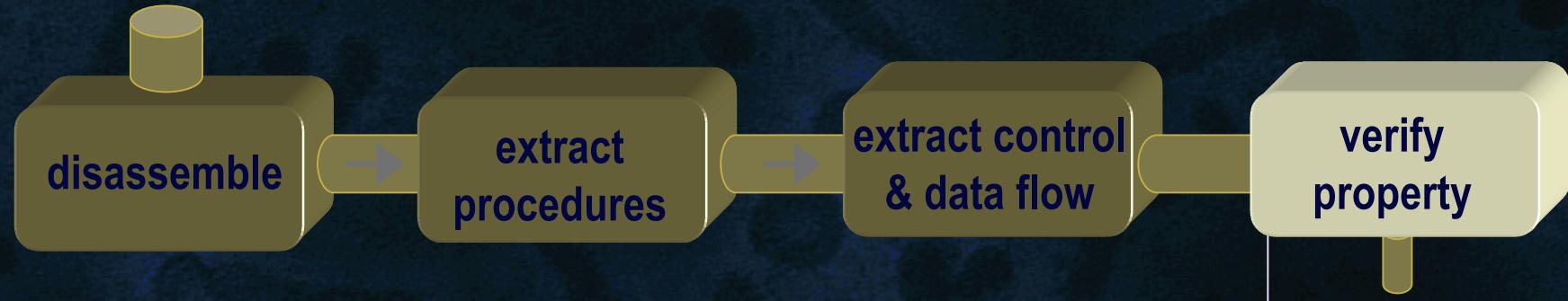
push %ebp  
mov %esp,%ebp  
sub \$0x8,%esp

pushl \$0100723 11\_malicious\_00d#0>  
pushl \$4010666 0<seedbbts@Offmain>  
call 401060 1\_\_condLotsOfEmail!

no call found

movl \$0x0,0xffffffffc(%ebp)

# Attack: Verify property



verify security or match pattern/signature

```
push x  
push y →  
ret
```

```
push x  
push z  
pop  
push y  
ret
```

```
pushl 401078 <_malicious+0x13>  
pushl 401060 <_sendLotsOfEmail>  
ret  
movl $0x0,0xffffffffc(%ebp)
```

- Transformations destroy signature/pattern match
  - eg metamorphic viruses: self-transforming
  - instruction substitution, nop insertion, etc.

# Motivation summary

---

- Soft target attacks
  - disassembler disruption
  - obfuscated calls
  - metamorphic transformations
- Pipeline disruptions
  - silent failures from every stage

# A shift in research focus

---

## Question

How to make program analysis battle-ready?

# Advertisement: Portfolio of results

---

Create  
malware  
phylogeny



Deobfuscate  
Calls

Reverse self  
transformations

All results are “Patent pending”

# Talk Contents

---

Motivation

Previous work

VSA (Balakrishnan and Reps, 2004)

Abstract Stack Graph (Lakhotia and Kumar 2004)

VSA+ASG

Domain

Interpretation example

Application

Detect Call obfuscations

Implementation

Future and Conclusions

# Value-Set Analysis

---

- Determines values of program variables statically
- RIC: Reduced Interval Congruence
  - Interval:  $[0,4] = \{0, 1, 2, 3, 4\}$
  - RIC:  $4[0,4] + 10 = \{10, 14, 18, 22, 26\}$
- Can ensure memory accesses align to memory boundaries

# Example

---



# Operations

---

- Easily supported operations
  - ADD
  - SUB
  - MOV
  - others
- Unsupported Operations
  - DIV
  - OR
  - AND
  - ...

# VSA Limitations

---

- Assumes executable conforms to standard conventions
- Depends on control-flow graph
- We would like to remove these two requirements

# Abstract Stack

---

- Each node stores an instruction address
- Example:

L1:	PUSH	20
L2:	PUSH	10
L3:	POP	ebx
L4:	PUSH	5

Actual Stack

5
20

Abstract Stack

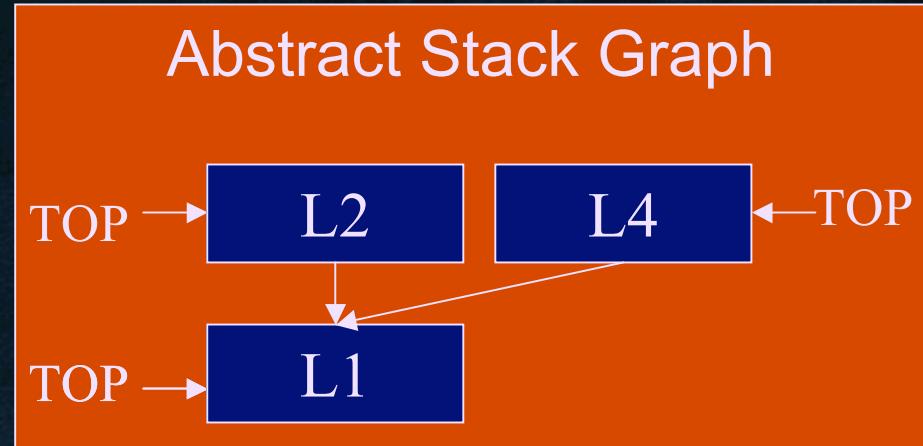
L4
L1

# Abstract Stack Graph

---

- Contains all possible abstract stack
- Example:

L1:	PUSH	20
L2:	PUSH	10
L3:	POP	ebx
L4:	PUSH	5



# Abstract Stack Graph

---

- Apply set of rules to detect obfuscations
- Limitations
  - Does not know register/memory contents
  - Can't handle instructions like:
    - sub esp, eax
    - mov esp, eax
  - Can't handle indirect jumps/calls

# Talk Contents

---

Motivation

Previous work

VSA (Balakrishnan and Reps, 2004)

Abstract Stack Graph (Lakhotia and Kumar 2004)

VSA+ASG

Domain

Interpretation example

Application

Detect Call obfuscations

Implementation

Future and Conclusions

# Our Goal

---

- VSA+ASG
- Achieve:
  - Support more instructions (ex. sub esp, eax)
  - Support indirect jumps/calls
  - Not rely on CFG
  - Do not assume binary obeys common standards

# Domain

---

- RIC
- STACK-LOCATION
- VALUE := 2-tuple
  - RIC<sub>+</sub>
  - P(STACK-LOCATION)<sub>+</sub>
- STATE := 3-tuple
  - REGISTER → VALUE
  - STACK-LOCATION → VALUE
  - STACK-LOCATION × STACK-LOCATION

# Operation: Addition

---

$+ : \text{VALUE} \times \text{VALUE} \times \text{STATE} \rightarrow \text{VALUE}$

Pairwise addition of both components of VALUE

$+ : \text{RIC} \times \text{RIC} \rightarrow \text{RIC}$

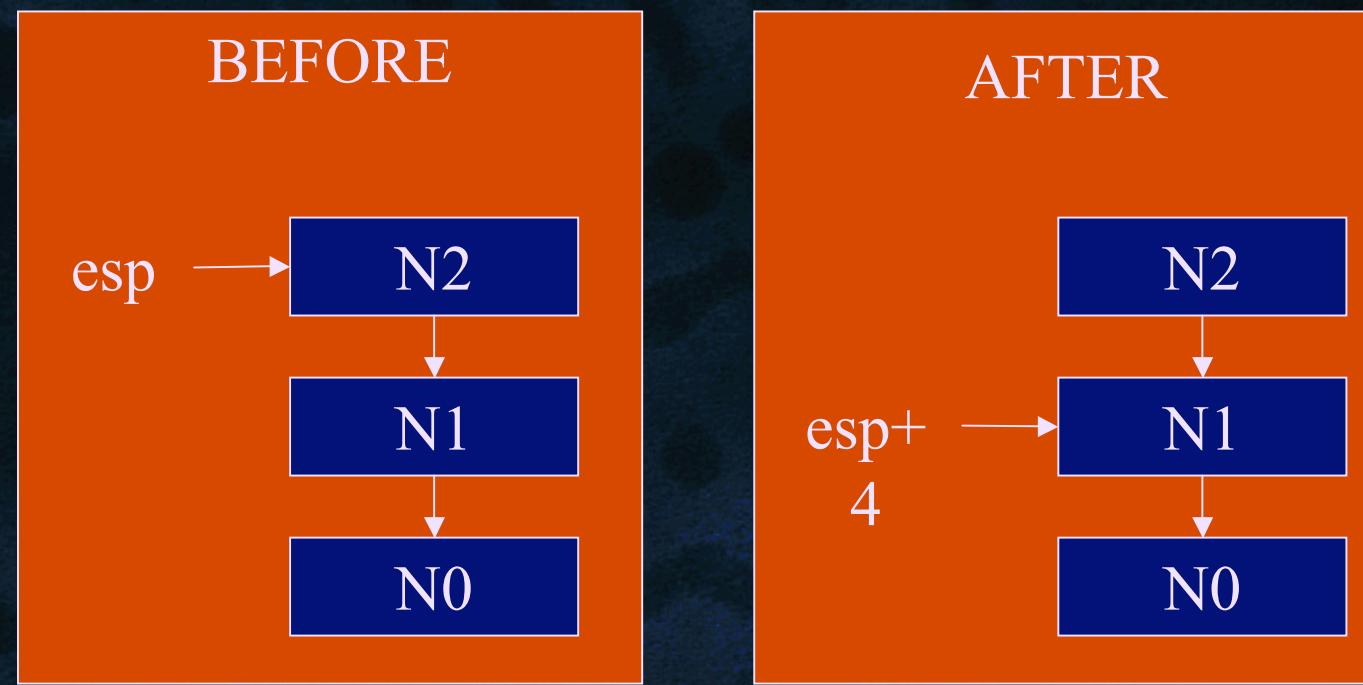
$+ : \text{STACK-LOCATION} \times \text{STACK-LOCATION} = \text{NULL}$

$+ : \text{RIC} \times \text{STACK-LOCATION} \rightarrow P(\text{STACK-LOCATION})$

- RIC+RIC
  - Equals approximation of the sum of each value in the two RICs
- Stack-location + Stack-location
  - Unable to do, since requires knowing actual addresses

# Operation: Addition

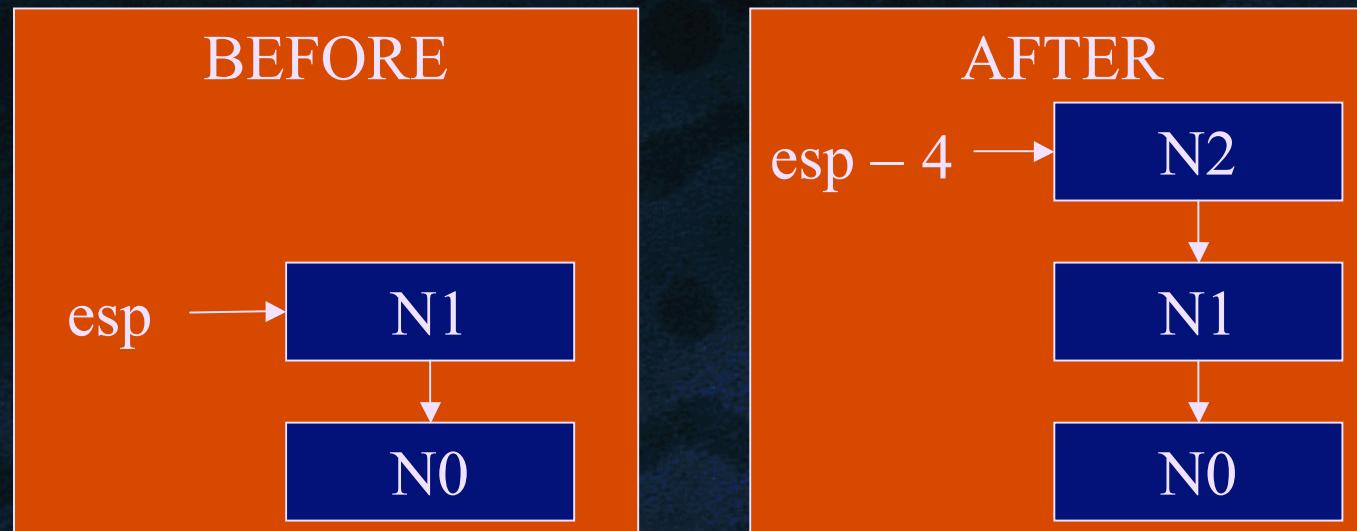
- Stack-location + RIC
  - Travel down the stack



# Operation: Subtraction

---

- RIC – RIC, Stack-location – Stack-location
  - Similar to addition
- Stack-location – RIC
  - Travel up the stack or add new nodes



# Operation: Memory Operations

---

- Push
  - Creates a new stack-location
  - Stores pushed value at stack-location
- Pop
  - Retrieves value at top of stack
  - Increments top of stack

# Operation: Memory Operations

---

- Store
  - Places a value in memory
- Load
  - Retrieves a value from memory
- Other operations defined in paper

# Interpretation Example

---

- Main
  - Pushes two values onto stack
  - Calls Max
- Max
  - Places max of parameters in eax

```
VAR1 EQU 4  
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1  
101 PUSH VAR2  
102 CALL Max  
103 ...
```

Max:

```
104 MOV eax, [esp+4]  
105 MOV ebx, [esp+8]  
106 CMP eax, ebx  
107 JG Done  
108 MOV eax, ebx
```

Done:

```
109 RET 8
```

# Interpretation Example

---

```
VAR1 EQU 4  
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1  
101 PUSH VAR2  
102 CALL Max  
103 ...
```

Max:

```
104 MOV eax, [esp+4]  
105 MOV ebx, [esp+8]  
106 CMP eax, ebx  
107 JG Done  
108 MOV eax, ebx
```

Done:

```
109 RET 8
```

STATE

```
eax = (⊤, ⊤)  
ebx = (⊤, ⊤)  
esp = (⊤, {N0})
```

N0: ⊥

# Interpretation Example

```
VAR1 EQU 4
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1
101 PUSH VAR2
102 CALL Max
103 ...
```

Max:

```
104 MOV eax, [esp+4]
105 MOV ebx, [esp+8]
106 CMP eax, ebx
107 JG Done
108 MOV eax, ebx
```

Done:

```
109 RET 8
```

STATE

```
eax = (⊤, ⊤)
ebx = (⊤, ⊤)
esp = (⊤, {N1})
```

N1: (4, ⊤)

N0: ⊥

# Interpretation Example

```
VAR1 EQU 4
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1
101 PUSH VAR2
102 CALL Max
103 ...
```

Max:

```
104 MOV eax, [esp+4]
105 MOV ebx, [esp+8]
106 CMP eax, ebx
107 JG Done
108 MOV eax, ebx
```

Done:

```
109 RET 8
```

STATE

```
eax = (⊤, ⊤)
ebx = (⊤, ⊤)
esp = (⊤, {N2})
```

N2: (2, ⊤)

N1: (4, ⊤)

N0: ⊥

# Interpretation Example

```
VAR1 EQU 4
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1
101 PUSH VAR2
102 CALL Max
```

```
103 ...
```

Max:

```
104 MOV eax, [esp+4]
105 MOV ebx, [esp+8]
106 CMP eax, ebx
107 JG Done
108 MOV eax, ebx
```

Done:

```
109 RET 8
```

STATE

$\text{eax} = (\top, \top)$   
 $\text{ebx} = (\top, \top)$   
 $\text{esp} = (\top, \{\text{N3}\})$

N3:  $(103, \top)$

N2:  $(2, \top)$

N1:  $(4, \top)$

N0:  $\perp$

# Interpretation Example

```
VAR1 EQU 4
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1
101 PUSH VAR2
102 CALL Max
103 ...
```

Max:

```
104 MOV eax, [esp+4]
105 MOV ebx, [esp+8]
106 CMP eax, ebx
107 JG Done
108 MOV eax, ebx
```

Done:

```
109 RET 8
```

STATE

eax = (2,  $\top$ )
ebx = ( $\top$ ,  $\top$ )
esp = ( $\top$ , {N3})

N3: (103,  
 $\top$ )

N2: (2,  $\top$ )

N1: (4,  $\top$ )

N0:  $\perp$

# Interpretation Example

```
VAR1 EQU 4
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1
101 PUSH VAR2
102 CALL Max
103 ...
```

Max:

```
104 MOV eax, [esp+4]
105 MOV ebx, [esp+8]
106 CMP eax, ebx
107 JG Done
108 MOV eax, ebx
```

Done:

```
109 RET 8
```

STATE

eax = (2,  $\top$ )
ebx = (4,  $\top$ )
esp = ( $\top$ , {N3})

N3: (103,  
 $\perp$ )

↓  
N2: (2,  $\top$ )

↓  
N1: (4,  $\top$ )

↓  
N0:  $\perp$

# Interpretation Example

```
VAR1 EQU 4
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1
101 PUSH VAR2
102 CALL Max
103 ...
```

Max:

```
104 MOV eax, [esp+4]
105 MOV ebx, [esp+8]
106 CMP eax, ebx
107 JG Done
108 MOV eax, ebx
```

Done:

```
109 RET 8
```

STATE

eax = (2,  $\top$ )
ebx = (4,  $\top$ )
esp = ( $\top$ , {N3})

N3: (103,  
 $\top$ )

↓  
N2: (2,  $\top$ )

↓  
N1: (4,  $\top$ )

↓  
N0:  $\perp$

# Interpretation Example

```
VAR1 EQU 4
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1
101 PUSH VAR2
102 CALL Max
103 ...
```

Max:

```
104 MOV eax, [esp+4]
105 MOV ebx, [esp+8]
106 CMP eax, ebx
107 JG Done
108 MOV eax, ebx
```

Done:

```
109 RET 8
```

STATE

eax = (2,  $\top$ )
ebx = (4,  $\top$ )
esp = ( $\top$ , {N3})

N3: (103,  
 $\top$ )

↓  
N2: (2,  $\top$ )

↓  
N1: (4,  $\top$ )

↓  
N0:  $\perp$

# Interpretation Example

```
VAR1 EQU 4
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1
101 PUSH VAR2
102 CALL Max
103 ...
```

Max:

```
104 MOV eax, [esp+4]
105 MOV ebx, [esp+8]
106 CMP eax, ebx
107 JG Done
108 MOV eax, ebx
```

Done:

```
109 RET 8
```

STATE

eax = (4,  $\top$ )
ebx = (4,  $\top$ )
esp = ( $\top$ , {N3})

N3: (103,  
 $\top$ )

N2: (2,  $\top$ )

N1: (4,  $\top$ )

N0:  $\perp$

# Interpretation Example

```
VAR1 EQU 4
VAR2 EQU 2

Main:
100 PUSH VAR1
101 PUSH VAR2
102 CALL Max
103 ...

Max:
104 MOV eax, [esp+4]
105 MOV ebx, [esp+8]
106 CMP eax, ebx
107 JG Done
108 MOV eax, ebx

Done:
109 RET 8
```

## STATE

$\text{eax} = (2[0,1]+2, \top)$   
 $\text{ebx} = (4, \top)$   
 $\text{esp} = (\top, \{\text{N0}\})$

N3: (103,  
 $\top$ )

N2: (2,  $\top$ )

N1: (4,  $\top$ )

N0:  $\perp$

# Interpretation Example

```
VAR1 EQU 4
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1
101 PUSH VAR2
102 CALL Max
103 ...
```

Max:

```
104 MOV eax, [esp+4]
105 MOV ebx, [esp+8]
106 CMP eax, ebx
107 JG Done
108 MOV eax, ebx
```

Done:

```
109 RET 8
```

STATE

$\text{eax} = (2[0,1]+2, \top)$   
 $\text{ebx} = (4, \top)$   
 $\text{esp} = (\top, \{\text{N0}\})$

N3: (103,  
 $\top$ )

N2: (2,  $\top$ )

N1: (4,  $\top$ )

N0:  $\perp$

# Talk Contents

---

Motivation

Previous work

VSA (Balakrishnan and Reps, 2004)

Abstract Stack Graph (Lakhotia and Kumar 2004)

VSA+ASG

Domain

Interpretation example

Application

Detect Call obfuscations

Implementation

Future and Conclusions

# Application

---

- Detect
  - Call replaced with equivalent instruction(s)
  - Return address manually removed from stack
- Flag as obfuscation if
  - `instruction = retn AND topOfStack.creator ≠ call`
  - `instruction = pop AND topOfStack.creator = call`
- Rules are applied to each instruction

# Replace CALL with JMP

```
VAR1 EQU 4  
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1  
101 PUSH VAR2  
102 PUSH 104  
103 JMP Max  
104 ...
```

Max:

```
105 MOV eax, [esp+4]  
106 MOV ebx, [esp+8]  
107 CMP eax, ebx  
108 JG Done  
109 MOV eax, ebx
```

Done:

```
110 RET 8
```

## STATE

```
eax = (⊤, ⊤)  
ebx = (⊤, ⊤)  
esp = (⊤, {N0})
```

N0: ⊥

# Replace CALL with JMP

```
VAR1 EQU 4  
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1  
101 PUSH VAR2  
102 PUSH 104  
103 JMP Max  
104 ...
```

Max:

```
105 MOV eax, [esp+4]  
106 MOV ebx, [esp+8]  
107 CMP eax, ebx  
108 JG Done  
109 MOV eax, ebx
```

Done:

```
110 RET 8
```

## STATE

eax = ( $\top$ ,  $\top$ )  
ebx = ( $\top$ ,  $\top$ )  
esp = ( $\top$ , {N1})

N1: (4,  $\top$ )

100

N0:  $\perp$

# Replace CALL with JMP

```
VAR1 EQU 4  
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1  
101 PUSH VAR2  
102 PUSH 104  
103 JMP Max  
104 ...
```

Max:

```
105 MOV eax, [esp+4]  
106 MOV ebx, [esp+8]  
107 CMP eax, ebx  
108 JG Done  
109 MOV eax, ebx
```

Done:

```
110 RET 8
```

## STATE

$\text{eax} = (\top, \top)$   
 $\text{ebx} = (\top, \top)$   
 $\text{esp} = (\top, \{\text{N2}\})$

N2: (2,  $\top$ )  
101

↓  
N1: (4,  $\top$ )  
100

↓  
N0:  $\perp$

# Replace CALL with JMP

```
VAR1 EQU 4  
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1  
101 PUSH VAR2  
102 PUSH 104  
103 JMP Max  
104 ...
```

Max:

```
105 MOV eax, [esp+4]  
106 MOV ebx, [esp+8]  
107 CMP eax, ebx  
108 JG Done  
109 MOV eax, ebx
```

Done:

```
110 RET 8
```

## STATE

$\text{eax} = (\top, \top)$   
 $\text{ebx} = (\top, \top)$   
 $\text{esp} = (\top, \{\text{N3}\})$

N3: (104,  
 $\top$ )

102

N2: (2,  $\top$ )  
101



N1: (4,  $\top$ )  
100



N0:  $\perp$

# Replace CALL with JMP

```
VAR1 EQU 4  
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1  
101 PUSH VAR2  
102 PUSH 104  
103 JMP Max
```

```
104 ...
```

Max:

```
105 MOV eax, [esp+4]  
106 MOV ebx, [esp+8]  
107 CMP eax, ebx  
108 JG Done  
109 MOV eax, ebx
```

Done:

```
110 RET 8
```

STATE

$\text{eax} = (\top, \top)$   
 $\text{ebx} = (\top, \top)$   
 $\text{esp} = (\top, \{\text{N3}\})$

N3: (104,  
 $\top$ )

102

N2: (2,  $\top$ )  
101



N1: (4,  $\top$ )  
100



N0:  $\perp$

# Replace CALL with JMP

```
VAR1 EQU 4  
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1  
101 PUSH VAR2  
102 PUSH 104  
103 JMP Max  
104 ...
```

Max:

```
105 MOV eax, [esp+4]  
106 MOV ebx, [esp+8]  
107 CMP eax, ebx  
108 JG Done  
109 MOV eax, ebx
```

Done:

```
110 RET 8
```

STATE

eax = (2,  $\top$ )  
ebx = ( $\top$ ,  $\top$ )  
esp = ( $\top$ , {N3})

N3: (104,  
 $\top$ )

102

N2: (2,  $\top$ )  
101



N1: (4,  $\top$ )  
100



N0:  $\perp$

# Replace CALL with JMP

```
VAR1 EQU 4  
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1  
101 PUSH VAR2  
102 PUSH 104  
103 JMP Max  
104 ...
```

Max:

```
105 MOV eax, [esp+4]  
106 MOV ebx, [esp+8]  
107 CMP eax, ebx  
108 JG Done  
109 MOV eax, ebx
```

Done:

```
110 RET 8
```

## STATE

$\text{eax} = (2, \top)$   
 $\text{ebx} = (4, \top)$   
 $\text{esp} = (\top, \{\text{N3}\})$

N3: (104,  
 $\top$ )

102

N2: (2,  $\top$ )  
101



N1: (4,  $\top$ )  
100



N0:  $\perp$

# Replace CALL with JMP

```
VAR1 EQU 4  
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1  
101 PUSH VAR2  
102 PUSH 104  
103 JMP Max  
104 ...
```

Max:

```
105 MOV eax, [esp+4]  
106 MOV ebx, [esp+8]  
107 CMP eax, ebx  
108 JG Done  
109 MOV eax, ebx
```

Done:

```
110 RET 8
```

## STATE

$\text{eax} = (2, \top)$   
 $\text{ebx} = (4, \top)$   
 $\text{esp} = (\top, \{\text{N3}\})$

N3: (104,  
 $\top$ )

102

N2: (2,  $\top$ )  
101



N1: (4,  $\top$ )  
100



N0:  $\perp$

# Replace CALL with JMP

```
VAR1 EQU 4  
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1  
101 PUSH VAR2  
102 PUSH 104  
103 JMP Max  
104 ...
```

Max:

```
105 MOV eax, [esp+4]  
106 MOV ebx, [esp+8]  
107 CMP eax, ebx  
108 JG Done  
109 MOV eax, ebx
```

Done:

```
110 RET 8
```

## STATE

$\text{eax} = (2, \top)$   
 $\text{ebx} = (4, \top)$   
 $\text{esp} = (\top, \{\text{N3}\})$

N3: (104,  
 $\top$ )

102

N2: (2,  $\top$ )  
101



N1: (4,  $\top$ )  
100



N0:  $\perp$

# Replace CALL with JMP

```
VAR1 EQU 4  
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1  
101 PUSH VAR2  
102 PUSH 104  
103 JMP Max  
104 ...
```

Max:

```
105 MOV eax, [esp+4]  
106 MOV ebx, [esp+8]  
107 CMP eax, ebx  
108 JG Done  
109 MOV eax, ebx
```

Done:

```
110 RET 8
```

## STATE

$\text{eax} = (4, \top)$   
 $\text{ebx} = (4, \top)$   
 $\text{esp} = (\top, \{\text{N3}\})$

N3: (104,  
 $\top$ )

102

N2: (2,  $\top$ )  
101



N1: (4,  $\top$ )  
100



N0:  $\perp$

# Replace CALL with JMP

```
VAR1 EQU 4  
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1  
101 PUSH VAR2  
102 PUSH 104  
103 JMP Max  
104 ...
```

Max:

```
105 MOV eax, [esp+4]  
106 MOV ebx, [esp+8]  
107 CMP eax, ebx  
108 JG Done  
109 MOV eax, ebx
```

Done:

```
110 RET 8
```

## STATE

eax =  $(2[0,1]+2, \top)$   
ebx =  $(4, \top)$   
esp =  $(\top, \{N0\})$

N3:  $(104, \top)$

102

N2:  $(2, \top)$

101

N1:  $(4, \top)$

100

N0:  $\perp$

# Replace CALL with JMP

```
VAR1 EQU 4  
VAR2 EQU 2
```

Main:

```
100 PUSH VAR1  
101 PUSH VAR2  
102 PUSH 104  
103 JMP Max  
104 ...
```

Max:

```
105 MOV eax, [esp+4]  
106 MOV ebx, [esp+8]  
107 CMP eax, ebx  
108 JG Done  
109 MOV eax, ebx
```

Done:

```
110 RET 8
```

STATE

$\text{eax} = (2[0,1]+2, \top)$   
 $\text{ebx} = (4, \top)$   
 $\text{esp} = (\top, \{\text{N0}\})$

Return address placed  
on stack by push

N3: (104,  
 $\top$ )

102

N2: (2,  $\top$ )  
101

↓

N1: (4,  $\top$ )  
100

↓

N0:  $\perp$

# Removal of Return Address

Main:

```
100  PUSH   4  
101  PUSH   2  
102  CALL    Max  
103  ...
```

Max:

```
104  MOV     eax,  [esp+4]  
105  MOV     ebx,  [esp+8]  
106  CMP     eax,  ebx  
107  JG      Done  
108  MOV     eax,  ebx
```

Done:

```
109  POP     ebx  
110  ADD     esp,  8  
111  JMP     ebx
```

STATE

$\text{eax} = (\top, \top)$   
 $\text{ebx} = (\top, \top)$   
 $\text{esp} = (\top, \{\text{N0}\})$

N0:  $\perp$

# Removal of Return Address

Main:

```
100 PUSH 4  
101 PUSH 2  
102 CALL Max  
103 ...
```

Max:

```
104 MOV eax, [esp+4]  
105 MOV ebx, [esp+8]  
106 CMP eax, ebx  
107 JG Done  
108 MOV eax, ebx
```

Done:

```
109 POP ebx  
110 ADD esp, 8  
111 JMP ebx
```

STATE

$\text{eax} = (\top, \top)$   
 $\text{ebx} = (\top, \top)$   
 $\text{esp} = (\top, \{\text{N1}\})$

N1:  $(4, \top)$

100

N0:  $\perp$

# Removal of Return Address

Main:

```
100 PUSH 4
101 PUSH 2
102 CALL Max
103 ...
```

Max:

```
104 MOV eax, [esp+4]
105 MOV ebx, [esp+8]
106 CMP eax, ebx
107 JG Done
108 MOV eax, ebx
```

Done:

```
109 POP ebx
110 ADD esp, 8
111 JMP ebx
```

STATE

$\text{eax} = (\top, \top)$   
 $\text{ebx} = (\top, \top)$   
 $\text{esp} = (\top, \{\text{N2}\})$

N2: (2,  $\top$ )  
101

↓  
N1: (4,  $\top$ )  
100

↓  
N0:  $\perp$

# Removal of Return Address

Main:

```
100  PUSH   4
101  PUSH   2
102  CALL    Max
103  ...
```

Max:

```
104  MOV     eax, [esp+4]
105  MOV     ebx, [esp+8]
106  CMP     eax, ebx
107  JG      Done
108  MOV     eax, ebx
```

Done:

```
109  POP     ebx
110  ADD     esp, 8
111  JMP     ebx
```

STATE

$\text{eax} = (\top, \top)$   
 $\text{ebx} = (\top, \top)$   
 $\text{esp} = (\top, \{\text{N3}\})$

N3: (103,  
       $\top$ )

102

N2: (2,  $\top$ )  
101

N1: (4,  $\top$ )  
100

N0:  $\perp$

# Removal of Return Address

Main:

```
100  PUSH   4  
101  PUSH   2  
102  CALL    Max  
103  ...
```

Max:

```
104  MOV     eax, [esp+4]  
105  MOV     ebx, [esp+8]  
106  CMP     eax, ebx  
107  JG      Done  
108  MOV     eax, ebx
```

Done:

```
109  POP     ebx  
110  ADD     esp, 8  
111  JMP     ebx
```

STATE

$\text{eax} = (2, \top)$   
 $\text{ebx} = (\top, \top)$   
 $\text{esp} = (\top, \{\text{N3}\})$

N3: (103,  
 $\top$ )

102

N2: (2,  $\top$ )  
101

N1: (4,  $\top$ )  
100

N0:  $\perp$

# Removal of Return Address

Main:

```
100 PUSH 4
101 PUSH 2
102 CALL Max
103 ...
```

Max:

```
104 MOV eax, [esp+4]
105 MOV ebx, [esp+8]
106 CMP eax, ebx
107 JG Done
108 MOV eax, ebx
```

Done:

```
109 POP ebx
110 ADD esp, 8
111 JMP ebx
```

STATE

$\text{eax} = (2, \top)$   
 $\text{ebx} = (4, \top)$   
 $\text{esp} = (\top, \{\text{N3}\})$

N3: (103,  
 $\top$ )

102

N2: (2,  $\top$ )  
101

N1: (4,  $\top$ )  
100

N0:  $\perp$

# Removal of Return Address

Main:

```
100 PUSH 4
101 PUSH 2
102 CALL Max
103 ...
```

Max:

```
104 MOV eax, [esp+4]
105 MOV ebx, [esp+8]
106 CMP eax, ebx
107 JG Done
108 MOV eax, ebx
```

Done:

```
109 POP ebx
110 ADD esp, 8
111 JMP ebx
```

STATE

$\text{eax} = (2, \top)$   
 $\text{ebx} = (4, \top)$   
 $\text{esp} = (\top, \{\text{N3}\})$

N3: (103,  
 $\top$ )

102

N2: (2,  $\top$ )  
101

N1: (4,  $\top$ )  
100

N0:  $\perp$

# Removal of Return Address

Main:

```
100 PUSH 4
101 PUSH 2
102 CALL Max
103 ...
```

Max:

```
104 MOV eax, [esp+4]
105 MOV ebx, [esp+8]
106 CMP eax, ebx
107 JG Done
108 MOV eax, ebx
```

Done:

```
109 POP ebx
110 ADD esp, 8
111 JMP ebx
```

STATE

$\text{eax} = (2, \top)$   
 $\text{ebx} = (4, \top)$   
 $\text{esp} = (\top, \{\text{N3}\})$

N3: (103,  
 $\top$ )

102

N2: (2,  $\top$ )  
101



N1: (4,  $\top$ )  
100



N0:  $\perp$

# Removal of Return Address

Main:

```
100  PUSH   4  
101  PUSH   2  
102  CALL    Max  
103  ...
```

Max:

```
104  MOV     eax, [esp+4]  
105  MOV     ebx, [esp+8]  
106  CMP     eax, ebx  
107  JG      Done  
108  MOV     eax, ebx
```

Done:

```
109  POP     ebx  
110  ADD     esp, 8  
111  JMP     ebx
```

STATE

$\text{eax} = (4, \top)$   
 $\text{ebx} = (4, \top)$   
 $\text{esp} = (\top, \{\text{N3}\})$

N3: (103,  
 $\top$ )

102

N2: (2,  $\top$ )  
101

N1: (4,  $\top$ )  
100

N0:  $\perp$

# Removal of Return Address

Main:

```
100 PUSH 4
101 PUSH 2
102 CALL Max
103 ...
```

Max:

```
104 MOV eax, [esp+4]
105 MOV ebx, [esp+8]
106 CMP eax, ebx
107 JG Done
108 MOV eax, ebx
```

Done:

```
109 POP ebx
110 ADD esp, 8
111 JMP ebx
```

STATE

$\text{eax} = (2[0,1]+2, \top)$   
 $\text{ebx} = (4, \top)$   
 $\text{esp} = (\top, \{\text{N2}\})$

N3: (103,  
 $\top$ )

102

N2: (2,  $\top$ )  
101

↓  
N1: (4,  $\top$ )  
100

↓  
N0:  $\perp$

# Removal of Return Address

Main:

```
100 PUSH 4  
101 PUSH 2  
102 CALL Max
```

103 ...

Max:

```
104 MOV eax, [esp+4]  
105 MOV ebx, [e  
106 CMP eax, ebx  
107 JG Done  
108 MOV eax, ebx
```

Done:

```
109 POP ebx  
110 ADD esp, 8  
111 JMP ebx
```

STATE

$\text{eax} = (2[0,1]+2, \top)$   
 $\text{ebx} = (4, \top)$   
 $\text{esp} = (\top, \{\text{N2}\})$

Return address is popped off stack

N3: (103,  $\top$ )

102

N2: (2,  $\top$ )  
101

↓

N1: (4,  $\top$ )  
100

↓

N0:  $\perp$

# Talk Contents

---

Motivation

Previous work

VSA (Balakrishnan and Reps, 2004)

Abstract Stack Graph (Lakhotia and Kumar 2004)

VSA+ASG

Domain

Interpretation example

Application

Detect Call obfuscations

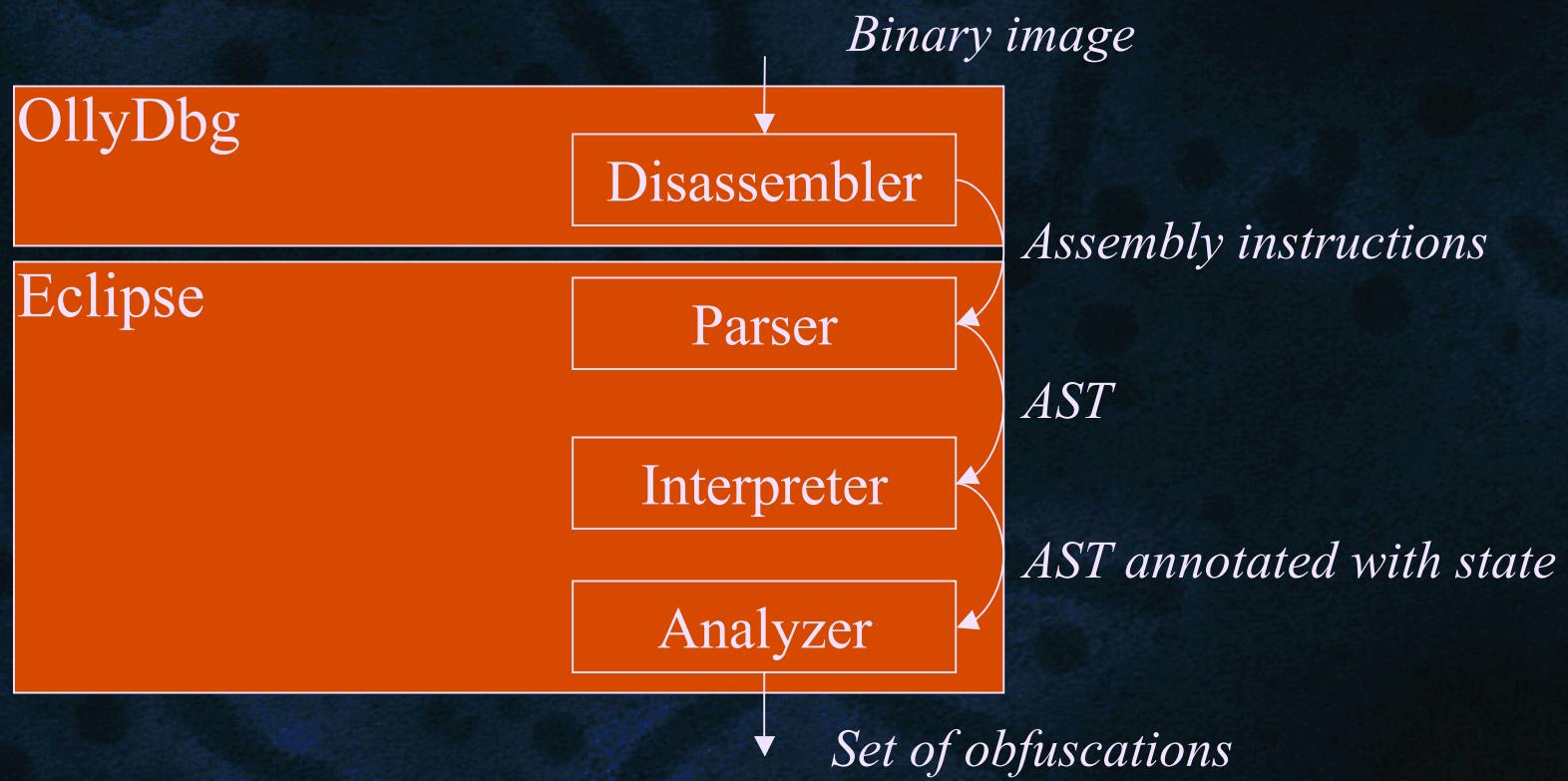
Implementation

Future and Conclusions

# Prototype

---

- Implemented on Eclipse platform



Obfuscation Detection - input.asm - Eclipse Platform

File Edit Navigate Search Project Run Window Help

input.asm X

```
401000h PUSH 4
401002h PUSH 2
401004h CALL 401010h
401009h PUSH 0
40100bh CALL KERNEL32.ExitProcess
401010h MOV eax, dword ptr ss:[esp+4]
401014h MOV ebx, dword ptr ss:[esp+8]
401018h CMP eax, ebx
40101ah JG 40101eh
40101ch MOV eax, ebx
40101eh POP ebx
40101fh ADD esp, 8
401022h JMP ebx
40000000h UNDEF eax
40000001h RET 12
40000005h UNDEF eax
40000006h RET 24
40000010h UNDEF eax
40000011h RET 4
10000000h UNDEF eax
10000001h RET 4
```

Messages Valid Call-Return Sites Obfuscated Calls Obfuscated Returns Associated Instructions

Line Instruction

11 POP ebx

Registers X

Register	Value
EAX	(T, T)
EBX	(T, T)
ECX	(T, T)
EDX	(T, T)
ESP	(T, {N1})
EBP	(T, T)
ESI	(T, T)
EDI	(T, T)
CS	(T, T)
DS	(T, T)
ES	(T, T)

Stack X

```
N1
(4, T)
↓
N0
(T, T)
```

Obfuscations shown on ruler

Register and stack contents

Obfuscations shown as list

The screenshot displays the Eclipse Platform interface for obfuscation detection on the assembly file 'input.asm'. The assembly code window shows various instructions like PUSH, CALL, and RET. Annotations highlight specific instructions (e.g., POP ebx) with orange boxes containing text: 'Obfuscations shown on ruler', 'Register and stack contents', and 'Obfuscations shown as list'. The registers window lists all general-purpose registers with their current values. The stack window shows the state of the stack, with N1 having a value of (4, T) and N0 having a value of (T, T). The bottom part of the interface shows tabs for 'Messages', 'Valid Call-Return Sites', 'Obfuscated Calls', 'Obfuscated Returns' (which is currently selected), and 'Associated Instructions'. A table below the tabs shows the line number 11 and the instruction POP ebx.

Obfuscation Detection - input.asm - Eclipse Platform

File Edit Navigate Search Project Run Window Help

input.asm X

```
401000h PUSH 4
401002h PUSH 2
401004h CALL 401010h
401009h PUSH 0
C 40100bh CALL KERNEL32.ExitProcess
401010h MOV eax, dword ptr ss:[esp+4]
401014h MOV ebx, dword ptr ss:[esp+8]
401018h CMP eax, ebx
40101ah JG 40101eh
40101ch MOV eax, ebx
40101eh POP ebx
40101fh ADD esp, 8
401022h JMP ebx
40000000h UNDEF eax
40000001h RET 12
40000005h UNDEF eax
40000006h RET 24
40000010h UNDEF eax
40000011h RET 4
10000000h UNDEF eax
10000001h RET 4
```

Messages Valid Call-Return Sites Obfuscated Calls Obfuscated Returns Associated Instructions

Line	Instruction
11	POP ebx

Registers X

Register	Value
EAX	(T, T)
EBX	(T, T)
ECX	(T, T)
EDX	(T, T)
ESP	(T, {N1})
EBP	(T, T)
ESI	(T, T)
EDI	(T, T)
CS	(T, T)
DS	(T, T)
ES	(T, T)

Stack X

```
graph TD; N1["N1  
(4, T)"] --> N0["N0  
(T, T)"]
```

Obfuscation Detection - input.asm - Eclipse Platform

File Edit Navigate Search Project Run Window Help

input.asm X Registers X Stack X

401000h PUSH 4  
401002h PUSH 2  
401004h CALL 401010h  
401009h PUSH 0  
C 40100bh CALL KERNEL32.ExitProcess  
401010h MOV eax, dword ptr ss:[esp+4]  
401014h MOV ebx, dword ptr ss:[esp+8]  
401018h CMP eax, ebx  
40101ah JG 40101eh  
40101ch MOV eax, ebx  
40101eh POP ebx  
40101fh ADD esp, 8  
401022h JMP ebx  
40000000h UNDEF eax  
40000001h RET 12  
40000005h UNDEF eax  
40000006h RET 24  
40000010h UNDEF eax  
40000011h RET 4  
10000000h UNDEF eax  
10000001h RET 4

Messages Valid Call-Return Sites Obfuscated Calls Obfuscated Returns Associated Instructions

Line Instruction

11 POP ebx

Registers X

Register	Value
EAX	(T, T)
EBX	(T, T)
ECX	(T, T)
EDX	(T, T)
ESP	(T, {N2})
EBP	(T, T)
ESI	(T, T)
EDI	(T, T)
CS	(T, T)
DS	(T, T)
ES	(T, T)

Stack X

```
graph TD; N2["N2<br>(2, T)"] --> N1["N1<br>(4, T)"]; N1 --> NO["NO<br>(T, T)"]
```

Obfuscation Detection - input.asm - Eclipse Platform

File Edit Navigate Search Project Run Window Help

input.asm X Registers X Stack X

401000h PUSH 4  
401002h PUSH 2  
401004h CALL 401010h  
401009h PUSH 0  
C 40100bh CALL KERNEL32.ExitProcess  
401010h MOV eax, dword ptr ss:[esp+4]  
401014h MOV ebx, dword ptr ss:[esp+8]  
401018h CMP eax, ebx  
40101ah JG 40101eh  
40101ch MOV eax, ebx  
40101eh POP ebx  
40101fh ADD esp, 8  
401022h JMP ebx  
40000000h UNDEF eax  
40000001h RET 12  
40000005h UNDEF eax  
40000006h RET 24  
40000010h UNDEF eax  
40000011h RET 4  
10000000h UNDEF eax  
10000001h RET 4

Messages Valid Call-Return Sites Obfuscated Calls Obfuscated Returns Associated Instructions

Line Instruction

11 POP ebx

Registers X

Register	Value
EAX	(T, T)
EBX	(T, T)
ECX	(T, T)
EDX	(T, T)
ESP	(T, {N3})
EBP	(T, T)
ESI	(T, T)
EDI	(T, T)
CS	(T, T)
DS	(T, T)
ES	(T, T)

Stack X

```
graph TD; N3["N3  
(4198409, T)"] --> N2["N2  
(2, T)"]; N2 --> N1["N1  
(4, T)"]; N1 --> N0["N0  
(T, T)"]
```

Obfuscation Detection - input.asm - Eclipse Platform

File Edit Navigate Search Project Run Window Help

input.asm X Registers X Stack X

401000h PUSH 4  
401002h PUSH 2  
401004h CALL 401010h  
401009h PUSH 0  
C 40100bh CALL KERNEL32.ExitProcess  
401010h MOV eax, dword ptr ss:[esp+4]  
401014h MOV ebx, dword ptr ss:[esp+8]  
401018h CMP eax, ebx  
40101ah JG 40101eh  
40101ch MOV eax, ebx  
40101eh POP ebx  
40101fh ADD esp, 8  
401022h JMP ebx  
40000000h UNDEF eax  
40000001h RET 12  
40000005h UNDEF eax  
40000006h RET 24  
40000010h UNDEF eax  
40000011h RET 4  
10000000h UNDEF eax  
10000001h RET 4

Messages Valid Call-Return Sites Obfuscated Calls Obfuscated Returns Associated Instructions

Line Instruction

11 POP ebx

Registers X

Register	Value
EAX	(2, T)
EBX	(T, T)
ECX	(T, T)
EDX	(T, T)
ESP	(T, {N3})
EBP	(T, T)
ESI	(T, T)
EDI	(T, T)
CS	(T, T)
DS	(T, T)
ES	(T, T)

Stack X

```
graph TD; N3["N3  
(4198409, T)"] --> N2["N2  
(2, T)"]; N2 --> N1["N1  
(4, T)"]; N1 --> N0["N0  
(T, T)"]
```

Obfuscation Detection - input.asm - Eclipse Platform

File Edit Navigate Search Project Run Window Help

input.asm X Registers X Stack X

401000h PUSH 4  
401002h PUSH 2  
401004h CALL 401010h  
401009h PUSH 0  
C 40100bh CALL KERNEL32.ExitProcess  
401010h MOV eax, dword ptr ss:[esp+4]  
401014h MOV ebx, dword ptr ss:[esp+8]  
401018h CMP eax, ebx  
40101ah JG 40101eh  
40101ch MOV eax, ebx  
40101eh POP ebx  
40101fh ADD esp, 8  
401022h JMP ebx  
40000000h UNDEF eax  
40000001h RET 12  
40000005h UNDEF eax  
40000006h RET 24  
40000010h UNDEF eax  
40000011h RET 4  
10000000h UNDEF eax  
10000001h RET 4

Messages Valid Call-Return Sites Obfuscated Calls Obfuscated Returns Associated Instructions

Line Instruction

11 POP ebx

Registers X

Register	Value
EAX	(2, T)
EBX	(4, T)
ECX	(T, T)
EDX	(T, T)
ESP	(T, {N3})
EBP	(T, T)
ESI	(T, T)
EDI	(T, T)
CS	(T, T)
DS	(T, T)
ES	(T, T)

Stack X

```
graph TD; N3["N3  
(4198409, T)"] --> N2["N2  
(2, T)"]; N2 --> N1["N1  
(4, T)"]; N1 --> N0["N0  
(T, T)"]
```

Obfuscation Detection - input.asm - Eclipse Platform

File Edit Navigate Search Project Run Window Help

input.asm X Registers X Stack X

401000h PUSH 4  
401002h PUSH 2  
401004h CALL 401010h  
401009h PUSH 0  
C 40100bh CALL KERNEL32.ExitProcess  
401010h MOV eax, dword ptr ss:[esp+4]  
401014h MOV ebx, dword ptr ss:[esp+8]  
401018h CMP eax, ebx  
40101ah JG 40101eh  
40101ch MOV eax, ebx  
40101eh POP ebx  
40101fh ADD esp, 8  
401022h JMP ebx  
40000000h UNDEF eax  
40000001h RET 12  
40000005h UNDEF eax  
40000006h RET 24  
40000010h UNDEF eax  
40000011h RET 4  
10000000h UNDEF eax  
10000001h RET 4

Messages Valid Call-Return Sites Obfuscated Calls Obfuscated Returns Associated Instructions

Line Instruction

11 POP ebx

Registers X

Register	Value
EAX	(2, T)
EBX	(4, T)
ECX	(T, T)
EDX	(T, T)
ESP	(T, {N3})
EBP	(T, T)
ESI	(T, T)
EDI	(T, T)
CS	(T, T)
DS	(T, T)
ES	(T, T)

Stack X

```
graph TD; N3["N3  
(4198409, T)"] --> N2["N2  
(2, T)"]; N2 --> N1["N1  
(4, T)"]; N1 --> N0["N0  
(T, T)"]
```

Obfuscation Detection - input.asm - Eclipse Platform

File Edit Navigate Search Project Run Window Help

input.asm X Registers X Stack X

401000h PUSH 4  
401002h PUSH 2  
401004h CALL 401010h  
401009h PUSH 0  
C 40100bh CALL KERNEL32.ExitProcess  
401010h MOV eax, dword ptr ss:[esp+4]  
401014h MOV ebx, dword ptr ss:[esp+8]  
401018h CMP eax, ebx  
40101ah JG 40101eh  
40101ch MOV eax, ebx  
40101eh POP ebx  
40101fh ADD esp, 8  
401022h JMP ebx  
40000000h UNDEF eax  
40000001h RET 12  
40000005h UNDEF eax  
40000006h RET 24  
40000010h UNDEF eax  
40000011h RET 4  
10000000h UNDEF eax  
10000001h RET 4

Messages Valid Call-Return Sites Obfuscated Calls Obfuscated Returns Associated Instructions

Line Instruction

11 POP ebx

Registers X

Register	Value
EAX	(2, T)
EBX	(4, T)
ECX	(T, T)
EDX	(T, T)
ESP	(T, {N3})
EBP	(T, T)
ESI	(T, T)
EDI	(T, T)
CS	(T, T)
DS	(T, T)
ES	(T, T)

Stack X

```
graph TD; N3["N3  
(4198409, T)"] --> N2["N2  
(2, T)"]; N2 --> N1["N1  
(4, T)"]; N1 --> N0["N0  
(T, T)"]
```

Obfuscation Detection - input.asm - Eclipse Platform

File Edit Navigate Search Project Run Window Help

input.asm X Registers X Stack X

401000h PUSH 4  
401002h PUSH 2  
401004h CALL 401010h  
401009h PUSH 0  
C 40100bh CALL KERNEL32.ExitProcess  
401010h MOV eax, dword ptr ss:[esp+4]  
401014h MOV ebx, dword ptr ss:[esp+8]  
401018h CMP eax, ebx  
40101ah JG 40101eh  
40101ch MOV eax, ebx  
40101eh POP ebx  
40101fh ADD esp, 8  
401022h JMP ebx  
40000000h UNDEF eax  
40000001h RET 12  
40000005h UNDEF eax  
40000006h RET 24  
40000010h UNDEF eax  
40000011h RET 4  
10000000h UNDEF eax  
10000001h RET 4

Messages Valid Call-Return Sites Obfuscated Calls Obfuscated Returns Associated Instructions

Line Instruction

11 POP ebx

Registers X

Register	Value
EAX	(4, T)
EBX	(4, T)
ECX	(T, T)
EDX	(T, T)
ESP	(T, {N3})
EBP	(T, T)
ESI	(T, T)
EDI	(T, T)
CS	(T, T)
DS	(T, T)
ES	(T, T)

Stack X

```
graph TD; N3["N3  
(4198409, T)"] --> N2["N2  
(2, T)"]; N2 --> N1["N1  
(4, T)"]; N1 --> N0["N0  
(T, T)"]
```

Obfuscation Detection - input.asm - Eclipse Platform

File Edit Navigate Search Project Run Window Help

input.asm X

```
401000h PUSH 4
401002h PUSH 2
401004h CALL 401010h
401009h PUSH 0
C 40100bh CALL KERNEL32.ExitProcess
401010h MOV eax, dword ptr ss:[esp+4]
401014h MOV ebx, dword ptr ss:[esp+8]
401018h CMP eax, ebx
40101ah JG 40101eh
40101ch MOV eax, ebx
40101eh POP ebx
40101fh ADD esp, 8
401022h JMP ebx
40000000h UNDEF eax
40000001h RET 12
40000005h UNDEF eax
40000006h RET 24
40000010h UNDEF eax
40000011h RET 4
10000000h UNDEF eax
10000001h RET 4
```

Messages Valid Call-Return Sites Obfuscated Calls Obfuscated Returns Associated Instructions

Line	Instruction
11	POP ebx

Registers X

Register	Value
EAX	(1[2,4]+0, T)
EBX	(4198409, T)
ECX	(T, T)
EDX	(T, T)
ESP	(T, {N2})
EBP	(T, T)
ESI	(T, T)
EDI	(T, T)
CS	(T, T)
DS	(T, T)
ES	(T, T)

Stack X

```
graph TD; N2["N2  
(2, T)"] --> N1["N1  
(4, T)"]; N1 --> NO["NO  
(T, T)"]
```

Obfuscation Detection - input.asm - Eclipse Platform

File Edit Navigate Search Project Run Window Help

input.asm X

```
401000h PUSH 4
401002h PUSH 2
401004h CALL 401010h
401009h PUSH 0
C 40100bh CALL KERNEL32.ExitProcess
401010h MOV eax, dword ptr ss:[esp+4]
401014h MOV ebx, dword ptr ss:[esp+8]
401018h CMP eax, ebx
40101ah JG 40101eh
40101ch MOV eax, ebx
40101eh POP ebx
40101fh ADD esp, 8
401022h JMP ebx
40000000h UNDEF eax
40000001h RET 12
40000005h UNDEF eax
40000006h RET 24
40000010h UNDEF eax
40000011h RET 4
10000000h UNDEF eax
10000001h RET 4
```

Registers X

Register	Value
EAX	(1[2,4]+0, T)
EBX	(4198409, T)
ECX	(T, T)
EDX	(T, T)
ESP	(T, {NO})
EBP	(T, T)
ESI	(T, T)
EDI	(T, T)
CS	(T, T)
DS	(T, T)
ES	(T, T)

Stack X

NO
(T, T)

Messages | Valid Call-Return Sites | Obfuscated Calls | Obfuscated Returns X | Associated Instructions

Line Instruction

11 POP ebx

Obfuscation Detection - input.asm - Eclipse Platform

File Edit Navigate Search Project Run Window Help

input.asm X Registers X Stack X

Registers

Register	Value
EAX	(1[2,4]+0, T)
EBX	(4198409, T)
ECX	(T, T)
EDX	(T, T)
ESP	(T, {NO})
EBP	(T, T)
ESI	(T, T)
EDI	(T, T)
CS	(T, T)
DS	(T, T)
ES	(T, T)

Stack

NO
(T, T)

input.asm

```
401000h PUSH 4
401002h PUSH 2
401004h CALL 401010h
401009h PUSH 0
C 40100bh CALL KERNEL32.ExitProcess
401010h MOV eax, dword ptr ss:[esp+4]
401014h MOV ebx, dword ptr ss:[esp+8]
401018h CMP eax, ebx
40101ah JG 40101eh
40101ch MOV eax, ebx
40101eh POP ebx
40101fh ADD esp, 8
401022h JMP ebx
40000000h UNDEF eax
40000001h RET 12
40000005h UNDEF eax
40000006h RET 24
40000010h UNDEF eax
40000011h RET 4
10000000h UNDEF eax
10000001h RET 4
```

Messages Valid Call-Return Sites Obfuscated Calls Obfuscated Returns Associated Instructions

Line	Instruction
11	POP ebx

Obfuscation Detection - input.asm - Eclipse Platform

File Edit Navigate Search Project Run Window Help

input.asm X

```
401000h PUSH 4
401002h PUSH 2
401004h CALL 401010h
401009h PUSH 0
40100bh CALL KERNEL32.ExitProcess
401010h MOV eax, dword ptr ss:[esp+4]
401014h MOV ebx, dword ptr ss:[esp+8]
401018h CMP eax, ebx
40101ah JG 40101eh
40101ch MOV eax, ebx
40101eh POP ebx
40101fh ADD esp, 8
401022h JMP ebx
40000000h UNDEF eax
40000001h RET 12
40000005h UNDEF eax
40000006h RET 24
40000010h UNDEF eax
40000011h RET 4
10000000h UNDEF eax
10000001h RET 4
```

Messages Valid Call-Return Sites Obfuscated Calls Obfuscated Returns Associated Instructions

Line	Instruction
11	POP ebx

Registers X

Register	Value
EAX	(1[2,4]+0, T)
EBX	(4198409, T)
ECX	(T, T)
EDX	(T, T)
ESP	(T, {N5})
EBP	(T, T)
ESI	(T, T)
EDI	(T, T)
CS	(T, T)
DS	(T, T)
ES	(T, T)

Stack X

```
graph TD; N5["N5  
(0, T)"] --> N0["N0  
(T, T)"]
```

Obfuscation Detection - input.asm - Eclipse Platform

File Edit Navigate Search Project Run Window Help

input.asm X

```
401000h PUSH 4
401002h PUSH 2
401004h CALL 401010h
401009h PUSH 0
C 40100bh CALL KERNEL32.ExitProcess
401010h MOV eax, dword ptr ss:[esp+4]
401014h MOV ebx, dword ptr ss:[esp+8]
401018h CMP eax, ebx
40101ah JG 40101eh
40101ch MOV eax, ebx
40101eh POP ebx
40101fh ADD esp, 8
401022h JMP ebx
40000000h UNDEF eax
40000001h RET 12
40000005h UNDEF eax
40000006h RET 24
40000010h UNDEF eax
40000011h RET 4
10000000h UNDEF eax
10000001h RET 4
```

Messages Valid Call-Return Sites Obfuscated Calls Obfuscated Returns Associated Instructions

Line	Instruction
11	POP ebx

Registers X

Register	Value
EAX	(1[2,4]+0, T)
EBX	(4198409, T)
ECX	(T, T)
EDX	(T, T)
ESP	(T, {N7})
EBP	(T, T)
ESI	(T, T)
EDI	(T, T)
CS	(T, T)
DS	(T, T)
ES	(T, T)

Stack X

```
graph TD; N7["N7  
(4198416, T)"] --> N5["N5  
(0, T)"]; N5 --> NO["NO  
(T, T)"]
```

Obfuscation Detection - input.asm - Eclipse Platform

File Edit Navigate Search Project Run Window Help

input.asm X

```
401000h PUSH 402000h
401005h CALL KERNEL32.SetCurrentDirectoryA
40100ah PUSH 40200dh
40100fh PUSH 402005h
401014h PUSH 40101fh
401019h PUSH KERNEL32.FindFirstFileA
40101eh RET
40101fh CMP eax, -1
401022h JE 401054h
401024h MOV dword ptr ds:[4202827], eax
401029h PUSH 402039h
40102eh PUSH 401039h
401033h PUSH KERNEL32.DeleteFileA
401038h RET
401039h PUSH 40200dh
40103eh PUSH dword ptr ds:[4202827]
401044h PUSH 40104fh
401049h PUSH KERNEL32.FindNextFileA
40104eh RET
40104fh CMP eax, 0
401052h JNZ 401029h
401054h PUSH 0
401056h CALL KERNEL32.ExitProcess
```

Messages Valid Call-Return Sites Obfuscated Calls Obfuscated Returns Associated Instructions

Line	Instruction
7	RET
14	RET
19	RET
67	RET 8
69	RET 4
71	RET 8

Registers X

Register	Value
ESI	(T, T)
EDI	(T, T)
CS	(T, T)
DS	(T, T)
ES	(T, T)

Stack X

```
graph TD; N5["N5  
(4198431, T)"] --> N4["N4  
(4202501, T)"]; N4 --> N3["N3  
(4202509, T)"]; N3 --> NO["NO  
(T, T)"]
```

Pushes return address and function address

The screenshot shows the Eclipse Platform interface with the 'Obfuscation Detection' plugin. The main window displays assembly code for 'input.asm'. A specific instruction at address 401019h, which pushes the address of 'KERNEL32.FindFirstFileA' onto the stack, is highlighted with a green circle. A callout box with a red background and white text points to this instruction with the text 'Pushes return address and function address'. The assembly code also includes calls to 'GetCurrentDirectoryA', 'DeleteFileA', and 'ExitProcess'. The 'Registers' view shows all general-purpose registers (ESI, EDI, CS, DS, ES) containing '(T, T)' values. The 'Stack' view shows a call stack starting from address 4198431, passing through 4202501, and ending at 4202509, all associated with '(T, T)' values. The bottom navigation bar includes tabs for 'Messages', 'Valid Call-Return Sites', 'Obfuscated Calls' (which is currently selected), 'Obfuscated Returns', and 'Associated Instructions'. A table below the assembly code lists various RET instructions with their respective line numbers and values.

# Talk Contents

---

Motivation

Previous work

VSA (Balakrishnan and Reps, 2004)

Abstract Stack Graph (Lakhotia and Kumar 2004)

VSA+ASG

Domain

Interpretation example

Application

Detect Call obfuscations

Implementation

Future and Conclusions

# Prototype Limitations

---

- More Memory Support
  - mov [eax], ebx
  - GetModuleHandle/GetProcAddress

```
PUSH offset DLL_NAME      ; "kernel32.dll"
CALL GetModuleHandleA

PUSH offset PROC_NAME     ; "ExitProcess"
PUSH eax
CALL GetProcAddress

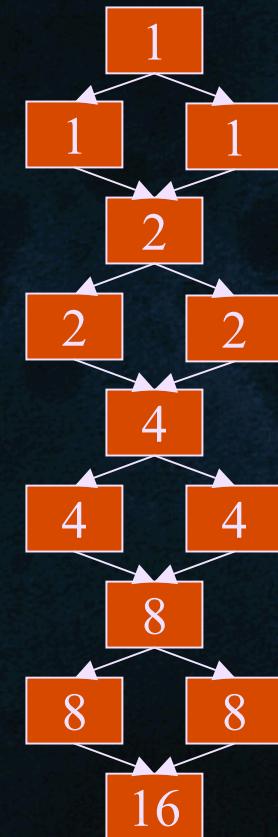
PUSH 0
CALL eax
```

# Prototype Limitations

---

- Efficiency
  - Many paths to each instruction
  - More efficient algorithms needed

CFG with branches



# General Limitations

---

- More RIC operations
  - Too many undefined RICs
- What about obfuscated call–obfuscated return pairs
  - Not a problem with system calls

# General Limitations

---

- Structure Exception Handling

Main:

```
PUSH  offset Handler      ; set up exception handler
PUSH  dword ptr FS:[0]
MOV   FS:[0], esp
MOV   edx, 0              ; force an exception
MOV   eax, 0
DIV   eax                 ; (edx / eax)
PUSH  0
CALL  ExitProcess
```

Handler:

```
...                      ; real code goes here
```

# Conclusion

---

- Method
  - Combines VSA+ASG
  - Interpret using abstract values
  - Apply rules to abstract stack graph to detect obfuscations
- Results
  - Can handle more instructions (`mov esp, eax`)
  - Can handle indirect jumps/calls
  - Anticipate support for `GetModuleHandle`/`GetProcAddress`
- Prototype performed as expected, though more work is needed