APT & FORENSICS
CS546: COMPUTER SYSTEM SECURITY

Slides about BEEP in this lecture are borrowed from Prof. Kyu Hyung Lee at UGA.
ABOUT THE COURSE

• Instructor: Shiqing Ma, shiqing.ma AT rutgers.edu
• Location: SEC-203
• Time: Friday, 13:40 to 16:40
• TA: Fangda Han, fh199 AT cs.rutgers.edu
• https://www.cs.rutgers.edu/~sm2283/19fall/index.html
• Send emails starting with “[546]:”
A FEW NOTES

- 2 programming assignments
  - System security
  - Adversarial sample

- Grading
  - 10% added for term-long project. Reminder: Oct. 4, you are expected to submit the proposal
  - 10% for in-class presentation, 2*5%
  - One hour each
  - You can talk about your self-proposed project
TIMELINE

- Oct. 11: in class proposal presentation
- Oct. 4: proposal deadline
- Sep. 27: deadline for deciding your topic and get approved by me
  - Also deadline for 2 papers to be presented
- Sep. 20: deadline for proposing initial ideas
QUESTIONS?

Shiqing Ma, Rutgers University
STUXNET (2010)

• Olympic operation
  • Targets Iran
  • Involves five different countries – that is where the name is from
    • USA, Israel, Dutch, France, German
  • Attack is believed to last for 6 years
  • Eventually, bring Iran back to negotiation desk
  • Complex, novel, remote, evolution, hidden, highly targeted
ADVANCED PERSISTENT THREAT (APT)

1. Attacker profiles the targeted institution through social engineering
2. Attacker sends well-crafted spear phishing emails
3. Target opens the malicious content in the phishing mail
4. Target system compromised
5. Target internal network system compromised
6. Data extraction to the APT Group
The targets of these assaults, which are very carefully chosen and researched, typically include large enterprises or governmental networks. The consequences of such intrusions are vast, and include:

- Intellectual property theft (e.g., trade secrets or patents)
- Compromised sensitive information (e.g., employee and user private data)
- The sabotaging of critical organizational infrastructures (e.g., database deletion)
- Total site takeovers
APT attacks differ from traditional web application threats, in that:

• They’re significantly more complex.
• They’re not hit and run attacks — once a network is infiltrated, the perpetrator remains in order to attain as much information as possible.
• They’re manually executed (not automated) against a specific mark and indiscriminately launched against a large pool of targets.
• They often aim to infiltrate an entire network, as opposed to one specific part.
APT ATTACKS

• Existing defense
  • Firewall, email filtering, end-point security product (e.g., anti-virus software), secure network configuration … …
  • Educated employees

• Have failed …
  • APTs are highly targeted: customized software, phishing emails … …
Forensic analysis refers to a detailed investigation for detecting and documenting the course, reasons, culprits, and consequences of a security incident or violation of rules of the organization or state laws.
Logging is a simple but effective technique
Record the dynamic information (ex. syscalls) during system execution

Logging is widely used
For attack investigation – Forensic analysis
For failure diagnosis – Execution replay
EXISTING LOGGING FACILITIES

- Windows
  - Event Tracing for Windows (ETW)
  - Performance logging
  - Many 3rd party logging facilities
- *NIX
  - Shipped with Audit
  - 3rd party facilities
ANALYZE THE ATTACK

Identify a source of the attack

Understand the damage to the victim system
PROVENANCE BY AUDIT LOGGING

- Audit logging system
  - Records important system events during system execution
  - Is a default kernel module in Linux, FreeBSD and MacOS
PROVENANCE BY AUDIT LOGGING

• Audit logs show
  • Event information

```
type=SYSCALL (02/10/14 16:28:20.782) : syscall=open exit=3 a0=912e98
type=PATH : name=/etc/file1 mode=12345 mod=file,744
ppid=2537 pid=2557 comm=vim exe=/bin/vim
uid=tom gid=tom euid=tom
```

< Audit log entry for a file open event >
PROVENANCE BY AUDIT LOGGING

• Audit logs show
  • Event information
  • File information

< Audit log entry for a file open event >

```plaintext
type=SYSCALL (02/10/14 16:28:20.782) : syscall=open exit=3 a0=912e98
type=PATH : name=/etc/file1 inode=12345 mod=file,744
ppid=2537 pid=2557 comm=vim exe=/bin/vim
uid=tom gid=tom euid=tom
```
PROVENANCE BY AUDIT LOGGING

• Audit logs show
  • Event information
  • File information
  • Process information

< Audit log entry for a file open event >
PROVENANCE BY AUDIT LOGGING

• Audit logs show
  • Event information
  • File information
  • Process information
  • User information

< Audit log entry for a file open event >
PROVENANCE BY AUDIT LOGGING

• Analyze audit logs to generate a causal graph

Audit Log

(1) Proc_A recv <x.x.x.x>
(2) Proc_A fork Malware
PROVENANCE BY AUDIT LOGGING

• Analyze audit logs to generate a causal graph
  • **Backward analysis** - identify the source of an attack

Audit Log

(1) Proc_A recv <x.x.x.x>
(2) Proc_A fork Malware
(3) Malware write File1
(4) Malware write File2
PROVENANCE BY AUDIT LOGGING

• Analyze audit logs to generate a causal graph
  • **Backward analysis** - identify the source of an attack

Audit Log:

(1) Proc_A recv <x.x.x.x>
(2) Proc_A fork Malware
(3) Malware write File1
(4) Malware write File2

Process A
Malware

PROVENANCE BY AUDIT LOGGING

- Analyze audit logs to generate a causal graph
  - **Backward analysis** - identify the source of an attack

Audit Log

1. Proc_A recv <x.x.x.x>
2. Proc_A fork Malware
3. Malware write File1
4. Malware write File2

```
x.x.x.x

Process A

Malware
```

PROVENANCE BY AUDIT LOGGING

- Analyze audit logs to generate a causal graph
  - **Forward analysis** - Understand damage to a system

Audit Log

(1) Proc_A recv <x.x.x.x>
(2) Proc_A fork Malware
(3) Malware write File1
(4) Malware write File2

---

PROVENANCE BY AUDIT LOGGING

- Analyze audit logs to generate a causal graph
- **Forward analysis** - Understand damage to a system

Audit Log:

1. `Proc_A recv <x.x.x.x>`
2. `Proc_A fork Malware`
3. `Malware write File1`
4. `Malware write File2`

---

- : Process
- : File
- : Network Socket
- : Event
PROVENANCE BY AUDIT LOGGING

- Analyze audit logs to generate a causal graph
  - **Forward analysis** - Understand damage to a system

Audit Log:

(1) Proc_A recv <x.x.x.x>
(2) Proc_A fork Malware
(3) Malware write File1
(4) Malware write File2

Malware

File 1

File 2
PROVENANCE BY AUDIT LOGGING

• Backward and forward analysis techniques
  • Sam T. King et al. “Backtracking Intrusion”, In SOSP 2003
  • Beset Paper Award
  • Ashvin Goel et al. “The Taser intrusion recovery system”, In SOSP 2005
PROGRAMMING ASSIGNMENT 1

• Write the process we discusses as an algorithm
  • Input: log file, starting point (file ID or process id)
  • Output: graph

• Language
  • C-C++ suggested for performance reasons
  • Include your Makefile

• Scores: 4 test cases * 25% each
LIMITATIONS OF PREVIOUS WORKS

Dependence explosion
Generated causal graph is too large and contains many bogus information
Almost infeasible to be inspected by human

Sheer size of audit logs
Backtracker[SOSP’03] : 1.2GB/day
Taser[SOSP’05] : 1.9GB/day
Purdue Web Server : 3.7GB/day
Regular Client : 1.2GB/day
DEPENDENCE EXPLOSION - EXAMPLE

SOCIAL ENGINEERING ATTACK BY PHISHING E-MAIL
Social engineering Attack by phishing e-mail

Shiqing Ma, Rutgers University

--

URGENT: Verification of Recent Activities Required
Your Chase Bank Account

Dear Customer:

As part of our ongoing effort to protect your account and our relationship, we monitor your account for possible fraudulent activity. We need to confirm that you or someone authorized to use your account made the following sign in error attempt on your Chase Bank account:

1) Sign in Error Attempt was noticed and registered at 70.43.95.120. Chantilly, Virginia United State on or around 2012-07-11 at 05:01AM.
2) Sign in Error Attempt was noticed and registered at 68.170.136.81. Commack, New York United State on or around 2012-07-11 at 9:30PM.
3) Sign in Error Attempt was noticed and registered at 74.11.103.43 DeIray Beech, Florida United State on or around 2012-07-11 at 8:30PM.
4) Sign in Error Attempt was noticed and registered at 84.46.148.86, Egg Harbor Township, New Jersey, United States on or around 2012-07-11 at 6:29AM.

Please click on the link below to sign in correctly to re-activate your online banking access:

www.chase.com

Your satisfaction is important to us, and we appreciate your prompt attention to this matter. If you already had the opportunity to discuss this matter with us, please disregard this message.

Thank you for being our customer.

Sincerely,

Christopher J. Palumbo
Senior Vice President
Chase Fraud Prevention
ATTACK PROVENANCE - EXAMPLE

- Social Engineering Attack
  - Phishing e-mail with a phishing link
Download Activity

Download account transactions — Download your transaction details into your Quicken®, QuickBooks® or Microsoft® Money software. Use the drop-down list to select your software format, then click “Download Activity.” Note: To download transactions from the last 45 days, leave the beginning and ending date fields blank.

Attention! WaMu credit card customers: Before you attempt to download transactions into your Quicken, QuickBooks or Microsoft Money software, please use these special instructions to help you update your software on or after March 9.

Download information

Select account*  
Choose date range*  
  All transactions available (limited to 45 days)
  Specify a date range

Beginning date
Ending date

Select software format*  

*Required field

Download Activity  Cancel
ATTACK PROVENANCE

• Social Engineering Attack
• Phishing e-mail with a phishing link

Shiqing Ma, Rutgers University
Antivirus Alert

A malicious item has been detected!

Name: Malware@#1xtkvqax9gzy
Location: /home/johnsmith/Virus samples_
More information: Unavailable

How should I answer? [Clean] [Ignore]
Malware
The user visited 11 web sites

**Dependence explosion!!**

(229 IP Addresses)

Firefox

Malware
DEPENDENCE EXPLOSION:
51 PROCESSES, 15 FILES,
251 NETWORK ADDRESSES, 351 EDGES

Shiqing Ma, Rutgers University
DEPENDENCE EXPLOSION – ROOT CAUSE

• Caused by long-running processes
  • Receive many inputs and produces many outputs
DEPENDENCE EXPLOSION – ROOT CAUSE

- Caused by long-running processes
  - Receive many inputs and produces many outputs
  - Any output is potentially related to all preceding inputs
BEEP : BINARY-BASED EXECUTION PARTITION

- Finer-grained subject: Execution "UNIT"
  - Dynamically partition the execution of a process into autonomous execution segments
  - Units are not always independent
    - Detect causality between units

Long running process

Symptom
**BEEP : BINARY-BASED EXECUTION PARTITION**

- Finer-grained subject: Execution "UNIT"
  - Dynamically partition the execution of a process into **autonomous** execution segments

Symptom: Long running process
**BEEP: BINARY-BASED EXECUTION PARTITION**

- Finer-grained subject: Execution “UNIT”
  - Dynamically partition the execution of a process into **autonomous** execution segments
  - Units are not always independent
    - Detect causality between units

Shiqing Ma, Rutgers University
Previous approaches [SOSP’03, SOSP’05] :
51 Processes, 15 Files,
251 Network addresses, 351 Edges

BEEP :
10 Processes, 2 Files,
6 Network addresses, 23 Edges

16.3 times smaller
## Observations from 100 Applications

<table>
<thead>
<tr>
<th>Category</th>
<th>Total Applications</th>
<th>Loop structured Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Servers</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web server</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>Mail server</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>FTP server</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>SSHD server</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>DNS server</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Database server</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Proxy server</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Media server</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Directory server</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Version control server</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Remote desktop server</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>UI Programs</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web browser</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>E-mail client</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>FTP client</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Office</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Text Editor</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Image tool</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Audio player</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Video player</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>P2P program</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Messanger</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>File manager</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Shell program</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Characteristics of long running programs

• Driven by external requests
• Dominated by event processing loops
IDENTIFYING UNIT LOOPS

EVENT PROCESSING LOOP

```c
void main(..) {
    while(..) {
        // Receive request
        // Process request
        // Send result
    }
}
```
IDENTIFYING UNIT LOOPS

- Event processing loop

```c
void main(..) {
    while(..) {
        // Receive request
        // Process request
        // Send result
    }
}
```

Execution Unit:

Iteration of the event processing loop
IDENTIFYING UNIT LOOPS

• Detecting unit loops
  • High level loop

```c
void main(..) {
  while(..)
    // Event process
}
```
IDENTIFYING UNIT LOOPS

- Detecting unit loops
  - High level loop
IDENTIFYING UNIT LOOPS

• Detecting unit loops
  • High level loop
  • Receive inputs and produce outputs

```c
void main(..) {
  while(..) { // Event process
    while(..) { // Memory pool initialize
      ..
    }
  }
  while(..) { // Argument handling
    ..
  }
  ..
  ..
  ..
  ..
  while(..) { // Free allocated memory
    ..
  }
}
```

- accept(…)
- recvfrom(…)
- read(…)
- write(…)
- sendto(…)

Shiqing Ma, Rutgers University
IDENTIFYING UNIT LOOPS

- Analyze a program binary
  - Construct control flow graph and call graph to identify loop heads and exits

Step 1: Static Analysis
IDENTIFYING UNIT LOOPS

- Loop analysis
  - Track loop iterations and system calls to identify event processing loops

Step 1 : Static Analysis

Step 2 : Dynamic Analysis (Training Runs)
• A unit alone may not correspond to a semantically independent sub-execution
INTER-UNIT DEPENDENCES

593    void *listener_thread(..) {
...  631    while(1) {
...  742        req=accept_func(..);
...  768        ap_queue_push(req);
...  798    } // while end
...  810    } // listener_thread end

820    void *worker_thread(..) {
...  842    while(!worker_may_exit) {
...  862        req=ap_queue_pop();
...  894        process_request(req);
...  899    } // while end
...  906    } // worker_thread end

<Apache-2.2.21> - Multi-thread
INTER-UNIT DEPENDENCES

void *worker_thread(...) {
  while(!worker_may_exit) {
    req = ap_queue_pop();
    process_request(req);
  }
}

void *listener_thread(...) {
  while(1) {
    req = accept_func();
    ap_queue_push(req);
  }
}

<Apache-2.2.21> - Multi-thread
void *worker_thread(...)

while(!worker_may_exit)

req = ap_queue_pop();

process_request(req);

// while end

void *listener_thread(...)

while(1)

req = accept_func(...);

req = ap_queue_push(req);

// while end

// listener_thread end

// worker_thread end

Apache-2.2.21 - Multi-thread
INTER-UNIT DEPENDENCES

593  void *listener_thread(..) {
... 631  while(1) {
... 742  req=accept_func(..);
... 768  ap_queue_push(req);
... 798  } // while end
... 810  } // listener_thread end

820  void *worker_thread(..) {
... 842  while(!worker_may_exit) {
... 862  req=ap_queue_pop();
... 894  process_request(req);
... 899  } // while end

906  } // worker_thread end

<Apache-2.2.21> - Multi-thread
void *worker_thread(..) {
  while(!worker_may_exit) {
    req = ap_queue_pop();
    process_request(req);
  }
// while end
}

// worker_thread end

void *listener_thread(..) {
  while(1) {
    req = accept_func(..);
    ap_queue_push(req);
  }
// while end
}

// listener_thread end

<Apache-2.2.21> - Multi-thread
A unit alone may not correspond to a semantically independent sub-execution

- A few units together compose an autonomous sub-execution

```
UNIT_Listener
req = accept();
push_queue(req);
```

```
UNIT_Worker
req = pop_queue();
process(req);
send_result();
```
INTER-UNIT DEPENDENCES

• A unit alone may not correspond to a semantically independent sub-execution
  • A few units together compose an autonomous sub-execution

• Dependences through memory object
  • Ex) queue – enqueue, dequeue

UNIT_Listener
req = accept();
push_queue(req);

UNIT_Worker
req = pop_queue();
process(req);
send_result();
WORKFLOW DEPENDENCE

UNIT_Loader_1
req1 = accept();
push_queue(req1);

UNIT_Loader_2
req2 = accept();
push_queue(req2);

UNIT_Loader_3
req3 = accept();
push_queue(req3);

UNIT_Worker_1
req1 = pop_queue();
process(req1);
update(log_buf);

UNIT_Worker_2
req2 = pop_queue();
process(req2);
update(log_buf);

UNIT_Worker_3
req3 = pop_queue();
process(req3);
update(log_buf);
Workflow Dependences

• Represent high-level program workflow
• Detect semantically relevant units
WORKFLOW DEPENDENCE

UNIT_Listener_1
req1 = accept();
push_queue(req1);

UNIT_Listener_2
req2 = accept();
push_queue(req2);

UNIT_Listener_3
req3 = accept();
push_queue(req3);

UNIT_Worker_1
req1 = pop_queue();
process(req1);
update(log_buf);

UNIT_Worker_2
req2 = pop_queue();
process(req2);
update(log_buf);

UNIT_Worker_3
req3 = pop_queue();
process(req3);
update(log_buf);
LOW LEVEL (BOGUS) DEPENDENCE

Low Level Dependences

• Not part of the workflow
• Caused by low level behavior
Characteristic of workflow objects

• Never shared by units from the same loop
### Workflow Dependence

- **UNIT Listener 1**
  ```
  req1 = accept();
  push_queue(req1);
  ```

- **UNIT Listener 2**
  ```
  req2 = accept();
  push_queue(req2);
  ```

- **UNIT Listener 3**
  ```
  req3 = accept();
  push_queue(req3);
  ```

- **UNIT Worker 1**
  ```
  req1 = pop_queue();
  process(req1);
  update(log_buf);
  ```

- **UNIT Worker 2**
  ```
  req2 = pop_queue();
  process(req2);
  update(log_buf);
  ```

- **UNIT Worker 3**
  ```
  req3 = pop_queue();
  process(req3);
  update(log_buf);
  ```
INTER-UNIT DEPENDENCES

Step 1 : Static Analysis

Step 2 : Dynamic Analysis (Training Runs)
INSTRUMENTATION

Step 1 : Static Analysis

Step 2 : Dynamic Analysis
  (Training Runs)

Step 3 : Instrumentation
Step 1: Static Analysis
Step 2: Dynamic Analysis (Training Runs)
Step 3: Instrumentation
Step 4: Audit Logging
Step 5: Log Analysis
### EVALUATION PROGRAMS

<table>
<thead>
<tr>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sshd-5.9</td>
</tr>
<tr>
<td>Sendmail-8.12</td>
</tr>
<tr>
<td>Proftpd-1.3.4</td>
</tr>
<tr>
<td>Apache-2.2.21</td>
</tr>
<tr>
<td>Cherokee-1.2.1</td>
</tr>
<tr>
<td>Wget-1.13</td>
</tr>
<tr>
<td>W3m-0.5.2</td>
</tr>
<tr>
<td>Pine-4.64</td>
</tr>
<tr>
<td>MidnightCommand-4.6.1</td>
</tr>
<tr>
<td>Vim-7.3</td>
</tr>
<tr>
<td>Bash-4.2</td>
</tr>
<tr>
<td>Firefox-11</td>
</tr>
<tr>
<td>Yafc-1.1.1</td>
</tr>
<tr>
<td>Transmission-2.6</td>
</tr>
</tbody>
</table>
LOGGING OVERHEAD (TIME)

Shiqing Ma, Rutgers University
LOGGING OVERHEAD (SPACE)

- Additional space consumption for storing unit / unit dependence events
BEEP LIMITATIONS

• Low level events
  • Mouse click

• Training for inter-unit dependences
  • Very heavy process

• Excessive Units
  • Mouse movement events

• Meaningful execution units
  • Tab

• Multiple Perspectives
  • E.g. tabs v.s. pages
  • No training

• Less units
  • Drop units if possible
MPI
(USENIX SEC’17 DISTINGUISHED PAPER AWARD)

• Inspired by process isolation mechanisms in operating systems

• IDEA: Execution partitioning based on user-defined Tasks
  • Task: represented by data structures

• Different tasks indicate different perspectives
  • Firefox: Tabs, Pages
ANNOTATION

• Part of Clang/Gcc language extensions

• Widely used: Firefox
  • 926 different types of annotations
  • NS_STACK_CLASS: 406 annotated classes

• Adding customized attributes to Variables/Functions etc.
  • `__attribute__((annotate("annotation strings")))`
BASIC ANNOTATIONS

• @indicator -> current in OS
  • *Indicating the switches between task instances*
  • This defines the place (where) to instrument

• @identifier -> pid in OS
  • *Identifying different task instances*
  • This defines the the value (what) to expose

• @channel -> IPCs in OS
  • *Communication channels used between instances*
  • This helps find relationships of all instances
struct buf_T {
    @identifier
    int buf_id;
    char* name;
    ...
};

@indicator
buf_T* curbuf;

@channel
struct yankreg *y_current;
do_ecmd(. . .){
    . . .
    //a new buffer
    curbuf = buf;
    curbuf->name = . . .
}

if (curbuf != oldbuf) {
    oldbuf = curbuf;
    expose(curbuf->buf_id);
}

expose(id){
    kill(-100, id);//for linux audit system
}
Tasks can be represented by data structures

- `buf_T` in Vim

Annotation miner is used to help developers find such data structures
**ANoATION MINER**

Type: # of instances

```
struct buf_T
{
  /* position */
  pos_T pos;
}
```

**Data Structure:**

No.(A) ≠ 1/2 No.(B)

buf_T: 1, 2
• Worker Threads
  • Working on the same type of tasks
  • Tasks in this thread are *sub-tasks*
  • One thread is serving for multiple *top-level tasks*

• *@delegator*
  • Add one field for top-leve task id
  • Inherit task identifiers from the top-level task
@delegator
class nsConnEvent {}

PostEvent(...) {
    event = new nsConnEvent(...);
    rv = target->Dispatch(event);
}

event->global_id = current_id;
## EVALUATION: ANNOTATIONS

<table>
<thead>
<tr>
<th>Application</th>
<th>LOC</th>
<th>Annotation</th>
<th>Inst</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ID</td>
<td>IND</td>
</tr>
<tr>
<td>Vim</td>
<td>313,283</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Yafe</td>
<td>22,823</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Firefox</td>
<td>8,073,181</td>
<td>3</td>
<td>32</td>
</tr>
<tr>
<td>TuxPaint</td>
<td>41,682</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Pine</td>
<td>353,665</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Apache</td>
<td>168,801</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>MC</td>
<td>135,668</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>ProFTPD</td>
<td>307,050</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Transmission</td>
<td>111,903</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>W3M</td>
<td>67,291</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
## EVALUATION: SPACE OVERHEAD

<table>
<thead>
<tr>
<th>Application</th>
<th>Perspective</th>
<th>BEEP-Audit</th>
<th>BEEP-HiFi</th>
<th>MPI-Audit</th>
<th>MPI-HiFi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>Connection</td>
<td>15.38%</td>
<td>12.87%</td>
<td>5.37%</td>
<td>3.75</td>
</tr>
<tr>
<td>Bash</td>
<td>Command</td>
<td>0.45%</td>
<td>0.34%</td>
<td>0.41%</td>
<td>0.34%</td>
</tr>
<tr>
<td>Firefox</td>
<td>Tab</td>
<td>42.16%</td>
<td>38.23%</td>
<td>18.20%</td>
<td>13.24%</td>
</tr>
<tr>
<td>Pine</td>
<td>Command</td>
<td>8.11%</td>
<td>6.09%</td>
<td>7.28%</td>
<td>4.09%</td>
</tr>
<tr>
<td>Vim</td>
<td>File</td>
<td>2.23%</td>
<td>2.32%</td>
<td>0.13%</td>
<td>0.13%</td>
</tr>
</tbody>
</table>
EVALUATION: SPACE OVERHEAD

- Window
- Tab
- Element
- Conn
- Request
EVALUATION: RUNTIME OVERHEAD CAUSED BY MPI
CASE STUDY

- Firefox visits a few websites
- During browsing, download a few torrent files and normal files
- Use Transmission to download the torrents
TRADITIONAL SOLUTION
MPI
(FIREFOX PARTITIONED BY TAB AND TRANSMISSION PARTITIONED BY INPUT FILE)
MPI
(FIREFOX PARTITIONED BY WEBSITE AND TRANSMISSION PARTITIONED BY INPUT FILE)
LIMITATIONS

Source code
LLVM-based solution

Annotation
Miner is helpful but not fully automated
Miner finds the data structure, not the @indicator/@identifier variables
OTHER READINGS

NoDoze
NDSS 19

PrioTracker
NDSS 18