Automatically Deriving Pointer Reference Expressions from Binary Code for Memory Dump Analysis

Yangchun Fu[†], Zhiqiang Lin[†] and David Brumley[*]

[†]The University of Texas at Dallas
[*]Carnegie Mellon University

Sep 4th, 2015

Locating a pointer in a cash dump

```
00001800  eb 40 1b 02 63 74 00 f0  00 00 00 00 00 00 00 00  |.@..ct..........|
00001810  00 00 00 00 80 00 00 00  00 00 00 00 00 00 00 00  |................|
00001820  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00001830  00 00 00 00 00 00 00 00  10 76 16 cc 00 00 00 00  |.........v......|
00001840  00 19 66 8c d0 50 b8 08  00 00 00 66 8e d0 53 8b  |..f..P.....f..S.|
00001850  d9 ff 2d 19 02 00 00 0f  20 c0 0f ba f0 1f 0f 22  |..-..... ......"|
00001860  c0 eb 00 b9 80 00 00 c0  0f 32 0f ba f0          |.........2.....0|
00001870  0f 20 e0 0f ba f0 05 0f  22 e0 60 9c              |. ......".`.....|
00001880  04 89 a3 76 02 00 00 0f  01 83 80 02              |...v............|
00001890  8b 88 02 00 00 8b 8b 3c  00 00 00 0b              |.......<.....t..|
000018a0  b3 38 00 00 00 8b fb 81  c7 00 30 00              |.8........0..+..|
000018b0  a4 0f 01 9b 90 02 00 00  0f 01 93 68              |...........h...f|
000018c0  b8 10 00 66 8e d8 66 8e  c0 66 8e d0 66          |...f..f..f..f..f|
…
*
00100f60  00 00 00 00 00 00 00 00  00 f0 ff 5d 76 e3 f0 2f  |...........]v../|
00100f70  93 c9 a4 1d f9 48 be f8  6c c7 1d 92 4c 1e 6e 35  |.....H..l...L.n5|
00100f80  b4 f8 1b ae f6 69 e8 c0  b7 34 74 a1 4e 5a a7 93  |.....i...4t.NZ..|
00100f90  97 2f f3 47 cf d7 10 df  f0 d6 e3 9b f5 cf a9 23  |./.G...........#|
00100fa0  cd 9f 87 4f 37 7f 1e f1  fe dc 7d b9 f9 f3 7b ef  |...O7.....}...{.|
00100fb0  cf 95 bf 94 3f 8d 63 9a  cc 8a 36 5b 56 7b d2 76  |....?.c...6[V{.v|
00100fc0  b6 d9 ad ee 61 f6 90 a4  2c 2b 54 66 37 de 3d a9  |....a...,+Tf7.=.|
00100fd0  b9 d9 67 37 1e 7a b5 ce  ef 0c 58 ee 4d 30 d0 9b  |..g7.z....X.M0..|
00100fe0  c0 6e bc e7 3d f3 e7 d0  9a bf a4 82 1b c7 9c f1  |.n..=...........|
00100ff0  db 66 2b d8 38 cb 2a 91  80 ad 7d 25 d8 0a e5 db  |.f+.8.*...}%....|
```

c174128b

Locating a pointer in a cash dump

```
00001800  eb 40 1b 02 63 74 00 f0  00 00 00 00 00 00 00 00  |.@..ct..........|
00001810  00 00 00 00 80 00 00 00  00 00 00 00 00 00 00 00  |................|
00001820  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00001830  00 00 00 00 00 00 00 00  10 76 16 cc 00 00 00 00  |.........v......|
00001840  00 19 66 8c d0 50 b8 08  00 00 00 66 8e d0 53 8b  |..f..P.....f..S.|
00001850  d9 ff 2d 19 02 00 00 0f  20 c0 0f ba f0 1f 0f 22  |..-..... ......"|
00001860  c0 eb 00 b9 80 00 00 c0  0f 32 0f ba f0          |.........2.....0|
00001870  0f 20 e0 0f ba f0 05 0f  22 e0 60 9c              |. ......".`.....|
00001880  04 89 a3 76 02 00 00 0f  01 83 80 02              |...v............|
00001890  8b 88 02 00 00 8b 8b 3c  00 00 00 0b              |.......<.....t..|
000018a0  b3 38 00 00 00 8b fb 81  c7 00 30 00              |.8........0..+..|
000018b0  a4 0f 01 9b 90 02 00 00  0f 01 93 68              |...........h...f|
000018c0  b8 10 00 66 8e d8 66 8e  c0 66 8e d0 66          |...f..f..f..f..f|
…
*
00100f60  00 00 00 00 00 00 00 00  00 f0 ff 5d 76 e3 f0 2f  |...........]v../|
00100f70  93 c9 a4 1d f9 48 be f8  6c c7 1d 92 4c 1e 6e 35  |.....H..l...L.n5|
00100f80  b4 f8 1b ae f6 69 e8 c0  b7 34 74 a1 4e 5a a7 93  |.....i...4t.NZ..|
00100f90  97 2f f3 47 cf d7 10 df  f0 d6 e3 9b f5 cf a9 23  |./.G...........#|
00100fa0  cd 9f 87 4f 37 7f 1e f1  fe dc 7d b9 f9 f3 7b ef  |...O7.....}...{.|
00100fb0  cf 95 bf 94 3f 8d 63 9a  cc 8a 36 5b 56 7b d2 76  |....?.c...6[V{.v|
00100fc0  b6 d9 ad ee 61 f6 90 a4  2c 2b 54 66 37 de 3d a9  |....a...,+Tf7.=.|
00100fd0  b9 d9 67 37 1e 7a b5 ce  ef 0c 58 ee 4d 30 d0 9b  |..g7.z....X.M0..|
00100fe0  c0 6e bc e7 3d f3 e7 d0  9a bf a4 82 1b c7 9c f1  |.n..=...........|
00100ff0  db 66 2b d8 38 cb 2a 91  80 ad 7d 25 d8 0a e5 db  |.f+.8.*...}%....|
```

c174128b

Pointer is extremely valuable

- Root cause of segmentation fault.
- Direct target of control flow hijacks.

Locating a pointer in a cash dump

```
00001800  eb 40 1b 02 63 74 00 f0  00 00 00 00 00 00 00 00  |.@..ct..........|
00001810  00 00 00 00 80 00 00 00  00 00 00 00 00 00 00 00  |................|
00001820  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00001830  00 00 00 00 00 00 00 00  10 76 16 cc 00 00 00 00  |.........v......|
00001840  00 19 66 8c d0 50 b8 08  00 00 00 66 8e d0 53 8b  |..f..P.....f..S.|
00001850  d9 ff 2d 19 02 00 00 0f  20 c0 0f ba f0 1f 0f 22  |..-..... ......"|
00001860  c0 eb 00 b9 80 00 00 c0  0f 32 0f ba f0          |.........2.....0|
00001870  0f 20 e0 0f ba f0 05 0f  22 e0 60 9c              |. ......".`.....|
00001880  04 89 a3 76 02 00 00 0f  01 83 80 02              |...v............|
00001890  8b 88 02 00 00 8b 8b 3c  00 00 00 0b              |.......<.....t..|
000018a0  b3 38 00 00 00 8b fb 81  c7 00 30 00              |.8........0..+..|
000018b0  a4 0f 01 9b 90 02 00 00  0f 01 93 68              |...........h...f|
000018c0  b8 10 00 66 8e d8 66 8e  c0 66 8e d0 66          |...f..f..f..f..f|
…
*
00100f60  00 00 00 00 00 00 00 00  00 f0 ff 5d 76 e3 f0 2f  |...........]v../|
00100f70  93 c9 a4 1d f9 48 be f8  6c c7 1d 92 4c 1e 6e 35  |.....H..l...L.n5|
00100f80  b4 f8 1b ae f6 69 e8 c0  b7 34 74 a1 4e 5a a7 93  |.....i...4t.NZ..|
00100f90  97 2f f3 47 cf d7 10 df  f0 d6 e3 9b f5 cf a9 23  |./.G...........#|
00100fa0  cd 9f 87 4f 37 7f 1e f1  fe dc 7d b9 f9 f3 7b ef  |...O7.....}...{.|
00100fb0  cf 95 bf 94 3f 8d 63 9a  cc 8a 36 5b 56 7b d2 76  |....?.c...6[V{.v|
00100fc0  b6 d9 ad ee 61 f6 90 a4  2c 2b 54 66 37 de 3d a9  |....a...,+Tf7.=.|
00100fd0  b9 d9 67 37 1e 7a b5 ce  ef 0c 58 ee 4d 30 d0 9b  |..g7.z....X.M0..|
00100fe0  c0 6e bc e7 3d f3 e7 d0  9a bf a4 82 1b c7 9c f1  |.n..=...........|
00100ff0  db 66 2b d8 38 cb 2a 91  80 ad 7d 25 d8 0a e5 db  |.f+.8.*...}%....|
```

c174128b

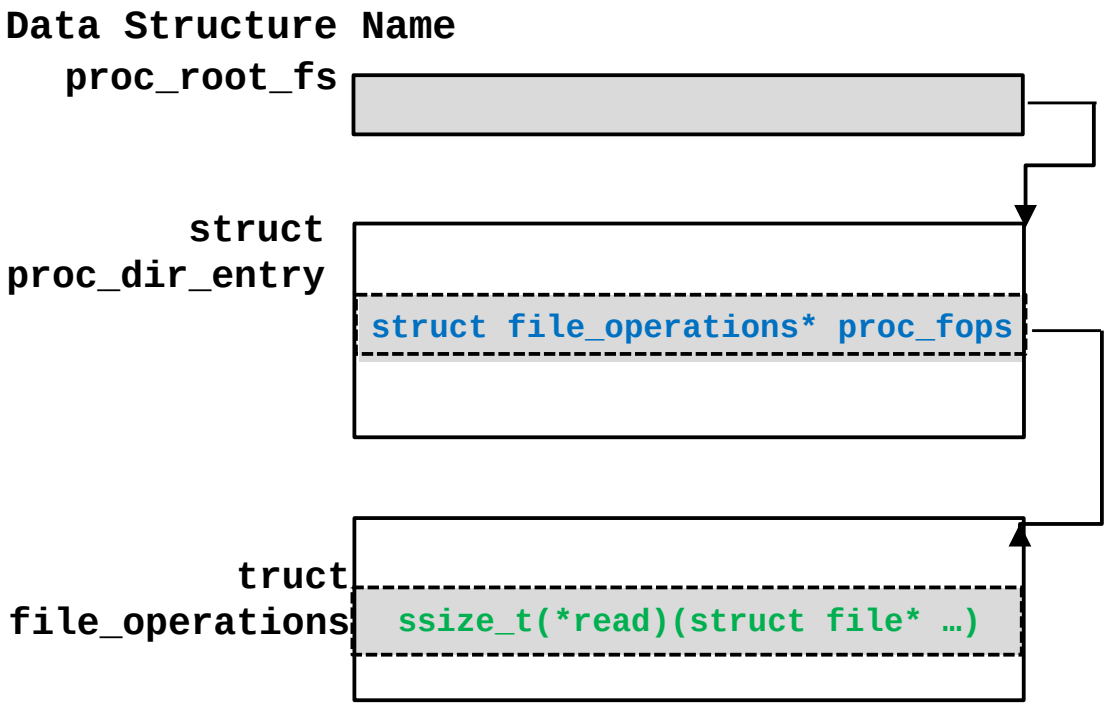
Pointer is extremely valuable

- Root cause of segmentation fault.
- Direct target of control flow hijacks.

Challenge

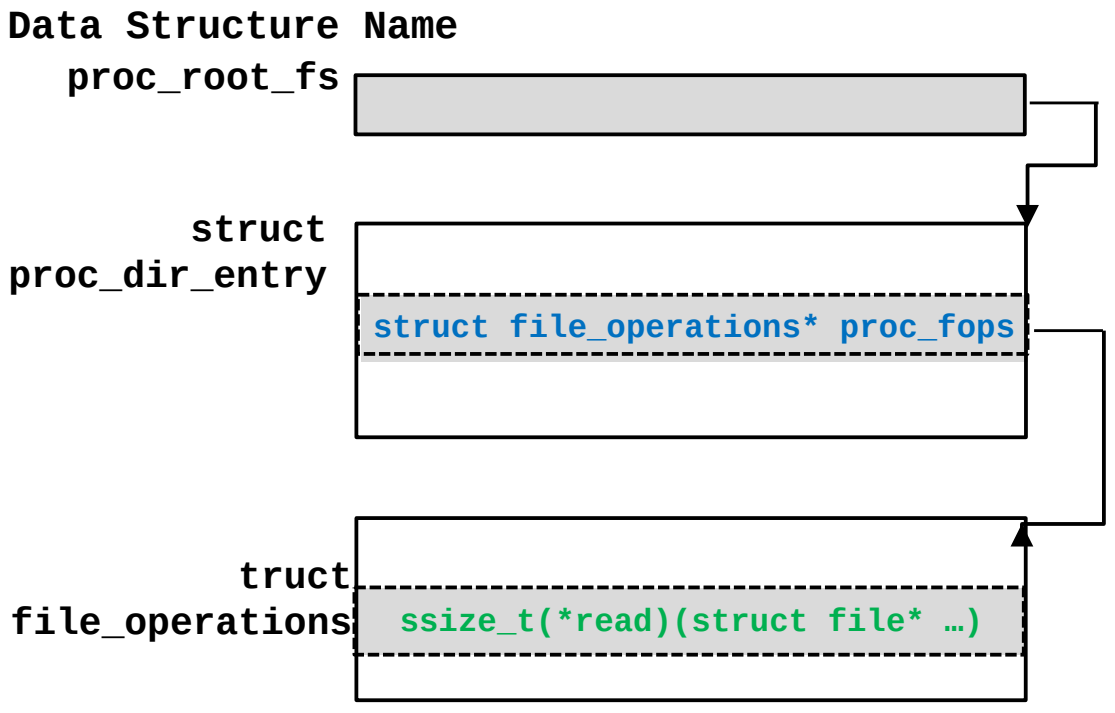
- Low level bits and bytes data.
- Requiring kernel data structure to traverse a pointer (Intuitively)

State-of-the-art



proc_root_fs->proc_fops->read

State-of-the-art



proc_root_fs->proc_fops->read

Can we directly traverse pointers without data structure knowledge and source code information?

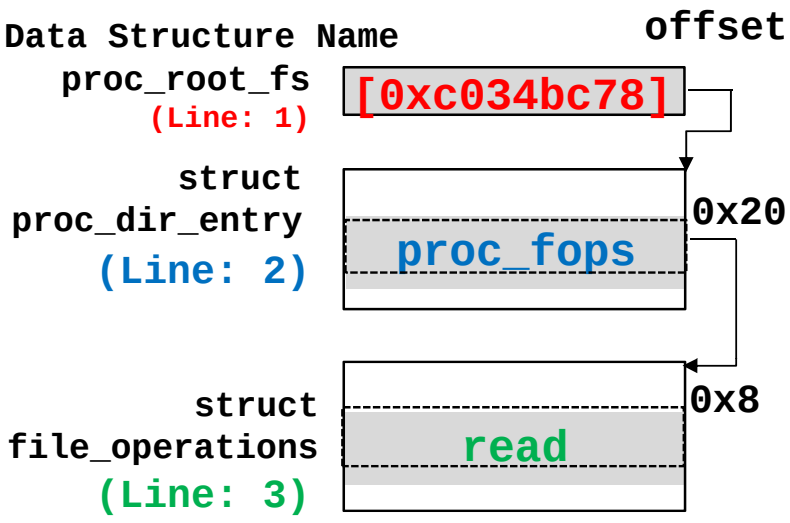
Recognizing Pointer Traversal Instructions

```
1.0xd894e007: mov  0xc034bc78,%eax

2.0xd894e00c: mov  0x20(%eax),%eax

3.0xd894e013: call 0x8(%eax)
```

Recognizing Pointer Traversal Instructions

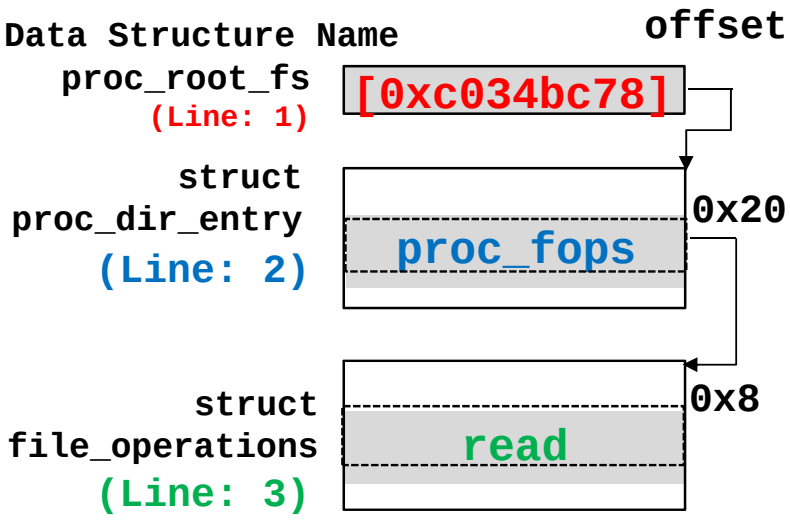
```
//Global Variable:struct proc_root_fs
1.0xd894e007: mov  0xc034bc78,%eax

// proc_root_fs->proc_fops
2.0xd894e00c: mov  0x20(%eax),%eax

// proc_root_fs->proc_fops->read
3.0xd894e013: call 0x8(%eax)
```

Recognizing Pointer Traversal Instructions




```
//Global Variable:struct proc_root_fs
1.0xd894e007: mov  0xc034bc78,%eax

// proc_root_fs->proc_fops
2.0xd894e00c: mov  0x20(%eax),%eax

// proc_root_fs->proc_fops->read
3.0xd894e013: call 0x8(%eax)
```

Pointer Reference Expression (ptr-rexp)




```
//Global Variable:struct proc_root_fs
1.0xd894e007: mov  0xc034bc78,%eax

// proc_root_fs->proc_fops
2.0xd894e00c: mov  0x20(%eax),%eax

// proc_root_fs->proc_fops->read
3.0xd894e013: call 0x8(%eax)
```

The ptr-rexp for `read`: *(*(*(0xc034bc78)+0x20)+0x8)

Binary Code Analysis: *(*(*(0xc034bc78)+0x20)+0x8)

```
0xd894e007: mov  0xc034bc78,%eax
0xd894e00c: mov  0x20(%eax),%eax
0xd894e013: call 0x8(%eax)
```

Binary Code Analysis: *(*(*(0xc034bc78)+0x20)+0x8)

```
0xd894e007: mov  0xc034bc78,%eax
0xd894e00c: mov  0x20(%eax),%eax
0xd894e013: call 0x8(%eax)
```

Static Analysis

- Disassemble challenge
- Over approximation (e.g., no bounds on loop)

Binary Code Analysis: *(*(*(0xc034bc78)+0x20)+0x8)

```
0xd894e007: mov  0xc034bc78,%eax
0xd894e00c: mov  0x20(%eax),%eax
0xd894e013: call 0x8(%eax)
```

Static Analysis

- Disassemble challenge
- Over approximation (e.g., no bounds on loop)

Dynamic Analysis

- Correctly disassembly
- Sound but Incomplete (coverage issues)

Binary Code Analysis: *(*(*(0xc034bc78)+0x20)+0x8)

```
0xd894e007: mov  0xc034bc78,%eax
0xd894e00c: mov  0x20(%eax),%eax
0xd894e013: call 0x8(%eax)
```

Static Analysis

- Disassemble challenge
- Over approximation (e.g., no bounds on loop)

Dynamic Analysis

- Correctly disassembly
- Sound but Incomplete (coverage issues)

We favor soundness over completeness, and therefore use dynamic analysis

Applications: Pointer Integrity Check & Crash Analysis

```
0xd894e007: mov  0xc034bc78,%eax
0xd894e00c: mov  0x20(%eax),%eax
0xd894e013: call 0x8(%eax)
```

Applications: Pointer Integrity Check & Crash Analysis

```
0xd894e007: mov  0xc034bc78,%eax
0xd894e00c: mov  0x20(%eax),%eax
0xd894e013: call 0x8(%eax)
```

Trusted Computer

Applications: Pointer Integrity Check & Crash Analysis

```
0xd894e007: mov  0xc034bc78,%eax
0xd894e00c: mov  0x20(%eax),%eax
0xd894e013: call 0x8(%eax)
```

Trusted Computer

```
*(*(*(0xc034bc78)+0x20)+0x8)
```

Applications: Pointer Integrity Check & Crash Analysis

```
0xd894e007: mov  0xc034bc78,%eax
0xd894e00c: mov  0x20(%eax),%eax
0xd894e013: call 0x8(%eax)
```

Trusted Computer

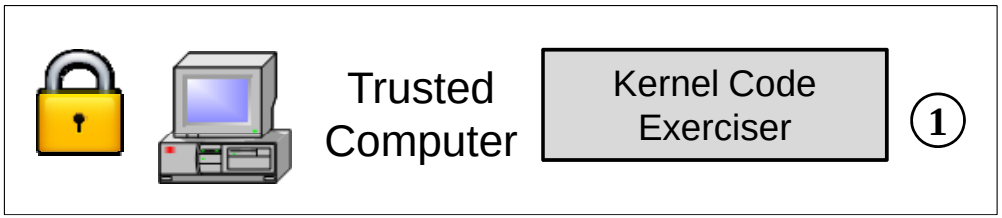
((*(0xc034bc78)+0x20)+0x8)

Patient Computer

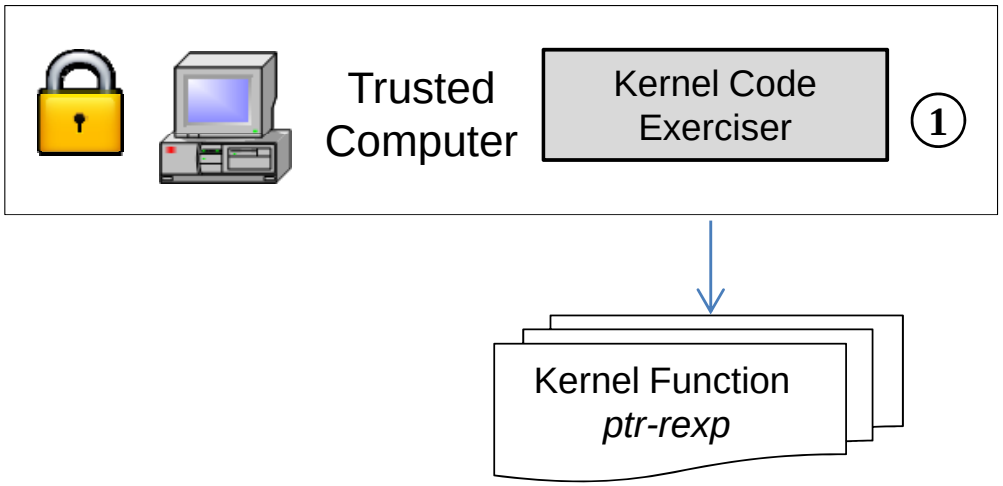
((*(0xc034bc78)+0x20)+0x8)

Function **P**ointer Integrity **C**hec**k**er: FPCK

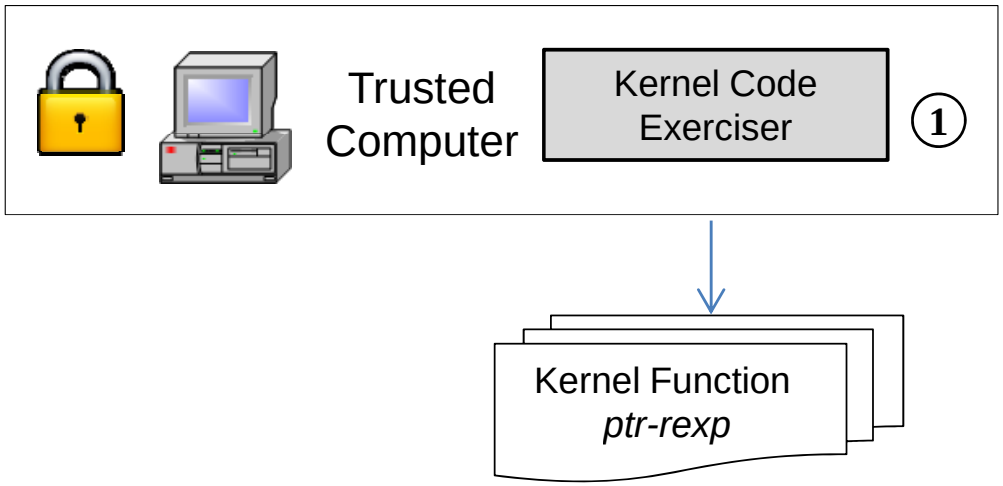
Function Pointer Integrity Checker: FPCK

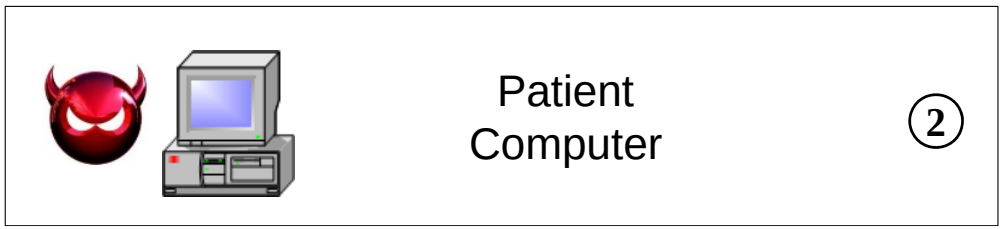


Function Pointer Integrity Checker: FPCK

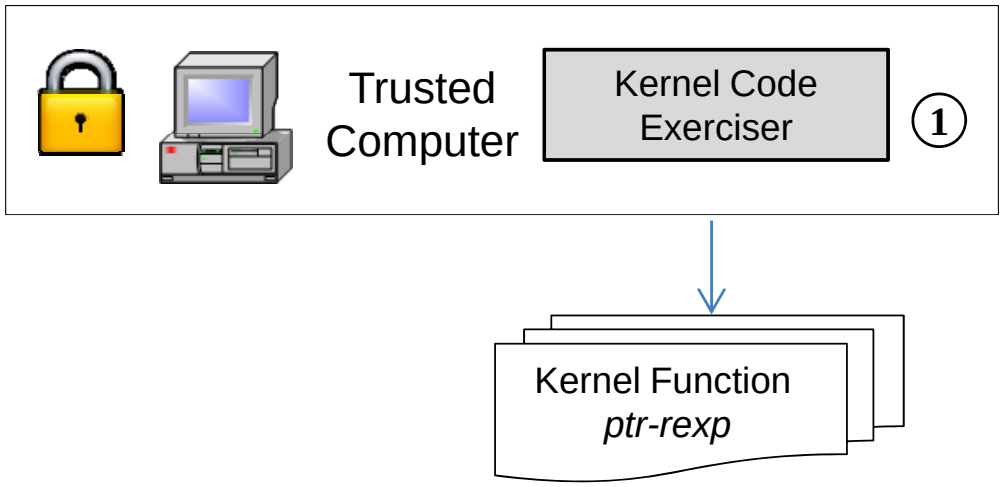


Function Pointer Integrity Checker: FPCK

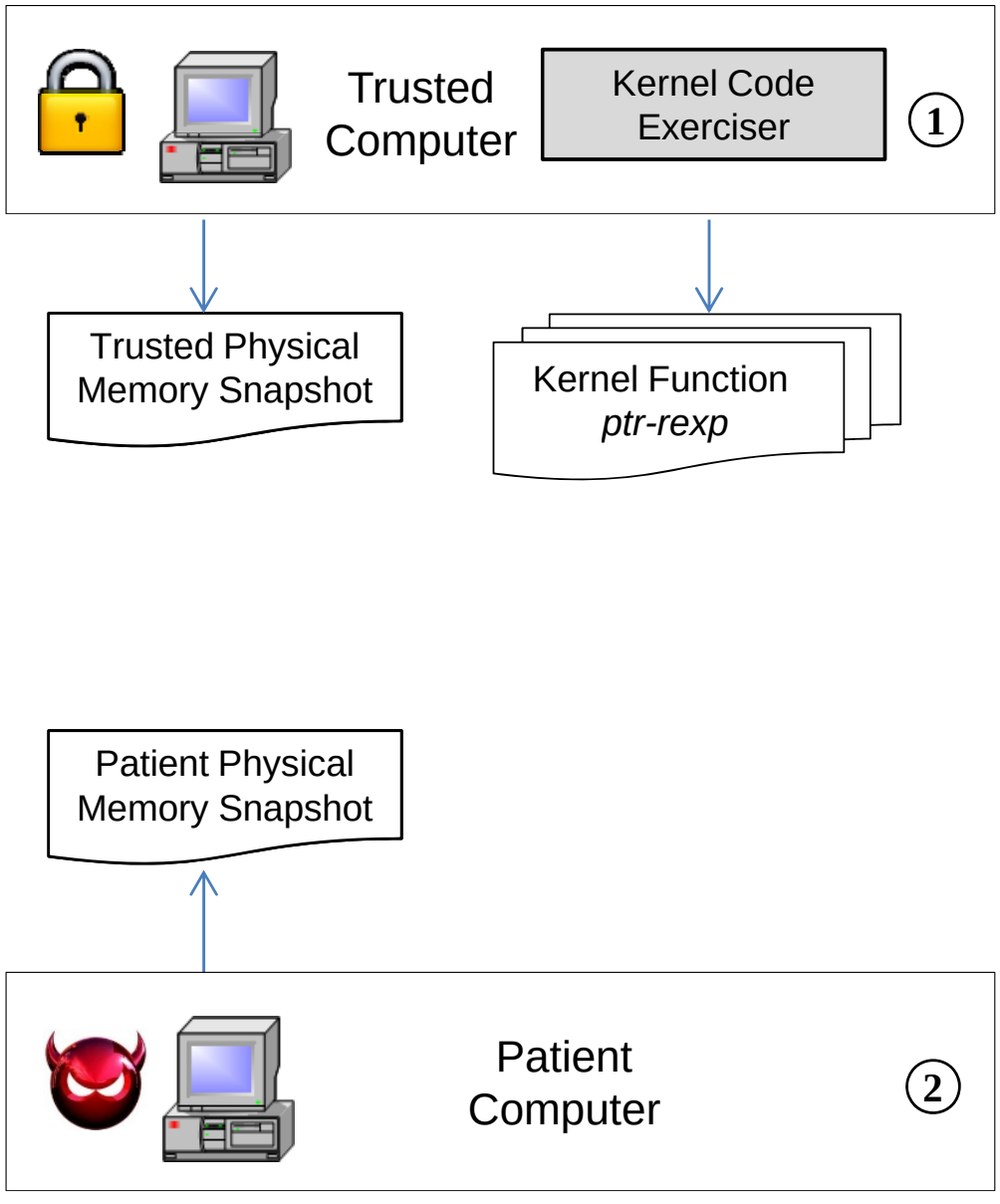




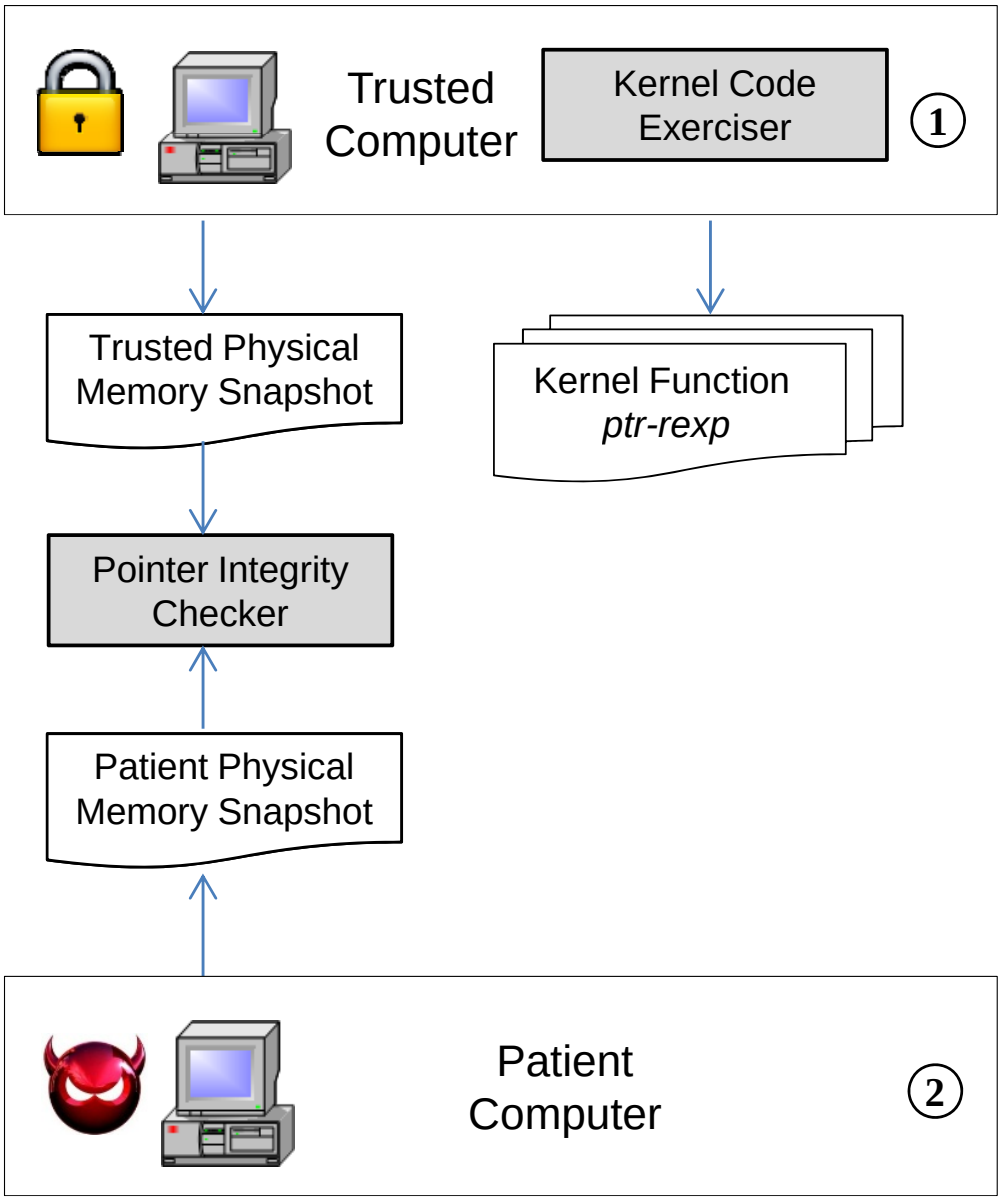
Function Pointer Integrity Checker: FPCK



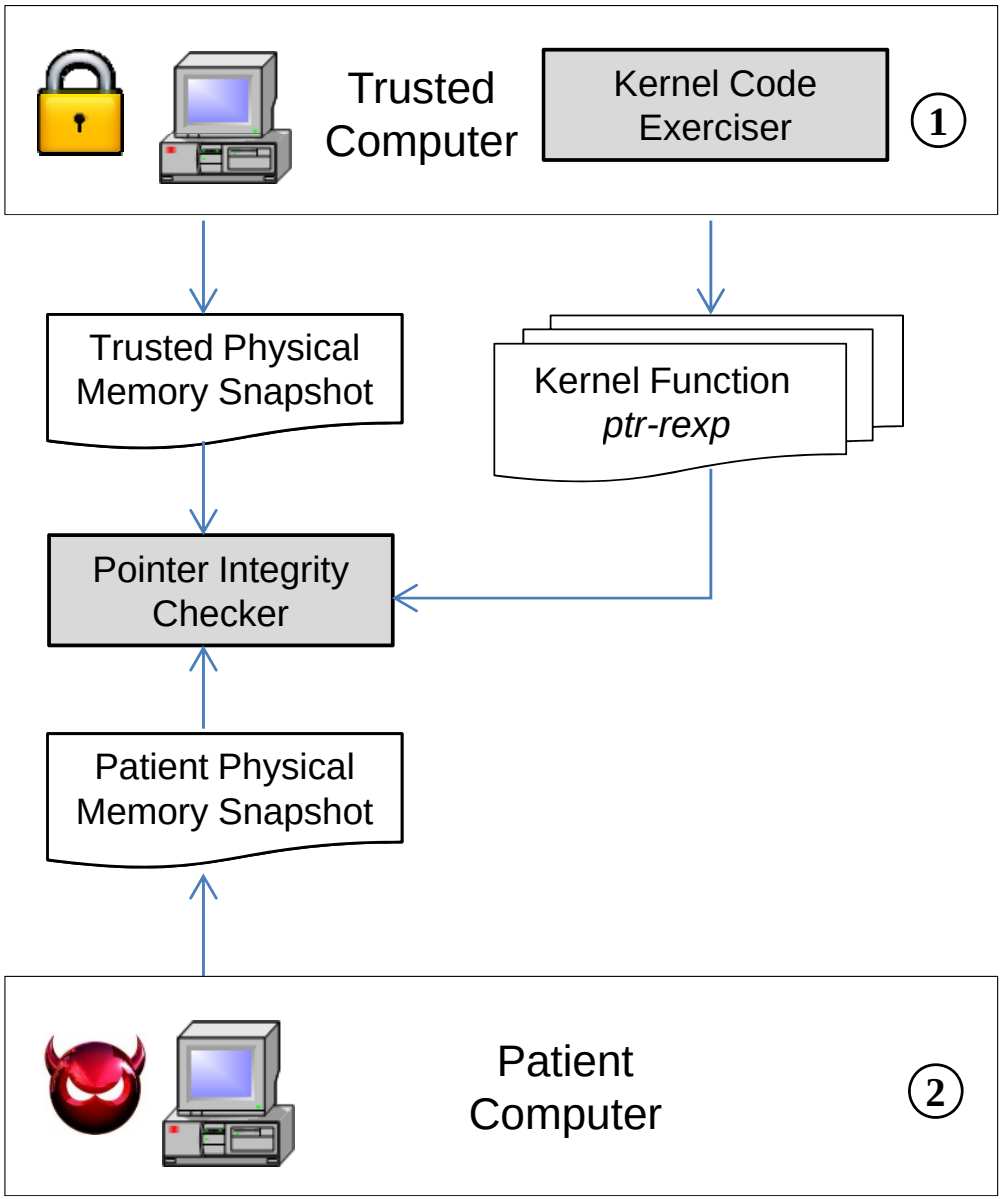
Function Pointer Integrity Checker: FPCK



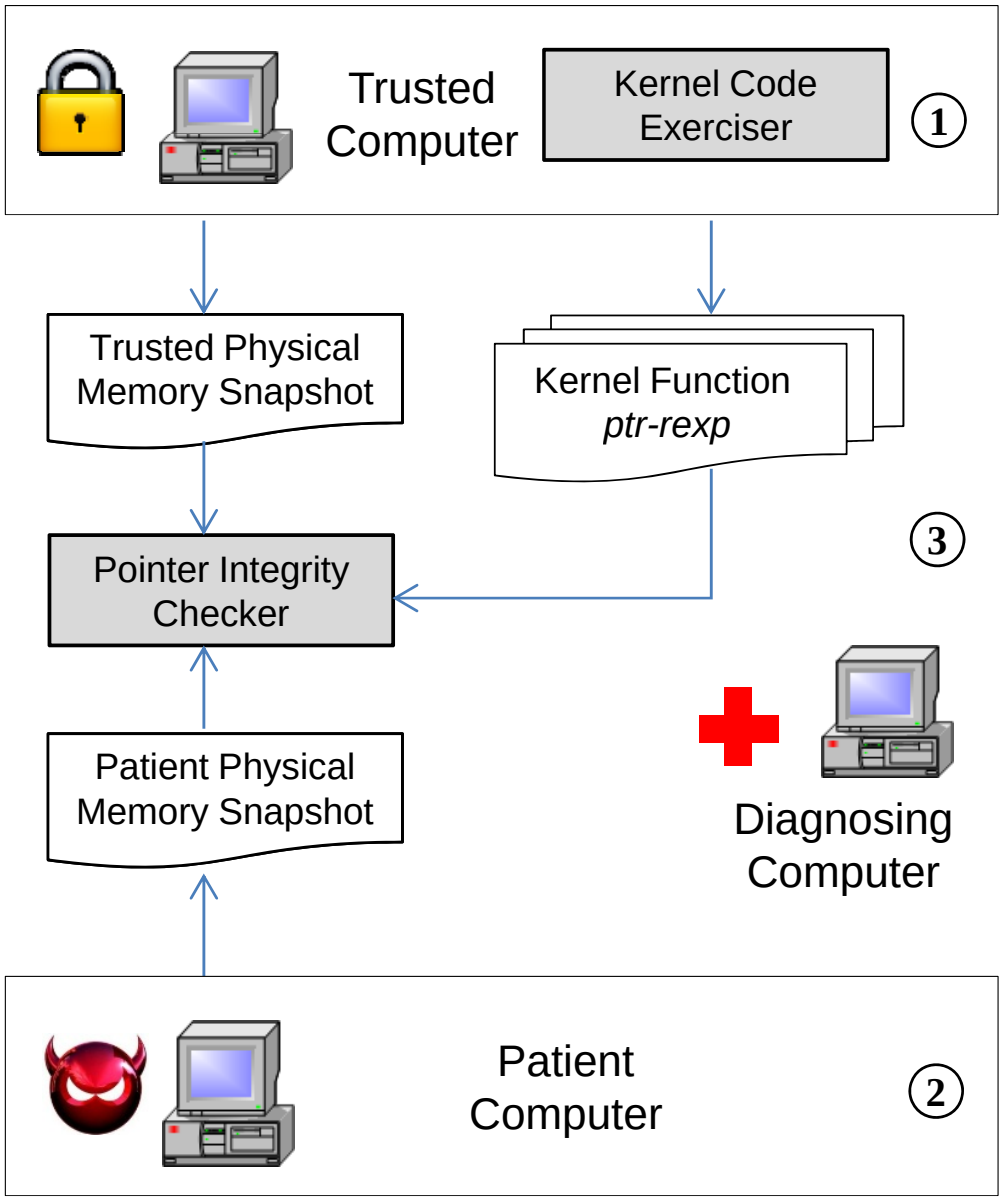
Function Pointer Integrity Checker: FPCK



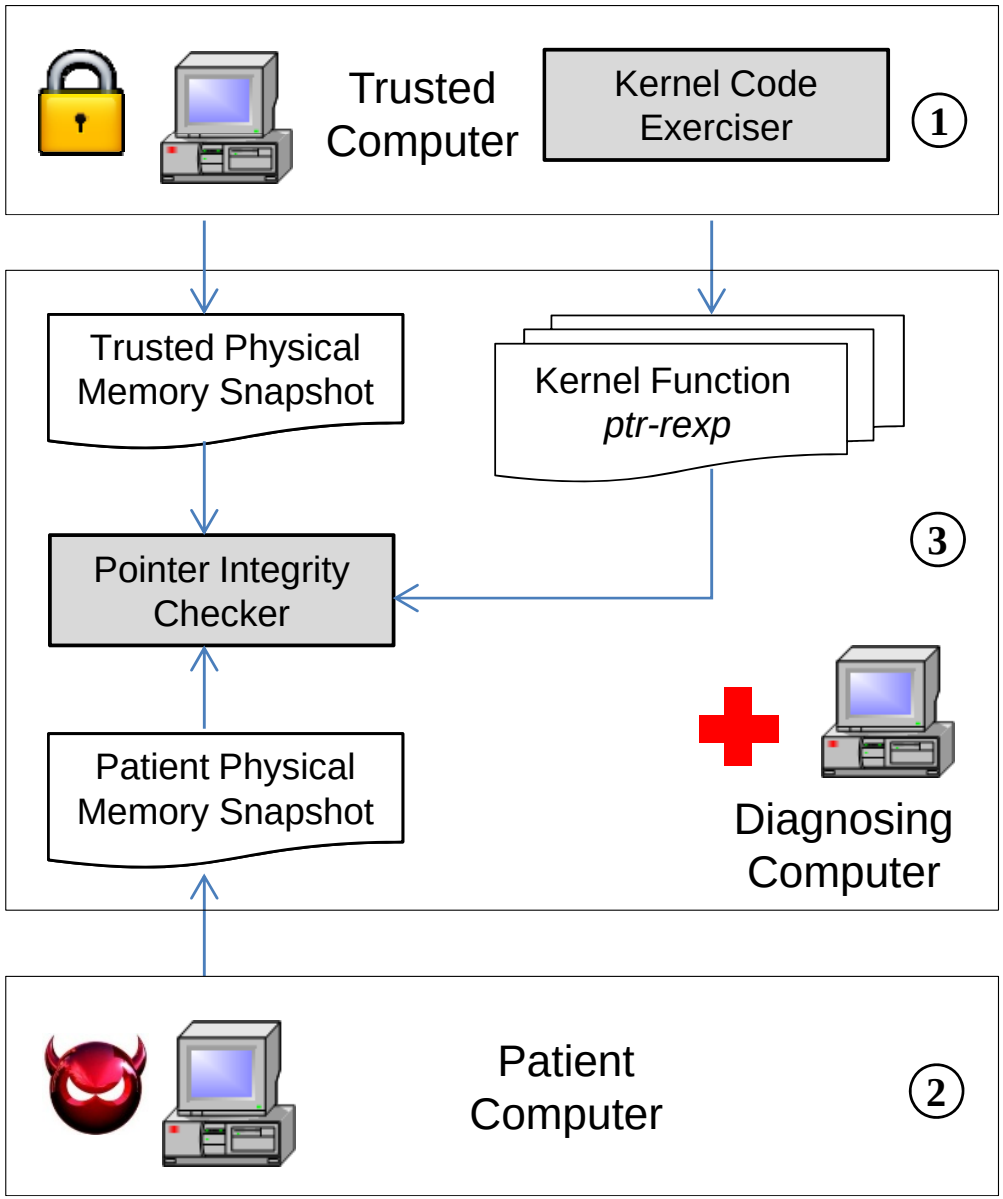
Function Pointer Integrity Checker: FPCK



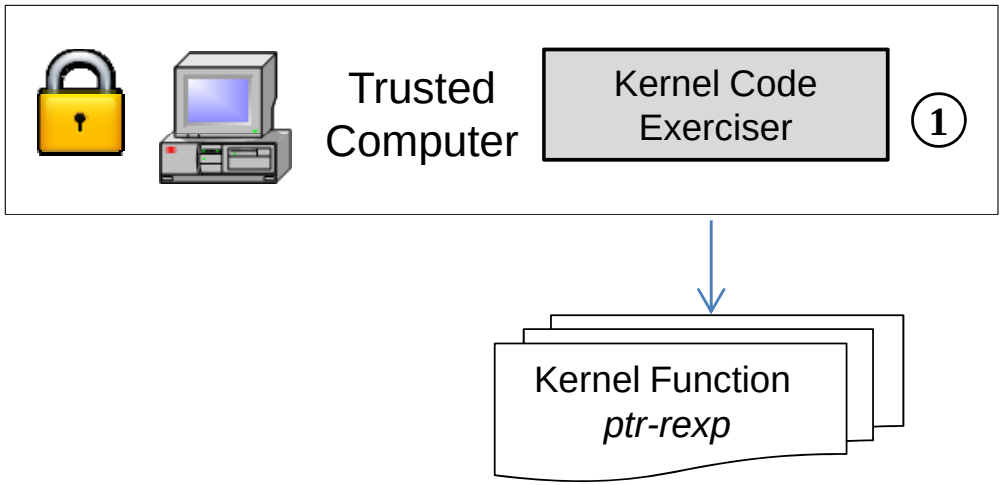
Function Pointer Integrity Checker: FPCK



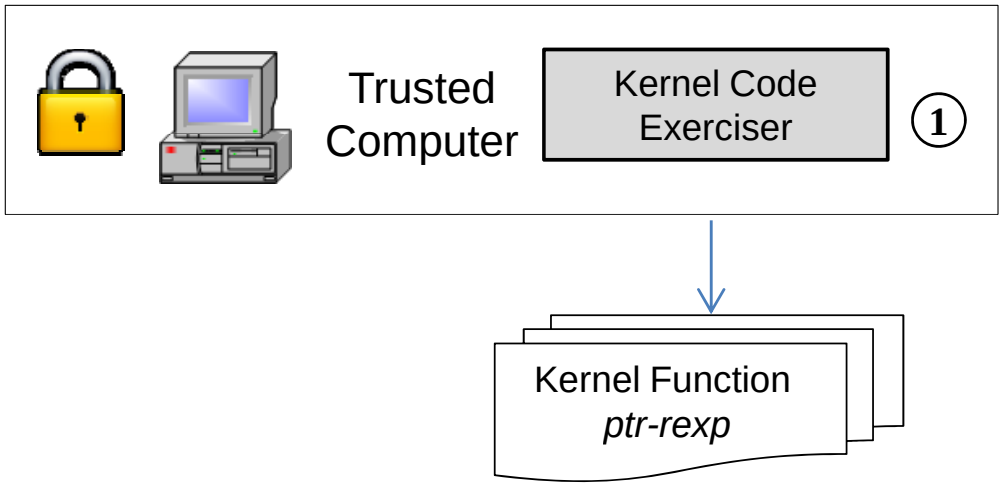
Function Pointer Integrity Checker: FPCK



Component-I: Kernel Code Exerciser



Component-I: Kernel Code Exerciser

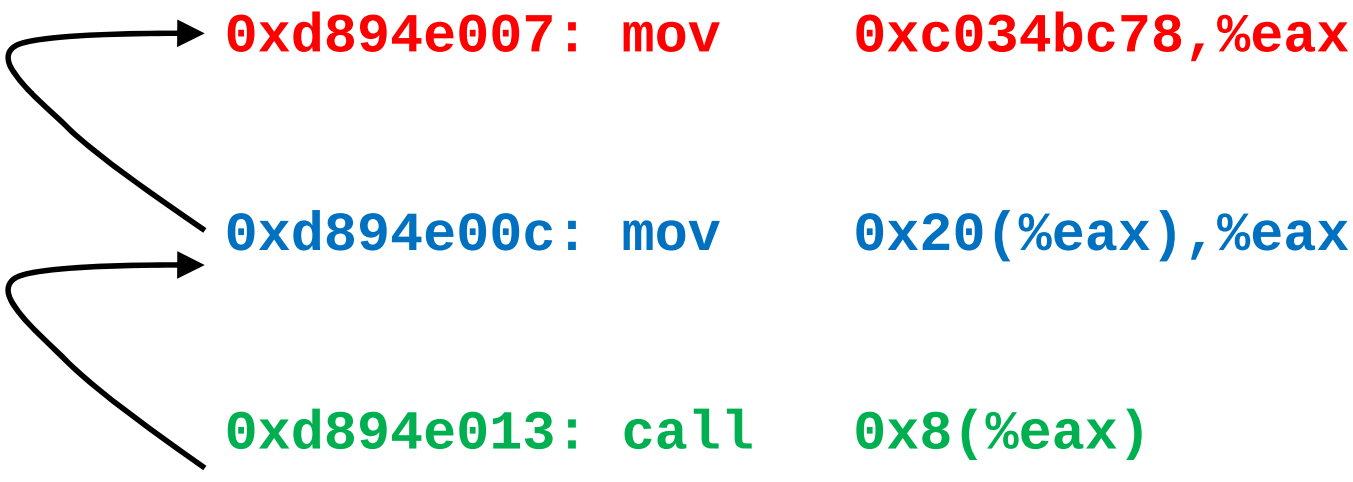



```
0xd894e007: mov  0xc034bc78,%eax
0xd894e00c: mov  0x20(%eax),%eax
0xd894e013: call 0x8(%eax)
```

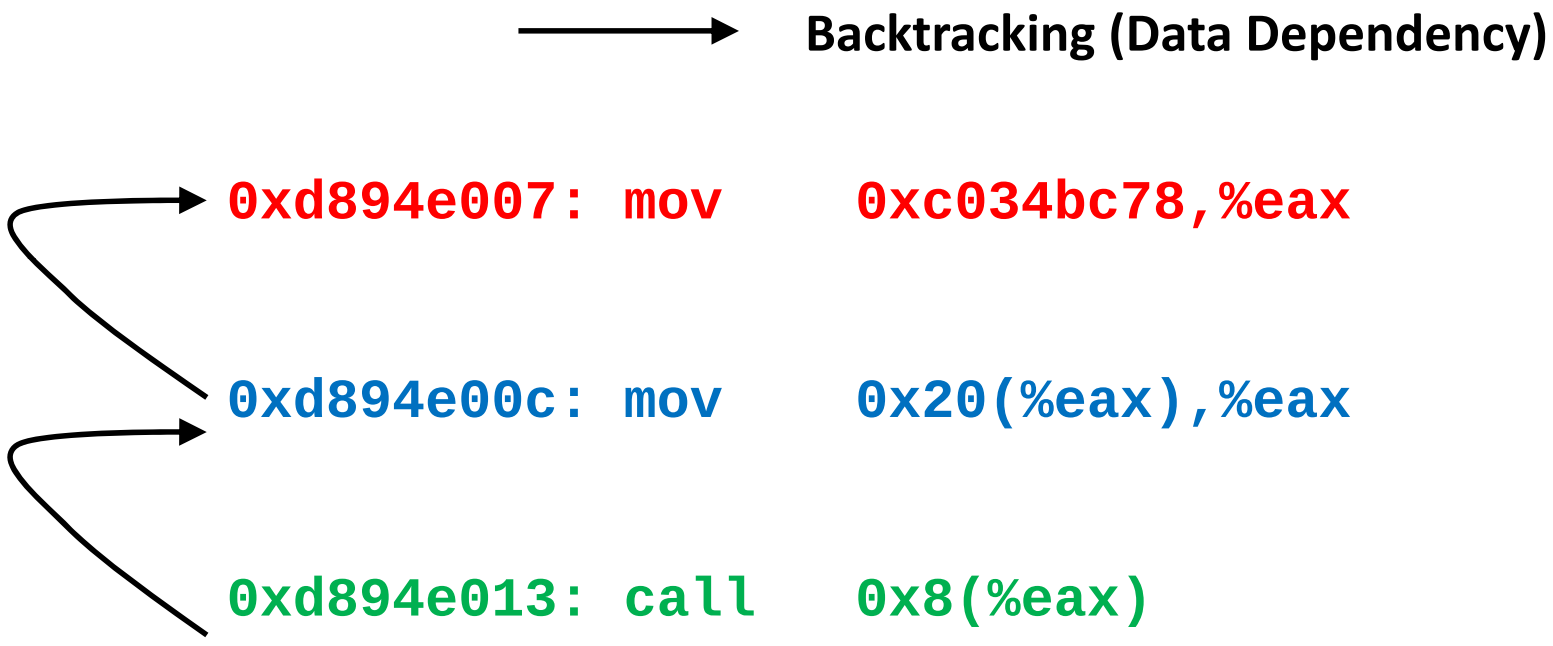
Key Observation: A Data Dependence Problem

```
0xd894e007: mov    0xc034bc78,%eax

0xd894e00c: mov    0x20(%eax),%eax

0xd894e013: call   0x8(%eax)
```

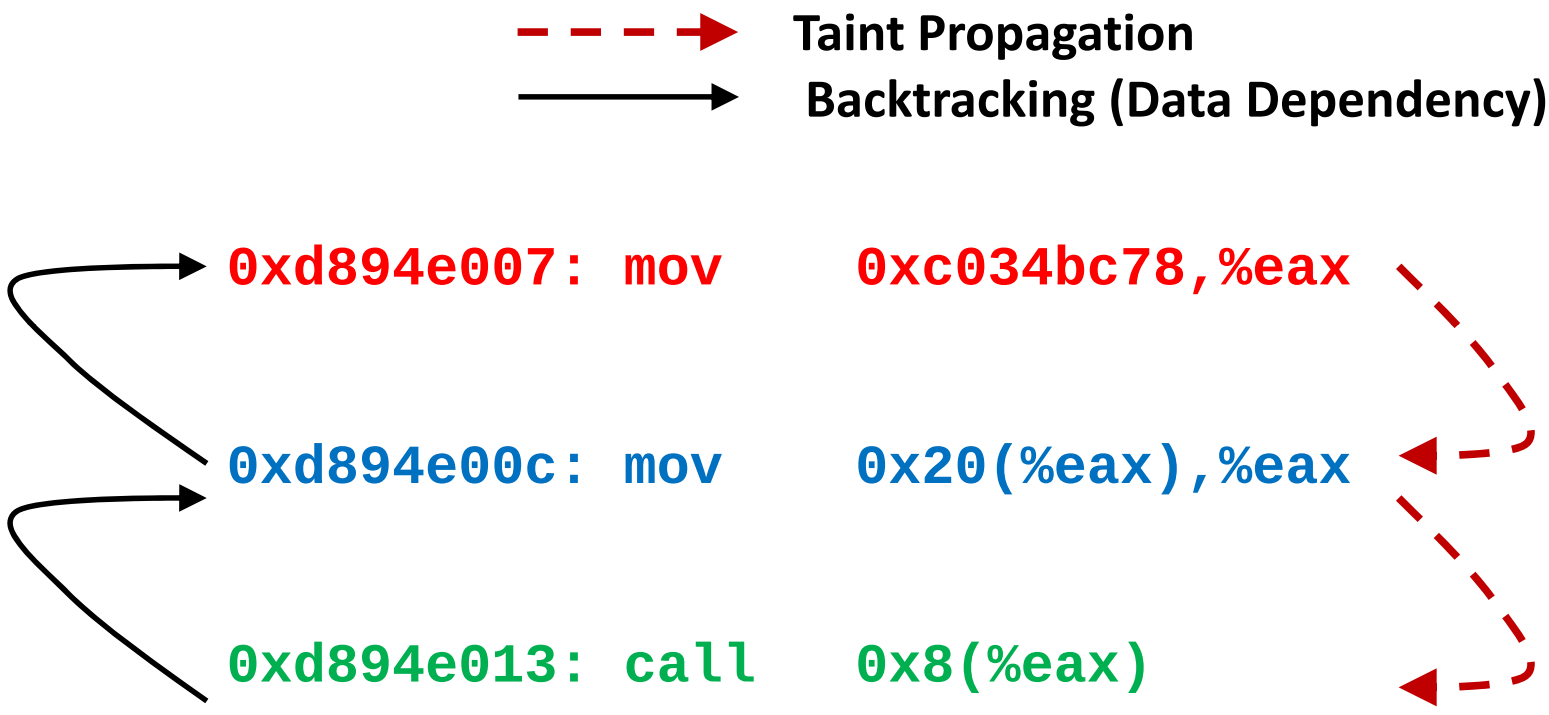
Key Observation: A Data Dependence Problem



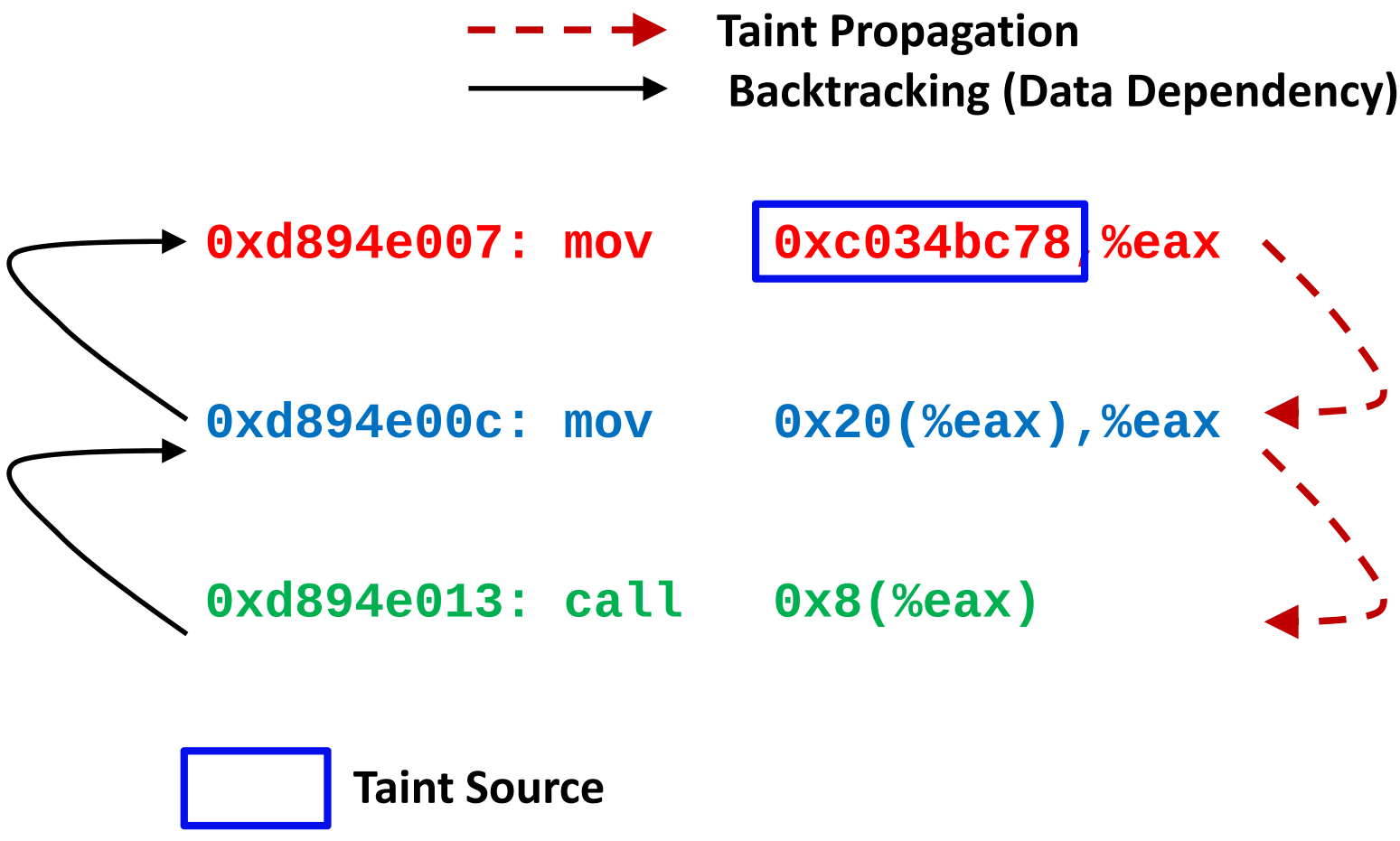
Key Observation: A Data Dependence Problem



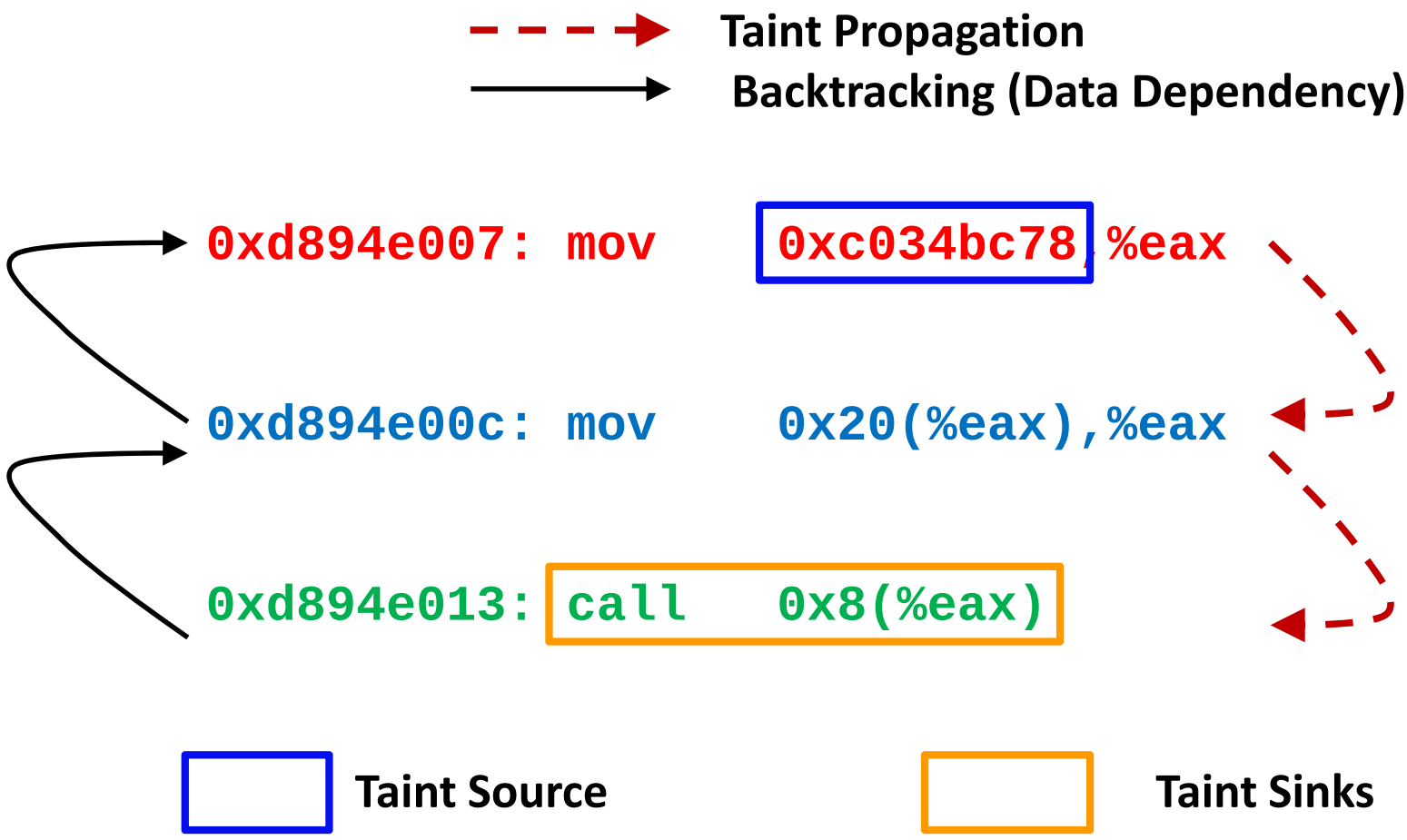
Key Observation: A Data Dependence Problem



Key Observation: A Data Dependence Problem



Key Observation: A Data Dependence Problem



Taint Sources

```
0xd894e007: mov  0xc034bc78,%eax
0xd894e00c: mov  0x20(%eax),%eax
0xd894e013: call 0x8(%eax)
```

Taint Sources

```
0xd894e007: mov  0xc034bc78,%eax
0xd894e00c: mov  0x20(%eax),%eax
0xd894e013: call 0x8(%eax)
```

What are the taint sources

1. An instruction which generates a data definition such as a register write (**M** $\rightarrow$ **R**) e.g.,
   - `mov 0xc034bc78, %eax`

Taint Sources

```
0xd894e007: mov  0xc034bc78,%eax
0xd894e00c: mov  0x20(%eax),%eax
0xd894e013: call 0x8(%eax)
```

What are the taint sources

1. An instruction which generates a data definition such as a register write (**M** $\rightarrow$ **R**) e.g.,
   - `mov 0xc034bc78, %eax`
2. An instruction which has a memory operand that involves a global memory address or its propagation, e.g.,
   - `mov 0x20(%eax), %eax` (**M** $\rightarrow$ **R**)
   - `mov %eax, %ebx` (Direct **R** $\rightarrow$ **R** won't generate new taint source)

Taint Propagations

```
0xd894e007: mov  0xc034bc78,%eax
0xd894e00c: mov  0x20(%eax),%eax
0xd894e013: call 0x8(%eax)
```

Taint Propagations

```
0xd894e007: mov  0xc034bc78,%eax
0xd894e00c: mov  0x20(%eax),%eax
0xd894e013: call 0x8(%eax)
```

When to propagate

1. **Data Movement Instructions**: The taint record will be flowed to the destination taint of the Reg or Mem operand, e.g., `mov 0x20(%eax), %eax`

Taint Propagations

```
0xd894e007: mov  0xc034bc78,%eax
0xd894e00c: mov  0x20(%eax),%eax
0xd894e013: call 0x8(%eax)
```

When to propagate

1. **Data Movement Instructions**: The taint record will be flowed to the destination taint of the Reg or Mem operand, e.g., `mov 0x20(%eax), %eax`
2. **Differences**: When propagating the taint record, if the source operand generates a new dependency, we will not propagate the original taint record, but rather propagate the newly generated taint.

Taint Propagations

```
0xd894e007: mov  0xc034bc78,%eax
0xd894e00c: mov  0x20(%eax),%eax
0xd894e013: call 0x8(%eax)
```

How to compute memory address: $r_1 + r_2 * scale + disp$

$$Displacement(BaseAddr, Index, Scale) = BaseAddr + Index \times Scale + Displacement$$

Taint Propagations

```
0xd894e007: mov  0xc034bc78,%eax
0xd894e00c: mov  0x20(%eax),%eax
0xd894e013: call 0x8(%eax)
```

How to compute memory address: $r_1 + r_2 * scale + disp$

$$Displacement(BaseAddr, Index, Scale) = BaseAddr + Index \times Scale + Displacement$$

A memory address could have two register dependencies:

- *BaseAddr* register
- *Index* register

Taint Sinks

```
0xd894e007: mov  0xc034bc78,%eax
0xd894e00c: mov  0x20(%eax),%eax
0xd894e013: call 0x8(%eax)
```

What are the taint sinks

1. Indirect call, e.g., `call 0x8%(eax)`
2. Indirect jump

Data structure type used in FPCK

type operand = Reg **of** name | Mem **of** addr
type shadow = (operand, PC) **Hashtbl**
type instMap = (PC, instRecord) **Hashtbl**
type instRecord = (I-semantics, taintOp, taintOp)
type I-semantics = Move | Binary | Call-Mem | …
type regTaint = (V, PC_p)
type taintOp =
MemOpTaint **of** regTaint × regTaint × scale × disp
| RegOpTaint **of** regTaint
| NoOpTaint

Shadow record creation and propagation

 Taint source  Taint sink

PC	Instructions	Program State	Shadow S[Operand] = PC
0xd894e007	mov 0xc034bc78,%eax	EAX=0xd7fee2e0	
0xd894e00c	mov 0x20(%eax), %eax	EAX=0xc028ea80	
0xd894e013	call 0x8(%eax)	EAX=0xc028ea80	

PC	*I-semantics*	Operand memOpTaint(regTaint,regTaint,Scal,Disp)			
		(V,PC)	(V,PC)	Sale	Disp

Shadow record creation and propagation

Taint source Taint sink

PC	Instructions	Program State	Shadow S[Operand] = PC
0xd894e007	mov 0xc034bc78,%eax	EAX=0xd7fee2e0	
0xd894e00c	mov 0x20(%eax), %eax	EAX=0xc028ea80	
0xd894e013	call 0x8(%eax)	EAX=0xc028ea80	

PC	*I-semantics*	Operand memOpTaint(regTaint,regTaint,Scal,Disp)			
		(V,PC)	(V,PC)	Sale	Disp
0xd894e007	MOV-M2R	(0,0)	(0,0)	0	0xc034bc78

Shadow record creation and propagation

Taint source Taint sink

PC	Instructions	Program State	Shadow S[Operand] = PC
0xd894e007	mov 0xc034bc78,%eax	EAX=0xd7fee2e0	S[EAX] = 0xd894e007
0xd894e00c	mov 0x20(%eax), %eax	EAX=0xc028ea80	
0xd894e013	call 0x8(%eax)	EAX=0xc028ea80	

PC	*I-semantics*	Operand memOpTaint(regTaint,regTaint,Scal,Disp)			
		(V,PC)	(V,PC)	Sale	Disp
0xd894e007	MOV-M2R	(0,0)	(0,0)	0	0xc034bc78

Shadow record creation and propagation

Taint source Taint sink

PC	Instructions	Program State	Shadow S[Operand] = PC
0xd894e007	mov 0xc034bc78,%eax	EAX=0xd7fee2e0	S[EAX] = 0xd894e007
0xd894e00c	mov 0x20(%eax), %eax	EAX=0xc028ea80	
0xd894e013	call 0x8(%eax)	EAX=0xc028ea80	

PC	*I-semantics*	Operand memOpTaint(regTaint,regTaint,Scal,Disp)			
		(V,PC)	(V,PC)	Sale	Disp
0xd894e007	MOV-M2R	(0,0)	(0,0)	0	0xc034bc78
0xd894e00c	MOV-M2R	(0,0xd89e007)	(0,0)	0	0x20

Shadow record creation and propagation

Taint source Taint sink

PC	Instructions	Program State	Shadow S[Operand] = PC
0xd894e007	mov 0xc034bc78,%eax	EAX=0xd7fee2e0	S[EAX] = 0xd894e007
0xd894e00c	mov 0x20(%eax), %eax	EAX=0xc028ea80	S[EAX] = 0xd894e00c
0xd894e013	call 0x8(%eax)	EAX=0xc028ea80	

PC	*I-semantics*	Operand memOpTaint(regTaint,regTaint,Scal,Disp)			
		(V,PC)	(V,PC)	Sale	Disp
0xd894e007	MOV-M2R	(0,0)	(0,0)	0	0xc034bc78
0xd894e00c	MOV-M2R	(0,0xd89e007)	(0,0)	0	0x20

Shadow record creation and propagation

Taint source Taint sink

PC	Instructions	Program State	Shadow S[Operand] = PC
0xd894e007	mov 0xc034bc78,%eax	EAX=0xd7fee2e0	S[EAX] = 0xd894e007
0xd894e00c	mov 0x20(%eax), %eax	EAX=0xc028ea80	S[EAX] = 0xd894e00c
0xd894e013	call 0x8(%eax)	EAX=0xc028ea80	S[0xc028ea88]=0xd894e013

PC	*I-semantics*	Operand memOpTaint(regTaint,regTaint,Scal,Disp)			
		(V,PC)	(V,PC)	Sale	Disp
0xd894e007	MOV-M2R	(0,0)	(0,0)	0	0xc034bc78
0xd894e00c	MOV-M2R	(0,0xd89e007)	(0,0)	0	0x20
0xd894e013	CALL-MEM	(0,0xd89e00c)	(0,0)	0	0x8

Shadow record creation and propagation

Taint source Taint sink

PC	Instructions	Program State	Shadow S[Operand] = PC
0xd894e007	mov 0xc034bc78,%eax	EAX=0xd7fee2e0	S[EAX] = 0xd894e007
0xd894e00c	mov 0x20(%eax), %eax	EAX=0xc028ea80	S[EAX] = 0xd894e00c
0xd894e013	call 0x8(%eax)	EAX=0xc028ea80	S[0xc028ea88]=0xd894e013

PC	*I-semantics*	Operand memOpTaint(regTaint,regTaint,Scal,Disp)			
		(V,PC)	(V,PC)	Sale	Disp
0xd894e007	MOV-M2R	(0,0)	(0,0)	0	0xc034bc78
0xd894e00c	MOV-M2R	(0,0xd89e007)	(0,0)	0	0x20
0xd894e013	CALL-MEM	(0,0xd89e00c)	(0,0)	0	0x8

Ptr-rexp generation algorithms

```
1: let rec resolve_data_path (p: PC) (v: value) (t: instMap): exp =
2:     if p = 0 then (Value(v)) else (
3:         let (sem, op1, op2) = Hashtbl.find t p in
4:             match sem with
5:                 Move -> resolve_op p op1 t
6:               | Binary -> BinOP(resolve_op p op1 t, resolve_op p op2 t)
7:               | Call-Mem -> resolve_op p op1 t
8:     )
9: and resolve_op (p: PC) (op: taintOP) (t: instMap): exp =:
10:     match op with
11:         memOpTaint ((v1, pc1), (v2, pc2), scale, disp) ->
12:                 let regValue1 = resolve_data_path pc1 v1 t in
13:                 let regValue2 = resolve_data_path pc2 v2 t in
14:                     DeRef (regValue1, regValue2, scale, disp)
15:       | regOpTaint (v3, pc3) -> ( resolve_data_path pc3 v3 t )
16:       | NoOpTaint -> Value (0)
```

Ptr-rexp Generation

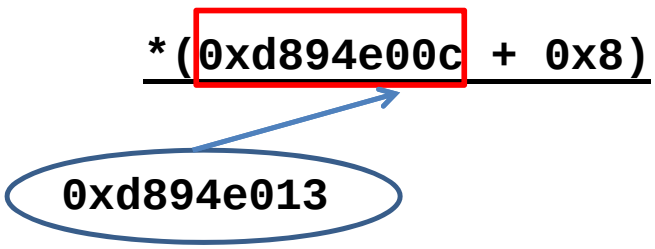
PC	*I-semantics*	Operand memOpTaint(regTaint,regTaint,Scal,Disp)			
		(V,PC)	(V,PC)	Sale	Disp
0xd894e007	MOV-M2R	(0,0)	(0,0)	0	0xc034bc78
0xd894e00c	MOV-M2R	(0,0xd89e007)	(0,0)	0	0x20
0xd894e013	CALL-MEM	(0,0xd89e00c)	(0,0)	0	0x8

Ptr-rexp Generation

0xd894e013

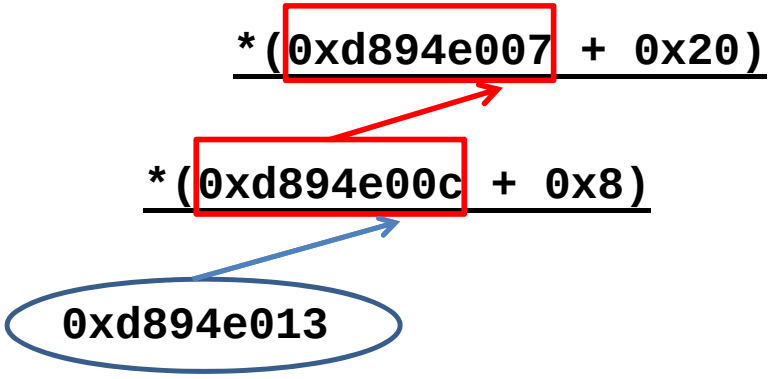
PC	*I-semantics*	Operand memOpTaint(regTaint,regTaint,Scal,Disp)			
		(V,PC)	(V,PC)	Sale	Disp
0xd894e007	MOV-M2R	(0,0)	(0,0)	0	0xc034bc78
0xd894e00c	MOV-M2R	(0,0xd89e007)	(0,0)	0	0x20
0xd894e013	CALL-MEM	(0,0xd89e00c)	(0,0)	0	0x8

Ptr-rexp Generation



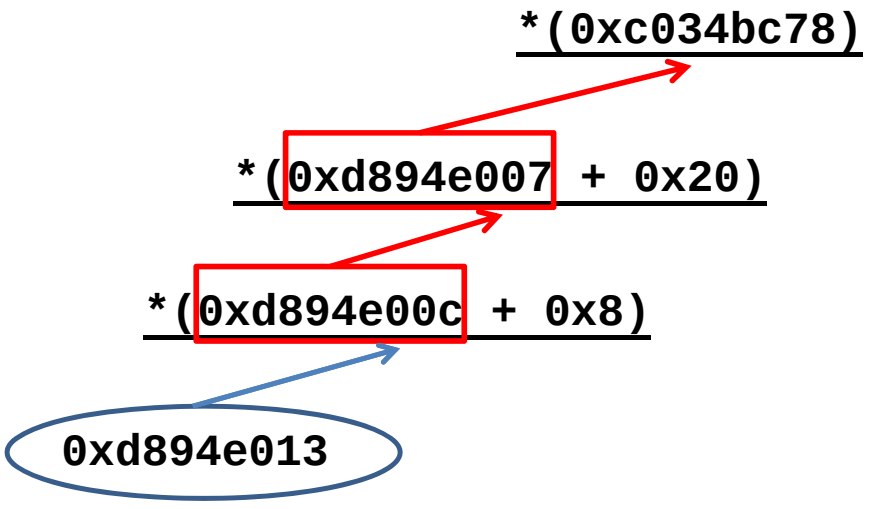
PC	*I-semantics*	Operand memOpTaint(regTaint,regTaint,Scal,Disp)			
		(V,PC)	(V,PC)	Sale	Disp
0xd894e007	MOV-M2R	(0,0)	(0,0)	0	0xc034bc78
0xd894e00c	MOV-M2R	(0,0xd89e007)	(0,0)	0	0x20
0xd894e013	CALL-MEM	(0,0xd89e00c)	(0,0)	0	0x8

Ptr-rexp Generation



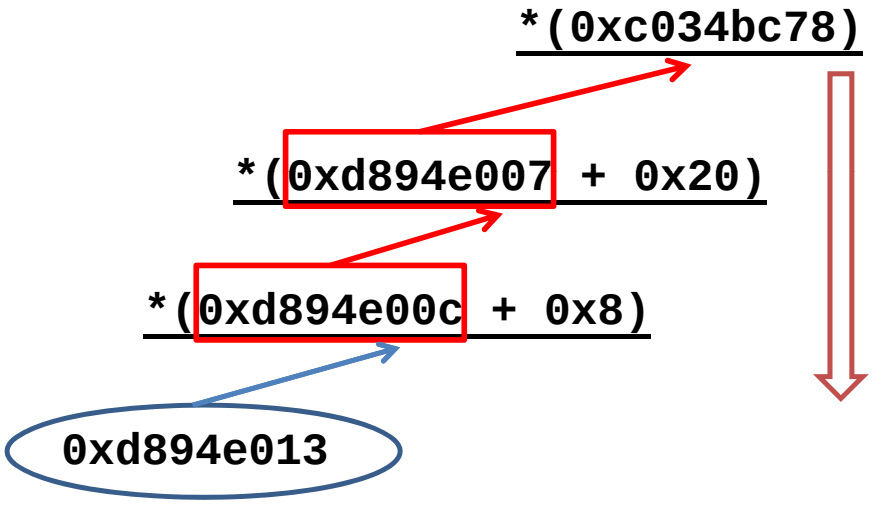
PC	*I-semantics*	Operand memOpTaint(regTaint,regTaint,Scal,Disp)			
		(V,PC)	(V,PC)	Sale	Disp
0xd894e007	MOV-M2R	(0,0)	(0,0)	0	0xc034bc78
0xd894e00c	MOV-M2R	(0,0xd89e007)	(0,0)	0	0x20
0xd894e013	CALL-MEM	(0,0xd89e00c)	(0,0)	0	0x8

Ptr-rexp Generation



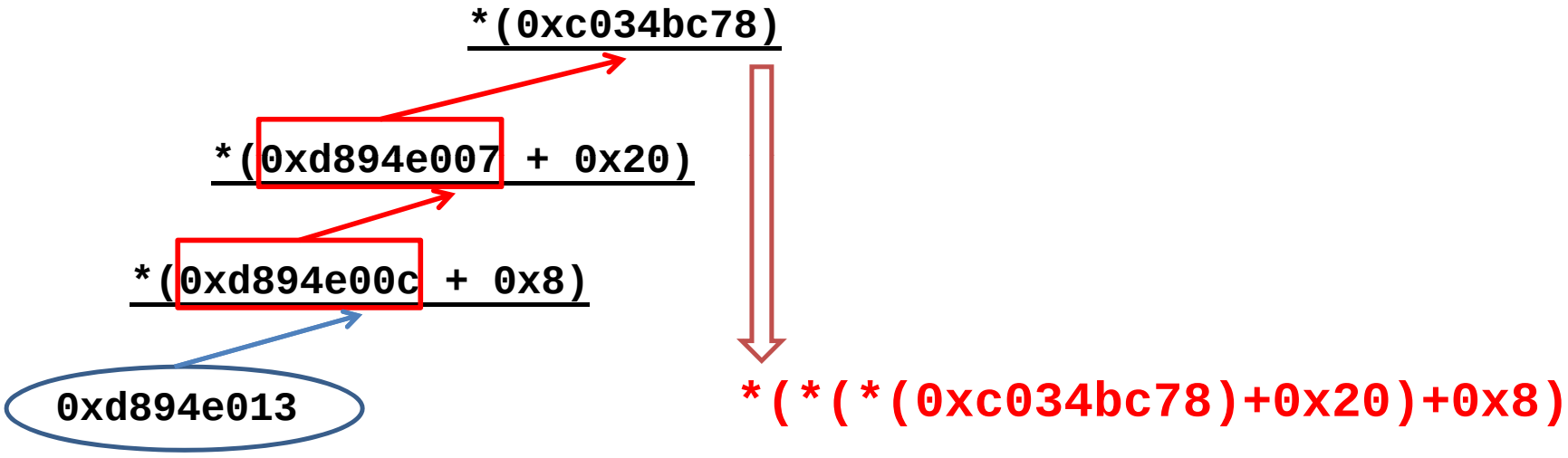
PC	*I-semantics*	Operand memOpTaint(regTaint,regTaint,Scal,Disp)			
		(V,PC)	(V,PC)	Sale	Disp
0xd894e007	MOV-M2R	(0,0)	(0,0)	0	0xc034bc78
0xd894e00c	MOV-M2R	(0,0xd89e007)	(0,0)	0	0x20
0xd894e013	CALL-MEM	(0,0xd89e00c)	(0,0)	0	0x8

Ptr-rexp Generation



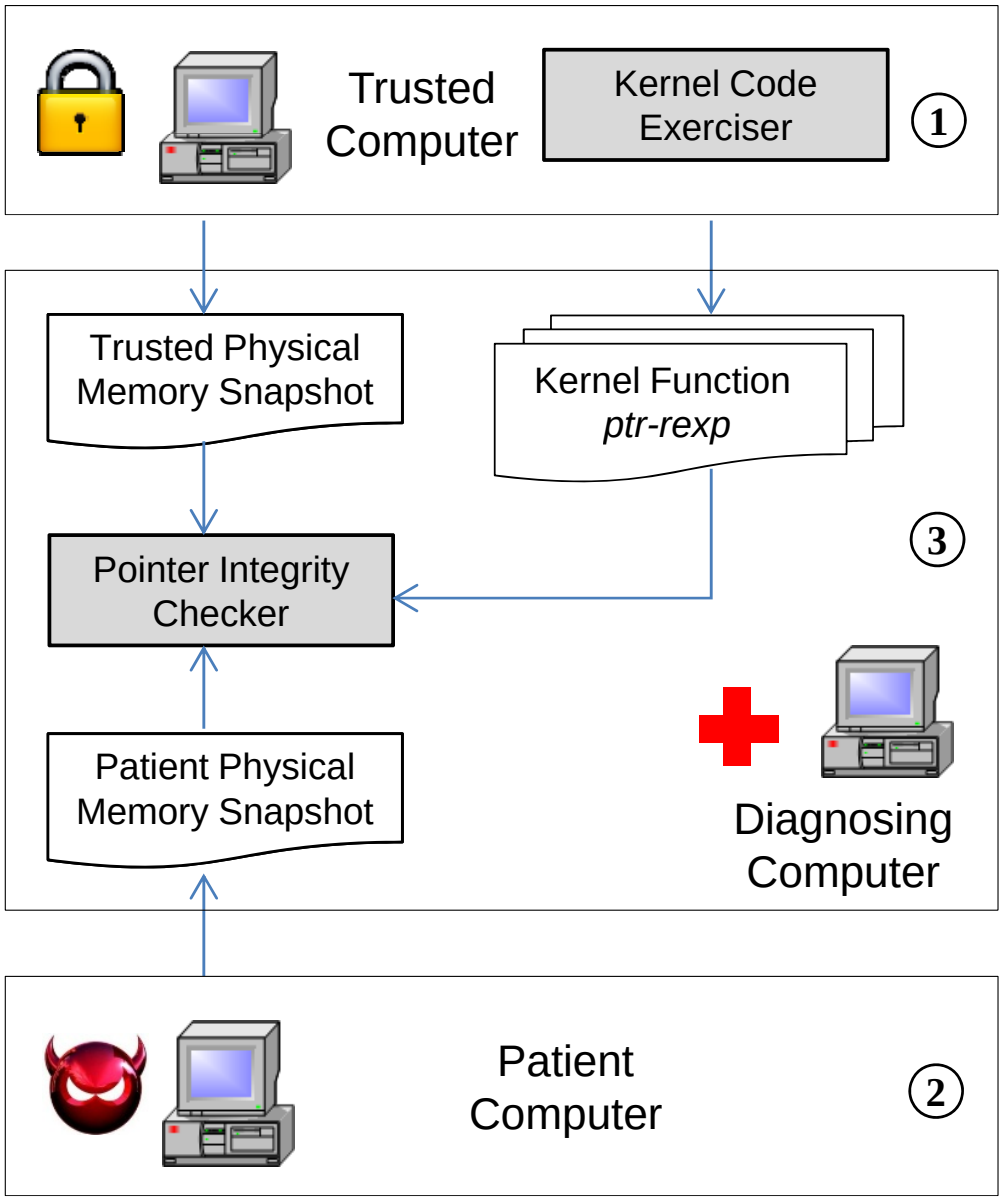
PC	*I-semantics*	Operand memOpTaint(regTaint,regTaint,Scal,Disp)			
		(V,PC)	(V,PC)	Sale	Disp
0xd894e007	MOV-M2R	(0,0)	(0,0)	0	0xc034bc78
0xd894e00c	MOV-M2R	(0,0xd89e007)	(0,0)	0	0x20
0xd894e013	CALL-MEM	(0,0xd89e00c)	(0,0)	0	0x8

Ptr-rexp Generation



PC	*I-semantics*	Operand memOpTaint(regTaint,regTaint,Scal,Disp)			
		(V,PC)	(V,PC)	Sale	Disp
0xd894e007	MOV-M2R	(0,0)	(0,0)	0	0xc034bc78
0xd894e00c	MOV-M2R	(0,0xd89e007)	(0,0)	0	0x20
0xd894e013	CALL-MEM	(0,0xd89e00c)	(0,0)	0	0x8

Component-II: Pointer Integrity Checker



Pointer integrity checker

1. **Direct Value Comparison for Core Kernel Code**
   - Core kernel Code address is static
2. **Direct Target Comparison for Kernel Modules**
   - Functions in dynamically loaded kernel modules may be loaded to different memory addresses.
   - Directly compare the code page of target function body, but exclude the relocated memory address operand (which is specified in the relocation tables)

The number of exercised ptr-rexp

Kernel Version	Call-MEM	Call-REG	Jmp-MEM	Jmp-REG	$\sum$
2.6.08	1234	155	250	0	1639
2.6.13	1175	141	257	11	1584
2.6.24	1237	474	231	0	1942
2.6.28	1182	423	273	0	1878
2.6.30	1262	456	282	0	2000
2.6.32	1284	365	232	0	1881
2.6.33	1284	366	227	0	1877
2.6.34	1286	360	245	0	1891
2.6.35	1239	352	239	0	1830
2.6.38	1213	375	234	15	1837
3.0.0	1394	451	276	29	2150
Average	1254	398	250	5	1907

Effectiveness of Testing w/ Linux Kernel Rootkits

Rootkit	Symbol Name of the Pointer	Trusted Value	Hijacked Value	|C|
override	module->init	-	0xd0923ad6	2
	moduel->exit	-	0xd0923af7	2
	sys_read	0xc0144d27	0xd092343c	1
	sys_chdir	0xc0143ced	0xd0923001	1
	sys_getuid	0xc011f59c	0xd09232ce	1
	sys_geteuid	0xc011f5ac	0xd09232f1	1
	sys_getdents64	0xc0154292	0xd0923314	1
Synapsys-0.4	module->init	-	0xd09267e8	2
	module->exit	-	0xd0926896	2
	sys_fork	0xc010488a	0xd092651e	1
	sys_write	0xc0144d8a	0xd09265f6	1
	sys_open	0xc014444c	0xd0926000	1
	sys_kill	0xc0121fa5	0xd09264c5	1
	sys_clone	0xc01048a4	0xd092657f	1
	sys_getdents	0xc0154082	0xd09265e0	1
	sys_getuid	0xc011f59c	0xd09263f9	1
kbdv3	module->init	-	0xd091b1aa	2
	module->exit	-	0xd091b215	2
	sys_utime	0xc0143970	0xd091b000	1
	sys_getuid	0xc011f59c	0xd091b142	1
	sys_utimes	0xc0143b84	0xd091b097	1
	sys_read	0xc0144d27	0xce271000	1
	sys_open	0xc014444c	0xcdde6000	1

Effectiveness of Testing w/ Linux Kernel Rootkits

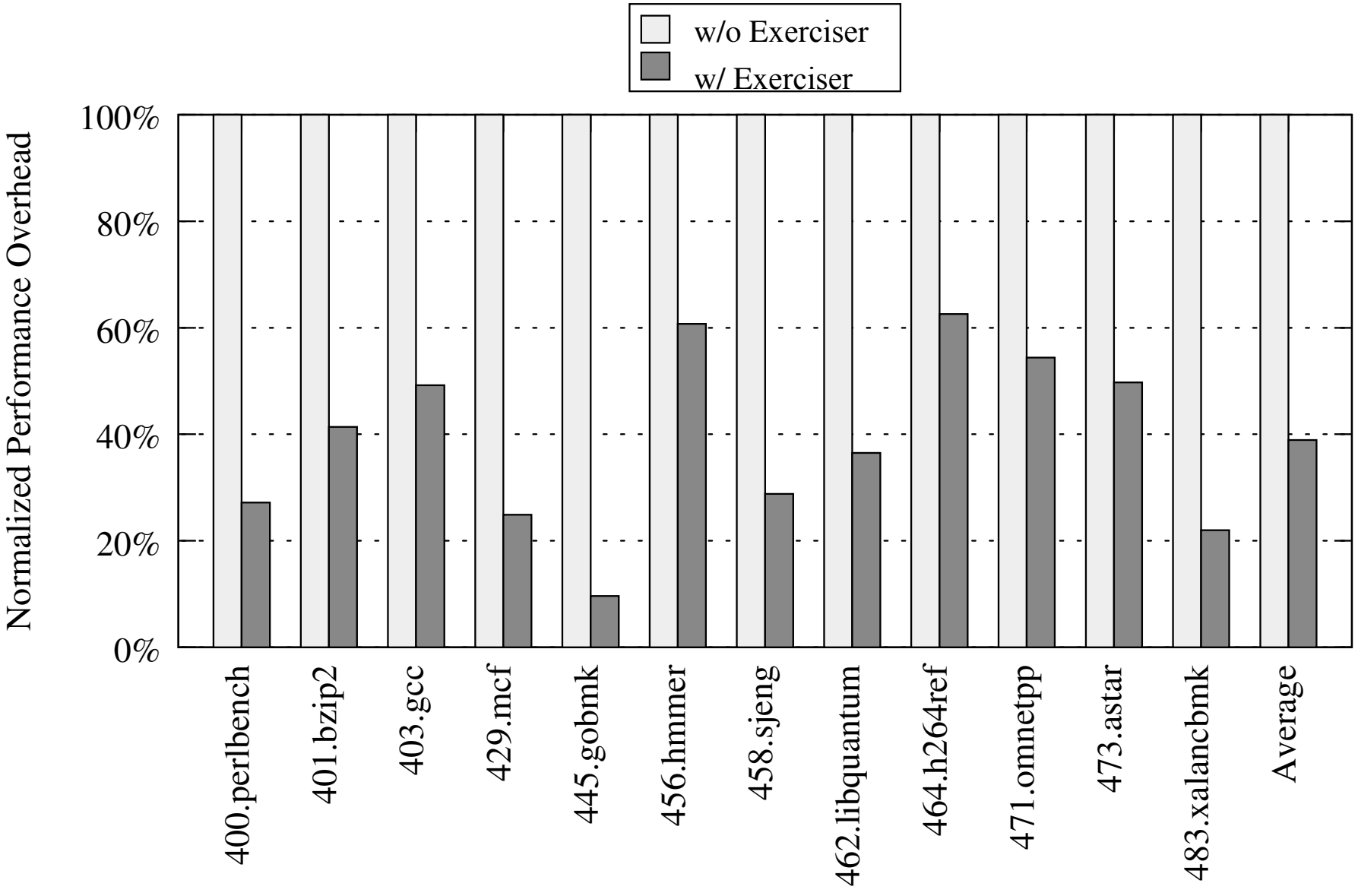
Rootkit	Symbol Name of the Pointer	Trusted Value	Hijacked Value	|C|
phalanx-b6	sys_read	0xc0144d27	0xce271000	1
	sys_open	0xc014444c	0xcdde6000	1
	sys_newlstat	0xc014c7ad	0xcdde3000	1
	sys_lstat64	0xc014c9a8	0xcdde2000	1
	tcp4_seq_show	0xc022be91	0xcdde5000	1
adore-2.6	module->init	-	0xd8985000	2
	module->exit	-	0xd897f9b4	2
	ext3.ext3_readdir	dynamic	0xdd97f774	6
	do_sync_write	0xc0144bb0	0xd897f8a4	5
	proc_root_readdir	0xc016b608	0xd897f477	6
	proc_root_lookup	0xc016b5ba	0xd897f13e	6
rkit-1.01	module->init	-	0xd091b05d	2
	module->exit	-	0xd091b097	2
	sys_setuid	0xc0123209	0xd091b000	1
suckit-2	idt enty 0x80	0xc0105f68	0xcc8c0906	1
hookswrite	module->init	-	0xd08c3000	2
	module->exit	-	0xd0843216	2
	idt enty 0x80	0xc0105f68	0xd0843000	1
int3backdoor	module->init	-	0xd08a119c	2
	idt enty 0x3	0xc0106b48	0xd08a1000	1

Effectiveness of Testing w/ Linux Kernel Rootkits

Rootkit	Symbol Name of the Pointer	Trusted Value	Hijacked Value	|C|
phalanx-b6	sys_read	0xc0144d27	0xce271000	1
	sys_open	0xc014444c	0xcdde6000	1
	sys_newlstat	0xc014c7ad	0xcdde3000	1
	sys_lstat64	0xc014c9a8	0xcdde2000	1
	tcp4_seq_show	0xc022be91	0xcdde5000	1
adore-2.6	module->init	-	0xd8985000	2
	module->exit	-	0xd897f9b4	2
	ext3.ext3_readdir	dynamic	0xdd97f774	6
	do_sync_write	0xc0144bb0	0xd897f8a4	5
	proc_root_readdir	0xc016b608	0xd897f477	6
	proc_root_lookup	0xc016b5ba	0xd897f13e	6
rkit-1.01	module->init	-	0xd091b05d	2
	module->exit	-	0xd091b097	2
	sys_setuid	0xc0123209	0xd091b000	1
suckit-2	idt enty 0x80	0xc0105f68	0xcc8c0906	1
hookswrite	module->init	-	0xd08c3000	2
	module->exit	-	0xd0843216	2
	idt enty 0x80	0xc0105f68	0xd0843000	1
int3backdoor	module->init	-	0xd08a119c	2
	idt enty 0x3	0xc0106b48	0xd08a1000	1

$current->fs\_struct->dentry->inode->file\_operations->readdir$

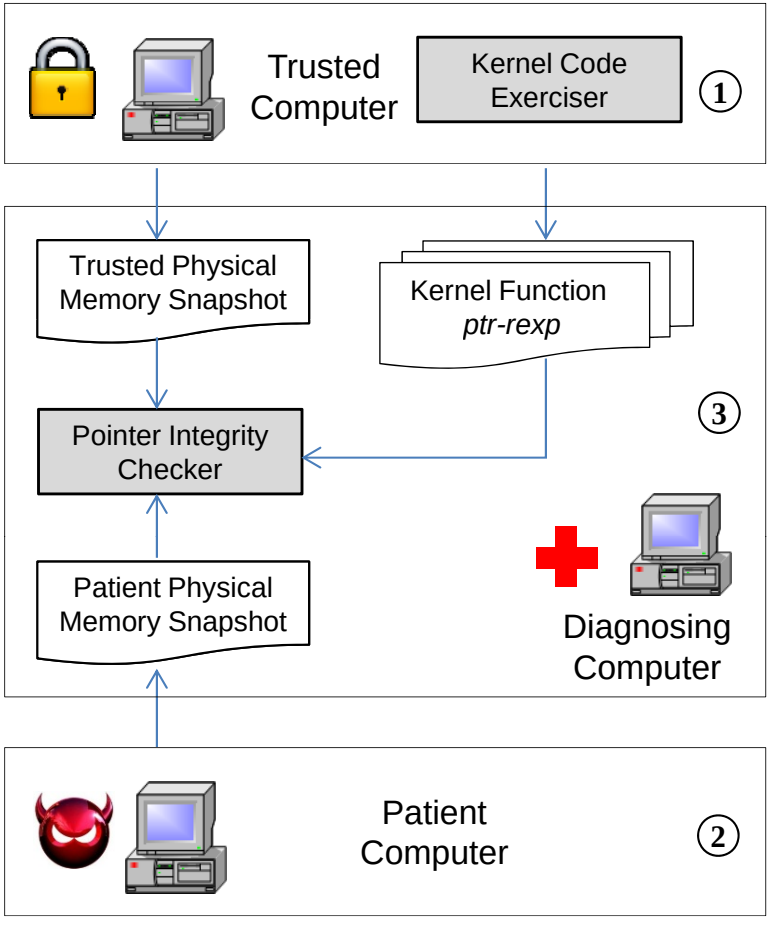
Performance Evaluation



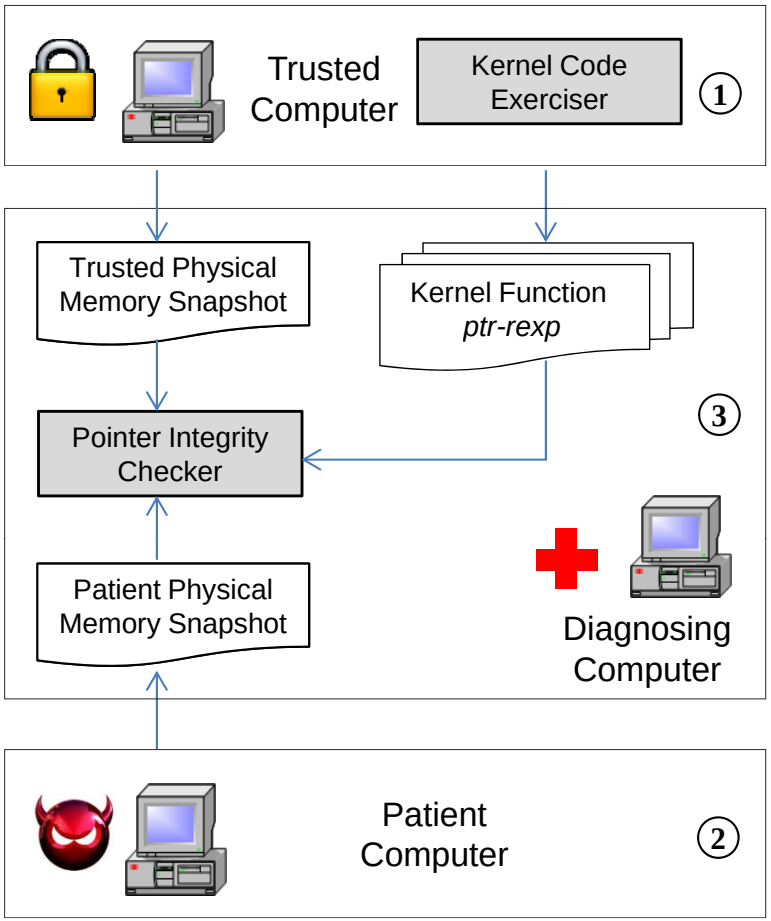
Related Works

Systems	wo/ Source Code	wo/ Kernel Symbols	wo/ Relocation Table	wo/ Kernel Data Structure	wo/ In-VM Assistance	wo/ Pattern Matching	Continuous Monitoring	Snapshot-based	Profiling	Detection
SBCFI [PH07]	✗	✓	✓	✗	✓	✓	✗	✓	✗	✓
HookFinder [YLS08]	✓	✓	✓	✗	✗	✓	✓	✗	✓	✗
HookMap [WJCW08]	✓	✗	✓	✓	✓	✓	✓	✗	✓	✗
Gibraltar [BGI08]	✗	✓	✓	✗	✓	✓	✗	✓	✓	✓
K-Tracer [LSL09]	✓	✗	✓	✗	✓	✓	✓	✗	✓	✗
Poker [RJX09]	✗	✗	✓	✗	✓	✓	✗	✓	✓	✗
KOP [CCL+09]	✗	✓	✓	✗	✓	✓	✗	✓	✓	✓
HookSafe [WJCN09]	✗	✗	✓	✗	✗	✓	✓	✗	✗	✓
HookScout [YPHS10]	✓	✓	✗	✓	✗	✓	✓	✗	✓	✓
LiveDM [RRXJ10]	✗	✗	✓	✗	✓	✓	✓	✗	✓	✓
OSck [HDK+11]	✗	✗	✓	✗	✓	✓	✗	✓	✓	✓
HUKO [XTL11]	✓	✗	✓	✓	✗	✓	✓	✗	✗	✓
MAS [CPXC12]	✗	✓	✓	✗	✓	✓	✗	✓	✓	✓
BlackSheep [BSKV12]	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓
HookLocator [ARZR13]	✓	✓	✗	✗	✓	✗	✗	✓	✓	✓
FPCK	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓

Conclusion


Trusted Computer
Kernel Code Exerciser
1
Trusted Physical Memory Snapshot
Kernel Function
ptr-rexp
3
Pointer Integrity Checker
Patient Physical Memory Snapshot
Diagnosing Computer
Patient Computer
2

Conclusion



- FPCK is a binary exclusive approach for automatically locating kernel function pointers.
- We developd a binary exclusive out-of-VM approach to automatically check the integrity of kernel function pointers hijacked by kernel malware.

Limitations and future works

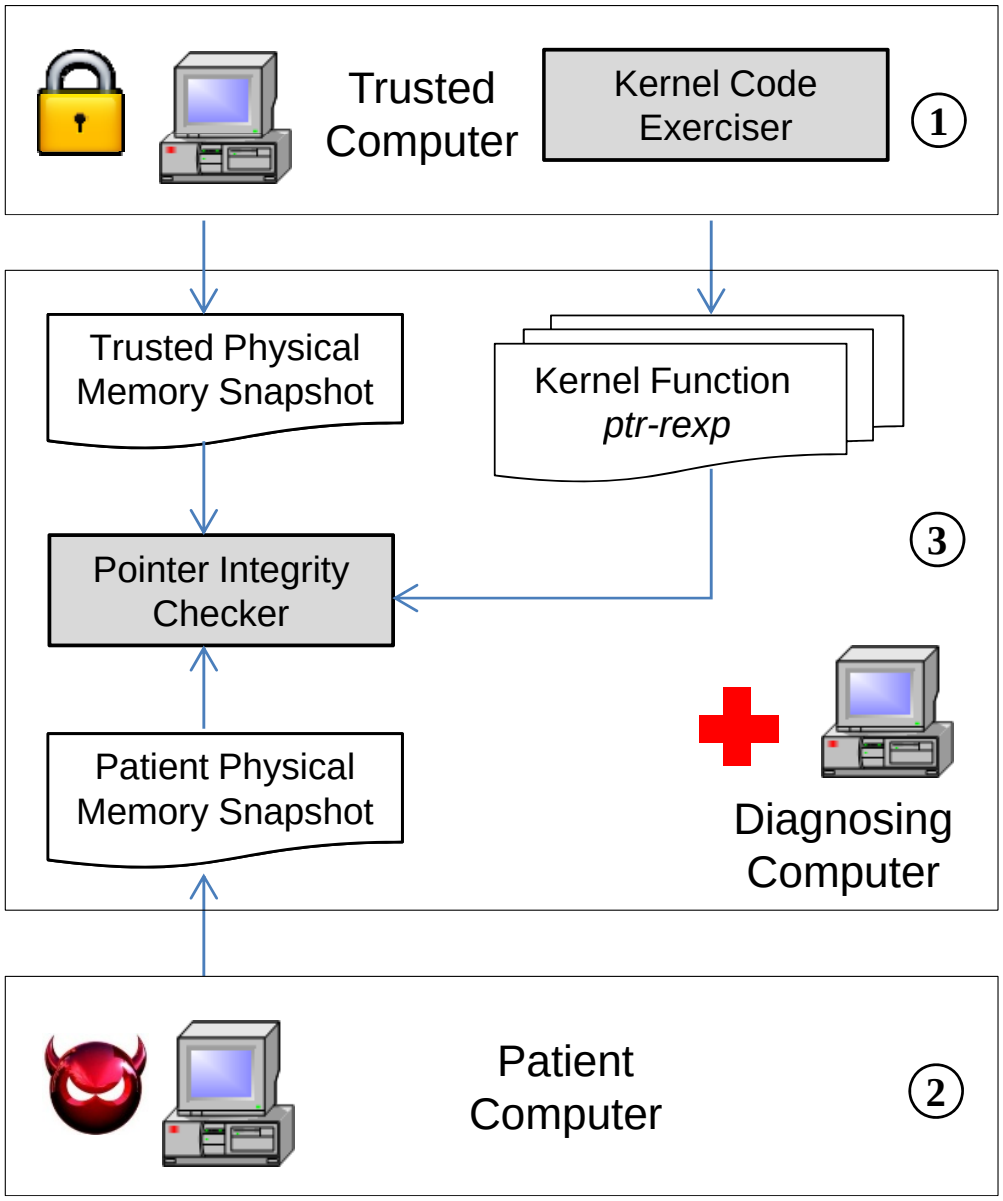
- Handling temporary pointer.
- $\Rightarrow$ Recognize the execution context, and associate the context to these temporary function pointers.

Limitations and future works

- Handling temporary pointer.
- $\Rightarrow$ Recognize the execution context, and associate the context to these temporary function pointers.

- Addressing other Attacks.
- $\Rightarrow$ Evaluate detecting the data-only rootkits.

Q&A


Trusted Computer
Kernel Code Exerciser
1
Trusted Physical Memory Snapshot
Kernel Function ptr-rexp
3
Pointer Integrity Checker
Patient Physical Memory Snapshot
Diagnosing Computer
Patient Computer
2

References I

I. Ahmed, G. Richard, A. Zoranic, and V. Roussev, Integrity checking of function pointers in kernel pools via virtual machine introspection, Proc. of th 16th Information Security Conference (ISC), 2013.

Arati Baliga, Vinod Ganapathy, and Liviu Iftode, Automatic inference and enforcement of kernel data structure invariants, Proceedings of the 2008 Annual Computer Security Applications Conference (ACSAC'08) (Anaheim, California), December 2008, pp. 77–86.

Antonio Bianchi, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna, Blacksheep: detecting compromised hosts in homogeneous crowds, Proceedings of the 2012 ACM conference on Computer and communications security, CCS'12, 2012.

Martim Carbone, Weidong Cui, Long Lu, Wenke Lee, Marcus Peinado, and Xuxian Jiang, Mapping kernel objects to enable systematic integrity checking, The 16th ACM Conference on Computer and Communications Security (CCS'09) (Chicago, IL, USA), 2009, pp. 555–565.

Weidong Cui, Marcus Peinado, Zhilei Xu, and Ellick Chan, Tracking rootkit footprints with a practical memory analysis system, Proceedings of USENIX Security Symposium, August 2012.

Owen S. Hofmann, Alan M. Dunn, Sangman Kim, Indrajit Roy, and Emmett Witchel, Ensuring operating system kernel integrity with osck, Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems (Newport Beach, California, USA), ASPLOS XVI, ACM, 2011, pp. 279–290.

Andrea Lanzi, Monirul I. Sharif, and Wenke Lee, K-tracer: A system for extracting kernel malware behavior, Proceedings of the 2009 Network and Distributed System Security Symposium, San Diego, California, USA,, 2009.

References II


Nick L. Petroni, Jr. and Michael Hicks, Automated detection of persistent kernel control-flow attacks, Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS'07) (Alexandria, Virginia, USA), ACM, October 2007, pp. 103–115.


Ryan Riley, Xuxian Jiang, and Dongyan Xu, Multi-aspect profiling of kernel rootkit behavior, Proceedings of the 4th ACM European conference on Computer systems (Nuremberg, Germany), EuroSys '09, 2009, pp. 47–60.


Junghwan Rhee, Ryan Riley, Dongyan Xu, and Xuxian Jiang, Kernel malware analysis with un-tampered and temporal views of dynamic kernel memory, Proceedings of the 13th International Symposium of Recent Advances in Intrusion Detection (RAID 2010) (Ottawa, Canada), September 2010.


Zhi Wang, Xuxian Jiang, Weidong Cui, and Peng Ning, Countering kernel rootkits with lightweight hook protection, Proceedings of the 16th ACM conference on Computer and communications security (Chicago, Illinois, USA), CCS '09, 2009, pp. 545–554.


Zhi Wang, Xuxian Jiang, Weidong Cui, and Xinyuan Wang, Countering persistent kernel rootkits through systematic hook discovery, Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection (Cambridge, MA, USA), RAID '08, 2008, pp. 21–38.


Xi Xiong, Donghai Tian, and Peng Liu, Practical protection of kernel integrity for commodity os from untrusted extensions, Proceedings of the Network and Distributed System Security Symposium (NDSS) (San Diego, CA), 2011.


Heng Yin, Zhenkai Liang, and Dawn Song, Hookfinder: Identifying and understanding malware hooking behaviors, Proceedings of the Network and Distributed System Security Symposium, 2008.

References III

Heng Yin, Pongsin Poosankam, Steve Hanna, and Dawn Song, HookScout: Proactive binary-centric hook detection, Proceedings of Seventh Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA'10), July 2010.