

The Hyperion System: Computing Software Behavior with Function Extraction Technology

Software Assurance Forum

Rick Linger

Cyber Security and Information Intelligence Research Group

Oak Ridge National Laboratory

202-701-6257, lingerr@ornl.gov



Two Observations

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division

Software with unknown behavior has
unknown security

Computer programs are mathematical
artifacts, subject to mathematical
analysis

Software Behavior Computation

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division

- New technology for software (malware) analysis
- What is computed behavior?
 - What a program does in all circumstances
 - The as-built specification
- What are key properties?
 - Programs as rules for mathematical functions
 - Operates on semantics, not syntax
 - Analyzes binaries to approach ground truth
 - Mathematical precision, no heuristics
 - Does not look for things in code

Function Extraction (FX) History

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division

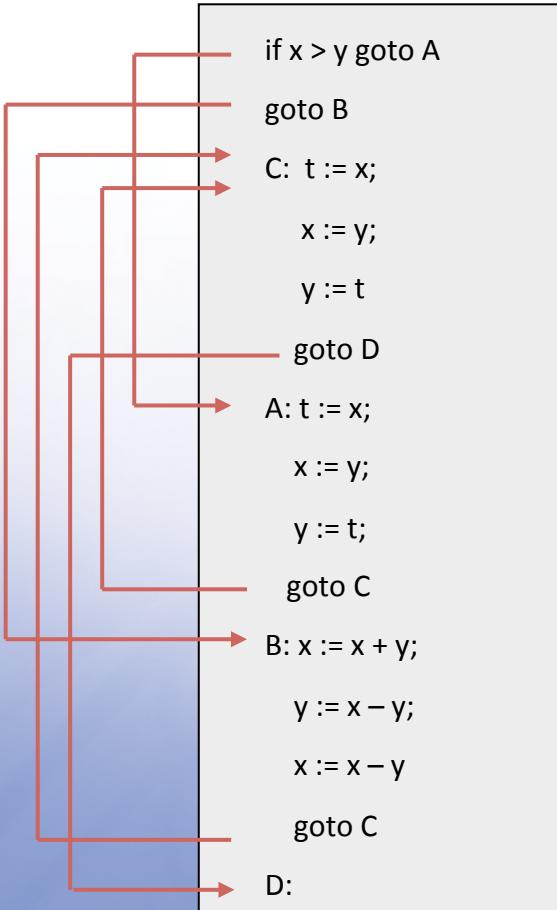
- 1990s: Mathematical foundations for deriving program behavior developed at IBM
- 2007: Carnegie Mellon CERT FX project to compute behavior of compiled binaries applied to (see CERT research reports)
- 2010: FX team brought to ORNL -- access to HPC for Hyperion development
 - Automated Vulnerability Detection Project (DOE)
 - Ultrascale Verification of Security Properties Project (Internal R&D))
 - Software Modernization Research Project (MITRE/IRS)
 - Classified Project (Federal sponsor)

Key Concept: Program Structuring

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division

Unstructured spaghetti logic:



Structure Theorem

Defines transformation from complex logic into function-equivalent structured form expressed in sequence, ifthenelse, and whiledo control structures.

Transformation to structured form:

```
if x > y  
then  
  t := x;  
  x := y;  
  y := t  
else  
  x := x + y;  
  y := x - y;  
  x := x - y  
endif;  
t := x;  
x := y;  
y := t
```

Automatically transforms code into systematic form

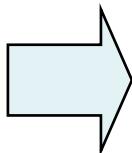
Key Concept: Behavior Computation

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division

Program:

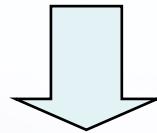
```
do
  x := x + y;
  y := x - y;
  x := x - y
enddo
```



Correctness
Theorem

Defines mathematical transformations from procedural logic expressed in sequence, ifthenelse, and whiledo forms into behaviorally-equivalent, functional forms.

Computation:



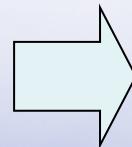
assignment	x	y
1 x := x + y	$x_1 = x_0 + y_0$	$y_1 = y_0$
2 y := x - y	$x_2 = x_1$	$y_2 = x_1 - y_1$
3 x := x - y	$x_3 = x_2 - y_2$	$y_3 = y_2$

Derivations:

$$\begin{aligned}x_3 &= x_2 - y_2 \\&= x_1 - (x_1 - y_1) \\&= y_1 \\&= y_0\end{aligned}$$

$$\begin{aligned}y_3 &= y_2 \\&= x_1 - y_1 \\&= x_0 + y_0 - y_0 \\&= x_0\end{aligned}$$

(x, y integers;
machine precision aside)



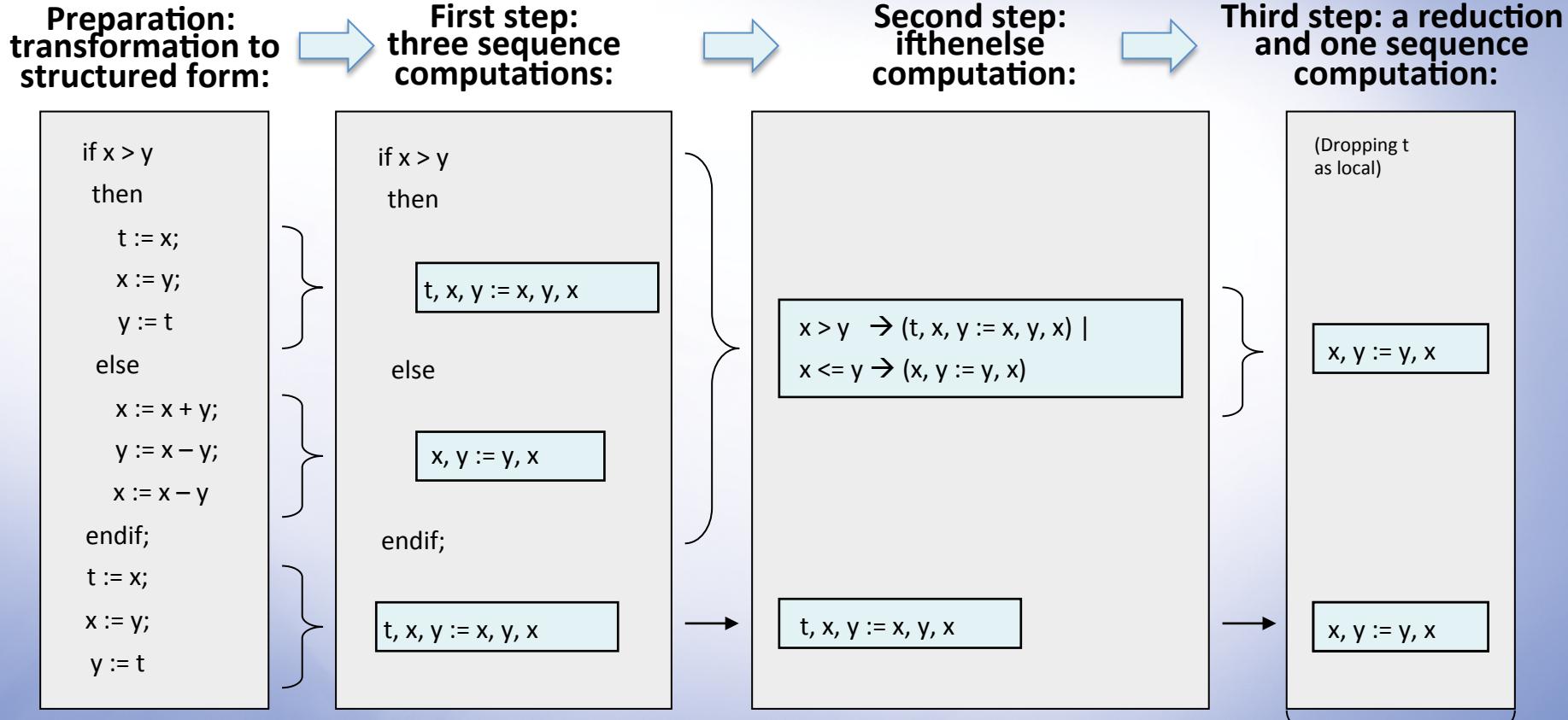
Computed behavior:

true → $\left. \begin{array}{l} x := y \\ y := x \end{array} \right\}$ Conditional concurrent assignment (CCA)
(swaps values of x and y)

Key Concept: Stepwise Computation

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division



(x, y, t integers; t a local variable; machine precision aside)

Mathematical analysis of program semantics

Program is identity function
-- has no effect.

$x, y := x, y$

Example: A Loop Behavior Computation

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division

The screenshot shows the whiledo3.0 interface with three main panes: Listing, Flowgraph, and CCA.

Input code: The Listing pane shows assembly code with columns for Address, Entry, and Raw. The raw assembly code includes instructions like cmp EAX, jmp, mov EDX, EBX, and various jumps and loops.

Structured code: The Flowgraph pane displays a hierarchical tree of control flow structures. It includes nodes for DO loops, WHILE-DO loops, and nested loops. The structure shows the flow from initial comparisons and moves through various conditional and iterative paths back to exits.

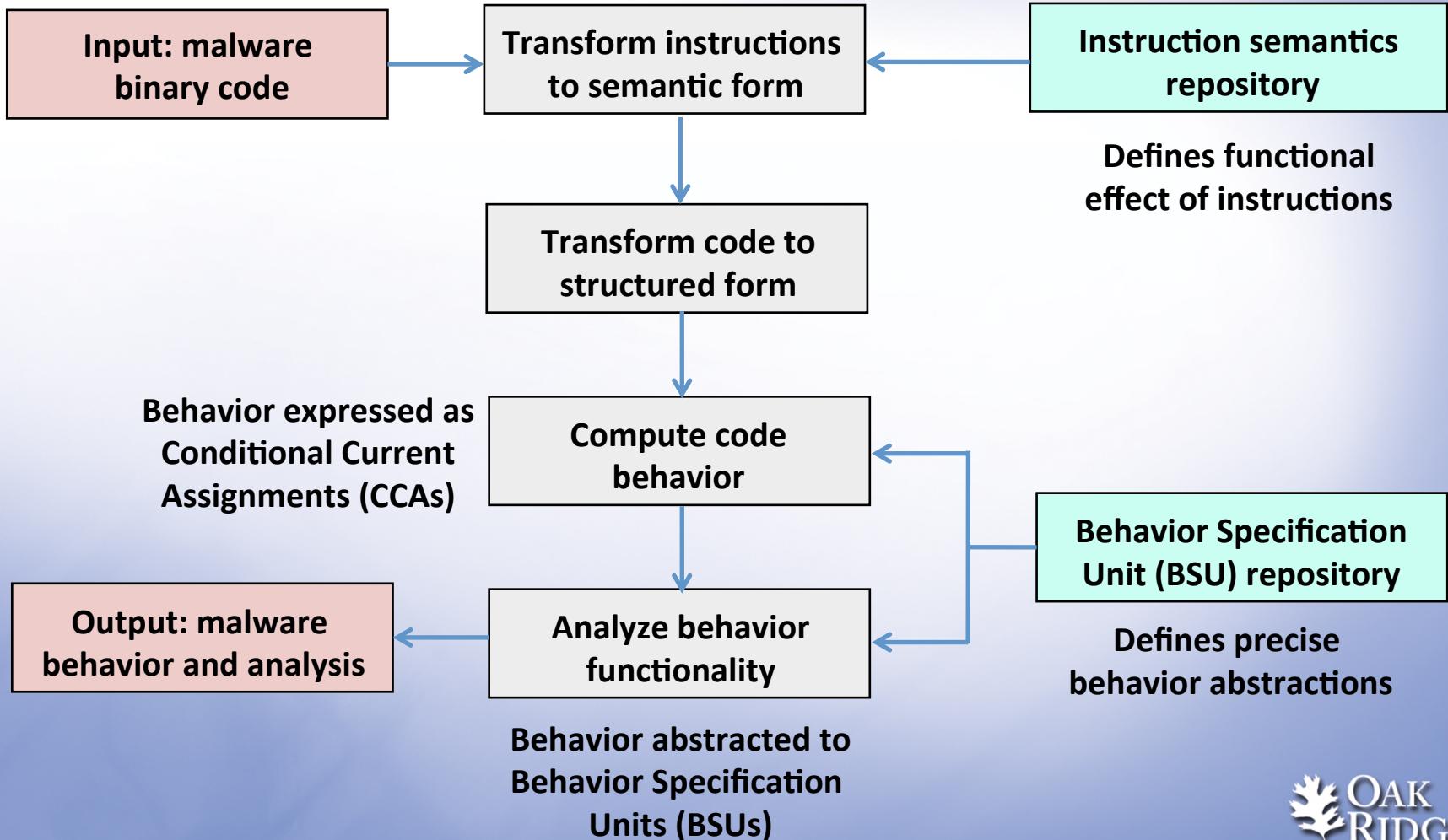
Computed behavior expressed as a Conditional Concurrent Assignment (CCA): The CCA pane shows the resulting behavior as a table. The table rows represent the state of various CPU flags and registers under different conditions (true/false). The table includes columns for Variable, AF, OF, EAX, ECX, EDX, OF, PF, SF, ZF, and label.

Variable	AF	OF	EAX	ECX	EDX	OF	PF	SF	ZF	label
true	false	false	0	((EAX+d EBX) +d ECX)	case((0 == EAX) -> EDX (0 != EAX) -> 0)	false	true	false	true	"exit"

The Behavior Computation Process

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division



Computing Behavior of a Virus in Various Forms

OW28b Virus Code

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division

Function Extractor

File View Help

Display output for file:
C:\Projects\FX\workspace\test\ours\demos\ow_28b.ext

Development Assembler Structured Behavior Extraction Library FXplorer

Zoom in Zoom out

Program: ow_28b

0x00000000	B4 4E	mov	AH, 78
0x00000002	66 8D 15 23 00 00 00	lea	DX, [35]
0x00000009	CD 21	int	33
0x0000000B	B4 3C	mov	AH, 60
0x0000000D	66 BA 9E 00	mov	DX, 0x009E
0x00000011	CD 21	int	33
0x00000013	B7 40	mov	BH, 64
0x00000015	66 93	xchg	bx, ax
0x00000017	66 8D 15 00 00 00 00	lea	DX, [0]
0x0000001E	B1 27	mov	CL, 39
0x00000020	CD 21	int	33
0x00000022	C3	ret	
0x00000023	2A 2E	sub	CH, [ESI]
0x00000025	2A 00	sub	AL, [EAX]

Virus Behavior – Four Cases

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division

The screenshot shows the FX Function Extractor interface. The title bar says "Function Extractor". The menu bar includes "File", "View", and "Help". The toolbar has buttons for "Zoom in" and "Zoom out". The main window displays assembly code for a program named "ow_28b" located at "C:\Projects\FX\workspace\test\ours\demos\ow_28b.ext". The assembly code is as follows:

```
1. mov AH, 78
2. lea DX, [35]
3. int 33
4. mov AH, 60
5. mov DX, 0x009E
6. int 33
7. mov BH, 64
8. xchg bx, ax
9. lea DX, [0]
10. mov CL, 39
11. int 33
12. (ret)
13. label = exit
```

A yellow box highlights the instruction "12. (ret)". To the right, a detailed behavior analysis is shown for each condition of the ret instruction:

- Condition 1: ?(create_file_failed?(file_name_addr=(EDX with first word set to 158), file_attribute=ECX) and then write_file_failed?(file_handle=(EBX with first word set to 158), file_attribute=ECX)), buffer_to_write=(EDX with first word set to 0), num_bytes_to_write=(ECX with first byte set to 39))
- Condition 2: ?(create_file_succeeded?(file_name_addr=(EDX with first word set to 158), file_attribute=ECX) and then write_file_failed?(file_handle=(EBX with first word set to 158), file_attribute=ECX)), buffer_to_write=(EDX with first word set to 0), num_bytes_to_write=(ECX with first byte set to 39))
- Condition 3: ?(create_file_succeeded?(file_name_addr=(EDX with first word set to 158), file_attribute=ECX) and then write_file_succeeded?(file_handle=(EBX with first word set to 158), file_attribute=ECX)), get_new_file_handle(file_name_addr=(EDX with first word set to 158), file_attribute=ECX)), buffer_to_write=(EDX with first word set to 0), num_bytes_to_write=(ECX with first byte set to 39))

The "External" section of the analysis shows:

- Dos
- Files

Under "Files", there is a highlighted line of code: FILES := (write_file(file_handle=(EBX with first word set to get_new_file_handle (file_name_addr=(EDX with first word set to 158), file_attribute=ECX)), buffer_to_write=(EDX with first word set to 0), num_bytes_to_write=(ECX with first byte set to 39)) to create_file_and_truncate(file_name_addr=(EDX with first word set to 158), file_attribute=ECX))

Annotations on the left side of the interface say "Fourth case: virus writes itself (39 bytes) into the existing file, overwriting your data". A yellow box on the right contains the text "This is a self-replicating virus that destroys your data".

Virus Obfuscated by an Intruder Tool

Massive amount of spaghetti logic has been introduced

```
.text:0800001A  xor    ax, bp          ; CODE XREF: .text:08000126↓j
.text:0800001A  jnp   m20
.text:0800001D  xor    ax, bp          ; CODE XREF: .text:08000126↓j
.text:0800001D  jnp   m21
.text:08000022  ;
.text:08000022  ;
.text:08000022  xor    dx, 2E13h        ; CODE XREF: .text:08000061↓j
.text:08000022  jnp   m1b
.text:08000027  ;
.text:08000027  xor    dx, 2E13h        ; CODE XREF: .text:08000061↓j
.text:08000027  jnp   m3a
.text:08000027  xor    dx, 2E13h        ; CODE XREF: .text:08000061↓j
.text:08000027  jnp   m3b
.text:08000031  ;
.text:08000031  add   ch, 004h        ; CODE XREF: .text:080000F2↓j
.text:08000031  jnp   m3F
.text:08000034  ;
.text:08000034  pop   ebx
.text:08000034  jnp   m45
.text:08000039  ;
.text:08000039  xor    dx, 004h        ; CODE XREF: .text:08000189↓j
.text:08000039  jnp   m46
.text:0800003F  ;
.text:0800003F  xor    dx, 004h        ; CODE XREF: .text:m2e↓j
.text:0800003F  jnp   m30
.text:08000044  ;
.text:08000044  inc   ebx
.text:08000044  jnp   m18
.text:08000044  inc   ebx
.text:08000044  jnp   m19
.text:08000049  ;
.text:08000049  inc   ebx
.text:08000049  jnp   m8
.text:08000049  inc   ebx
.text:08000049  jnp   m9
.text:0800004F  ;
.text:0800004F  xor    dx, 004h        ; CODE XREF: .text:m48↓j
.text:0800004F  jnp   m4a
.text:08000054  ;
.text:08000054  add   ch, 0B3h        ; CODE XREF: .text:08000086↓j
.text:08000054  jnp   m41
.text:08000054  add   ch, 0B3h        ; CODE XREF: .text:08000086↓j
.text:08000054  jnp   m42
.text:0800005C  ;
.text:0800005C  xor    dx, 0B73Ch      ; CODE XREF: .text:08000154↓j
.text:0800005C  jnp   m39
.text:08000061  ;
.text:08000061  xor    dx, 0B73Ch      ; CODE XREF: .text:08000154↓j
.text:08000061  jnp   short m3a
.text:08000063  ;
.text:08000063  xor    bh, al          ; CODE XREF: .text:08000082↓j
.text:08000063  jnp   m36
.text:08000065  ;
.text:08000065  xor    bh, al          ; CODE XREF: .text:08000082↓j
.text:08000065  jnp   m37
.text:0800006A  ;
```

.text:080000F7: Can't find name (hint: use manual arg)
.text:080000F7: Can't find name (hint: use manual arg)
.text:080000F7: Can't find name (hint: use manual arg)

AU: idle Down Disk: 10GB

Obfuscated Virus – Same Behavior

Spaghetti logic transformed to structured form (jumps are comments)

Show whole program behavior

- 1. jmp 0x000001D6
- 2. jmp 0x00000160
- 3. jmp 0x00000005
- 4. sub AX, BX
- 5. jmp 0x00000140
- 6. add AX, BX
- 7. jmp 0x000001AE
- 8. jmp 0x000000C0
- 9. jmp 0x00000040
- 10. inc ebx
- 11. jmp 0x00
- 12. dec ebx
- 13. jmp 0x00
- 14. mov AH, 78
- 15. jmp 0x00000080
- 16. push ebx
- 17. jmp 0x0000007B
- 18. jmp 0x000001C0
- 19. jmp 0x00000076
- 20. jmp 0x000000C2
- 21. Jmp EBP [1265]

Fourth case: the virus opens and writes to an existing file, overwriting your data (length of virus is different)

Same behavior is calculated despite the obfuscation

Computed Behavior

- + condition: ?(create_file_failed?(file_name_addr=(EDX with first word set to 158), file_attribute=ECX) and then write_file_succeeded?(file_handle=(EBX with first word set to create_file_error_code(file_name_addr=(EDX with first word set to 158), file_attribute=ECX)), buffer_to_write=(EDX with first word set to 0), num_bytes_to_write=(ECX with first word set to 481)))
- + condition: ?(create_file_failed?(file_name_addr=(EDX with first word set to 158), file_attribute=ECX) and then write_file_failed?(file_handle=(EBX with first word set to create_file_error_code(file_name_addr=(EDX with first word set to 158), file_attribute=ECX)), buffer_to_write=(EDX with first word set to 0), num_bytes_to_write=(ECX with first word set to 481)))
- + condition: ?(create_file_succeeded?(file_name_addr=(EDX with first word set to 158), file_attribute=ECX) and then write_file_failed?(file_handle=(EBX with first word set to get_new_file_handle(file_name_addr=(EDX with first word set to 158), file_attribute=ECX)), buffer_to_write=(EDX with first word set to 0), num_bytes_to_write=(ECX with first word set to 481)))
- condition: ?(create_file_succeeded?(file_name_addr=(EDX with first word set to 158), file_attribute=ECX) and then write_file_succeeded?(file_handle=(EBX with first word set to get_new_file_handle(file_name_addr=(EDX with first word set to 158), file_attribute=ECX)), buffer_to_write=(EDX with first word set to 0), num_bytes_to_write=(ECX with first word set to 481)))

Registers
Flags
Memory
External
DOS
Files

FILES := (write_file(file_handle=(EBX with first word set to get_new_file_handle(file_name_addr=(EDX with first word set to 158), file_attribute=ECX)), buffer_to_write=(EDX with first word set to 0), num_bytes_to_write=(ECX with first word set to 481)) to create_file_and_truncate(file_name_addr=(EDX with first word set to 158), file_attribute=ECX))

FX blocks control obfuscation as a weapon for intruders

Computed Behavior of 10K-line Program

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division

The screenshot shows the FXML Function Extractor interface. The title bar says "Function Extractor". The menu bar includes "File", "View", and "Help". The toolbar has buttons for "Zoom in" and "Zoom out". The main window displays the file path "C:\Projects\FX\workspace\test\ours\demos\source\10kprogram.ext". The tabs at the top are "Development", "Assembler", "Structured", "Behavior", "Extraction Library", and "FExplorer". The "Behavior" tab is selected. The left pane shows assembly code numbered 1 to 28. The right pane shows "Computed Behavior" with two cases: one for `?(OF != SF)` and another for `?(OF == SF)`. A yellow callout box points to the "Behavior" tab with the text "Two cases of behavior". A yellow callout box in the bottom right corner asks "What happens when the OW28b virus is hidden in this program?" and lists two bullet points: "• No two lines adjacent." and "• Triggered by an obscure condition."

Two cases of behavior

What happens when the OW28b virus is hidden in this program?

- No two lines adjacent.
- Triggered by an obscure condition.

Program: 10kprogram

Show whole program behavior

1. push eax
2. push ebx
3. //if-then-else function
4. IF not_equiv[OF,SF]
5. THEN //then part function
6. push eax
7. //if-then-else function
8. IF equiv[OF,SF]
9. THEN //then part function
10. push ecx
11. sub ECX, ECX
12. sub ECX, EAX
13. add ECX, EBX
14. sub EAX, EAX
15. jmp 0x00000005F
16. add EAX, 0x000000008
17. ELSE //else part function
18. push ebx
19. pop eax
20. pop ebx
21. pop eax
22. pop ebx
23. jmp 0x000000035
24. push ecx
25. sub ECX, ECX
26. sub ECX, EAX
27. add ECX, EBX
28. sub EAX, EAX

Computed Behavior

- + condition: ?(OF != SF)
- + condition: ?(OF == SF)

Behavior of Program with OW28b Virus

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division

Function Extractor

File View Help

Display output for file:
C:\Projects\FX\workspace\test\ours\demos\source\10kprogramEmbeddedOWB.ext

Development Assembler Structured Behavior Extraction Library FXplorer

Zoom in Zoom out Open All Close All Zoom in Zoom out

Program: 10kprogramEmbeddedOWB

Show whole program behavior

1. push eax
2. push ebx
3. //if-then-else function
4. IF not_equiv(OF,SF)
5. THEN //then part function
6. push
7. //if-
8. IF e
9. THE
10. push eax
11. push ebx
12. pop eax
13. pop ebx
14. push eax
15. push ebx
16. pop eax
17. push eax
18.
19.
20.
21.
22. push eax
23. push ebx
24. pop eax
25. pop ebx
26. push eax
27. push ebx
28. pop eax
29. pop ebx
30. push eax
31. push ebx
32. pop eax
33. pop ebx

Five cases of behavior

Cases 1,2, 4, and 5 show the ow_28b virus behavior

Computed Behavior

- + condition: ?((OF == SF) and then create_file_succeeded?(file_name_addr=(EDX with first word set to 158), file_attribute=ECX) and then write_file_succeeded?(file_handle=(EBX with first word set to 158), file_attribute=ECX), buffer_to_write=(EDX with first word set to 9629), num_bytes_to_write=(ECX with first byte set to 43))
- + condition: ?((OF == SF) and then create_file_failed?(file_name_addr=(EDX with first word set to 158), file_attribute=ECX) and then write_file_failed?(file_handle=(EBX with first word set to 158), file_attribute=ECX), buffer_to_write=(EDX with first word set to 9629), num_bytes_to_write=(ECX with first byte set to 43))
- + condition: ?(OF != SF)
- + condition: ?((OF == SF) and then create_file_succeeded?(file_name_addr=(EDX with first word set to 158), file_attribute=ECX) and then write_file_error_code(file_name_addr=(EDX with first word set to 158), file_attribute=ECX), buffer_to_write=(EDX with first word set to 9629), num_bytes_to_write=(ECX with first byte set to 43))

Registers Flags Memory External Dos Files

FILES := (write_file(file_handle=(EBX with first word set to get_new_file_handle(file_name_addr=(EDX with first word set to 158), file_attribute=ECX)), buffer_to_write=(EDX with first word set to 9629), num_bytes_to_write=(ECX with first byte set to 43)) to create_file_and_truncate(file_name_addr=(EDX with first word set to 158), file_attribute=ECX))

+ condition: ?((OF == SF) and then create_file_failed?(file_name_addr=(EDX with first word set to 158), file_attribute=ECX) and then write_file_error_code(file_name_addr=(EDX with first word set to 158), file_attribute=ECX), buffer_to_write=(EDX with first word set to 9629), num_bytes_to_write=(ECX with first byte set to 43))

One of the two original cases

FX blocks obfuscation through hiding as a weapon for intruders

16

Computing Behavior of an Embedded Software Component

Behavior Computation for Robot Arm

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division

- Matrix operations key to robot spatial maneuvering
- Behavior computations carried out for assembly code that computes $A \times B \rightarrow C$:
 1. Various matrix sizes
No errors, obfuscations, or malware inserted
 2. Programming errors/insider corrupted functionality
Errors/changes inserted
 3. Intruder attacks
Obfuscation, interleaving, malware inserted

Behavior Specification Units (BSUs)

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division

- BSUs operate on computed behavior:
 - Pre-defined units of specification
 - Very general (many implementations, one behavior)
 - Define once, reuse forever
- Serve as
 - Functional specifications: Malware components
 - Semantic signatures: Malicious behavior
 - Behavior structures: Human understanding

BSUs for Matrix Operations

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division

BSU definitions:

- Vector memory shape BSU
start, number of elements, stride
- Dot product BSU
row and column vectors, element products and summations
- Matrix memory shape BSU
start, number of vector rows, number of vector columns
- Matrix multiplication BSU
matrix of dot products

Behavior computation for A x B → C:

4x4 and 4x4 matrices with no errors, obfuscations, or malicious content

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division

C matrix is initialized before matrix calculations are performed

CONDITION

TRUE →

REGISTERS

EAX := DOT_PRODUCT of
VECTOR(4) at (12 + (dword at (8 + ESP))) by stride 16
and
VECTOR(4) at (48 + (dword at (4 + ESP))) by stride 4

ECX := DOT_PRODUCT of
VECTOR(3) at (12 + (dword at (8 + ESP))) by stride 16
and
VECTOR(3) at (48 + (dword at (4 + ESP))) by stride 4

EDX := dword at (60 + (dword at (4 + ESP)))

EIP := dword at ESP

ESP := 4 + ESP

MEMORY

M :=
MATRIX of dwords at (dword at (12 + ESP)) :=
MATRIX_MULTIPLICATION of
MATRIX(4,4) of dwords at (dword at (4 + ESP))
and
MATRIX(4,4) of dwords at (dword at (8 + ESP)))

label := "exit"

FLAGS

AF := false
CF := arb_bool_val
OF := arb_bool_val
PF := is_even_parity_lowbyte(48 + (dword at (8 + ESP)))
SF := is_neg_signed_32(48 + (dword at (8 + ESP)))
ZF := (4294967248 == (dword at (8 + ESP)))

Behavior computation for A x B → C: 4x4 and 4x4 matrices with programming error (or corrupted function)

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division

C matrix is not initialized. Behavior computation stops at dot products of vectors because matrix multiplication BSU is not satisfied. Each C matrix element is the sum of the uninitialized value plus the dot product operation for that element.

CONDITION

TRUE →

(Flags not shown)

REGISTERS

EAX := (dword at (60 + (dword at (12 + ESP))) +
DOT_PRODUCT of
 VECTOR(4) at (12 + (dword at (8 + ESP))) by stride 16
 and
 VECTOR(4) at (48 + (dword at (4 + ESP))) by stride 4

ECX := (dword at (60 + (dword at (12 + ESP))) +
DOT_PRODUCT of
 VECTOR(3) at (12 + (dword at (8 + ESP))) by stride 16
 and
 VECTOR(3) at (48 + (dword at (4 + ESP))) by stride 4

EDX := dword at (60 + (dword at (4 + ESP)))

EIP := dword at ESP

ESP := 4 + ESP

MEMORY

M :=
dword at (dword at (12 + ESP)) :=
dword at (dword at (12 + ESP)) +
DOT_PRODUCT of
 VECTOR(4) at (dword at (4 + ESP)) by stride 4
 and
 VECTOR(4) at (dword at (8 + ESP)) by stride 16

dword at (4 + (dword at (12 + ESP))) :=
dword at (4 + (dword at (12 + ESP))) +
DOT_PRODUCT of
 VECTOR(4) at (dword at (4 + ESP)) by stride 4
 and
 VECTOR(4) at (4 + (dword at (8 + ESP))) by stride 16

dword at (8 + (dword at (12 + ESP))) :=
dword at (8 + (dword at (12 + ESP))) +
DOT_PRODUCT of
 VECTOR(4) at (dword at (4 + ESP)) by stride 4
 and
 VECTOR(4) at (8 + (dword at (8 + ESP))) by stride 16

...

Behavior computation for A x B → C:

4x4 and 4x4 matrices with obfuscation, interleaving, embedded malware

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division

CONDITION 1

```
create_file_succeeded(  
    file_name_addr = 158,  
    file_attribute = (word at (40 + (dword at (4 + ESP)))))  
and  
write_file_succeeded(  
    file_handle = get_new_file_handle(  
        file_name_addr = 158,  
        file_attribute = (word at (40 + (dword at (4 + ESP)))))),  
    buffer_to_write = 0,  
    num_bytes_to_write=6738)
```

FILE SYSTEM

```
FILES :=  
  
create_file_and_truncate(  
    file_name_addr = 158,  
    file_attribute = (word at (40 + (dword at (4 + ESP)))))  
then  
write_file(  
    file_handle = get_new_file_handle(  
        file_name_addr = 158,  
        file_attribute = (word at (40 + (dword at (4 + ESP)))))),  
    buffer_to_write = 0,  
    num_bytes_to_write = 6738)
```

MEMORY

M :=

MATRIX of dwords at 100 :=
 MATRIX_MULTIPLICATION of
 (MATRIX(3,4) of dwords at (dword at (4 + ESP)))
 and
 (MATRIX(4,4) of dwords at (dword at (8 + ESP)))

MATRIX of dwords at (dword at (12 + ESP)) :=

MATRIX_MULTIPLICATION of
 (MATRIX(4,4) of dwords at (dword at (4 + ESP)))
 and
 (MATRIX(4,4) of dwords at (dword at (8 + ESP)))

OPERATING SYSTEM

DOS :=

```
mark_file_handle_as_writing(  
    file_handle = get_new_file_handle(  
        file_name_addr=158,  
        file_attribute = (word at (40 + (dword at (4 + ESP)))))),  
    buffer_to_write=0,  
    num_bytes_to_write=6738)
```

(Flags not shown)

Behavior computation for A x B → C:

4x4 and 4x4 matrices with obfuscation, interleaving, embedded malware

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division

CONDITION 2

```
create_file_succeeded(  
    file_name_addr = 158,  
    file_attribute = (word at (40 + (dword at (4 + ESP)))))  
and  
write_file_failed(  
    file_handle = get_new_file_handle(  
        file_name_addr = 158,  
        file_attribute = (word at (40 + (dword at (4 + ESP)))))),  
    buffer_to_write = 0,  
    num_bytes_to_write=6738)
```

CONDITION 4

```
create_file_failed(  
    file_name_addr = 158,  
    file_attribute = (word at (40 + (dword at (4 + ESP)))))  
and  
write_file_failed(  
    file_handle = get_new_file_handle(  
        file_name_addr = 158,  
        file_attribute = (word at (40 + (dword at (4 + ESP)))))),  
    buffer_to_write = 0,  
    num_bytes_to_write=6738)
```

CONDITION 3

```
create_file_failed(  
    file_name_addr = 158,  
    file_attribute = (word at (40 + (dword at (4 + ESP)))))  
and  
write_file_succeeded (  
    file_handle = get_new_file_handle(  
        file_name_addr = 158,  
        file_attribute = (word at (40 + (dword at (4 + ESP)))))),  
    buffer_to_write = 0,  
    num_bytes_to_write=6738)
```

(In each case, a successful matrix multiplication is carried out, with side effects on the file system and operating system, but no malware replication.)

Project Status

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division

- Next-generation Hyperion system under development
 - More powerful semantics language
 - More powerful rewriter
 - Parallel computation
- Many applications
 - Malware detection and analysis
 - Vulnerability detection
 - Rigorous software development
 - Correctness verification
 - Supply chain analysis
 - Anti-tamper analysis
 - Hardware analysis
 - ...

Long-Term Vision

Cyberspace Sciences & Information Intelligence (CSIIR) Group

Computational Sciences & Engineering Division

Computed behavior available for all
common software

Behavior computation available for
one-off software