Dynamic Program Analyses and Their Security Applications

Xiangyu Zhang

PURDUE
UNIVERSITY

PURDUE UNIVERSITY

What is Dynamic Program Analysis

- Dynamic analysis analyzes program execution
  - Examples
    - Identify the sequence of instructions that get executed
    - Identify the set of values that a variable holds during its lifetime
    - Identify the set of memory addresses/files that are accessed
  - Static analysis analyzes programs without executing them
    - Identify loops in the program
    - Generate control flow graph
    - Identify code clones
    - Static tainting

Applications of Dynamic Analysis

- Software engineering
  - Debugging: what is the root cause that leads to an observed program failure?
  - Software optimization
  - Software validation

- Security
  - Anomaly detection
  - Forensic analysis
  - Expose hidden malicious logic

Outline

Temporal Dynamic Analysis:

analyze program execution history

1. Attack investigation

2. Application vetting

Spatial Dynamic Analysis:

analyze a snapshot of program execution state

3. Memory forensics

Dynamic Analyses

Outline

Temporal Dynamic Analysis:

analyze program execution history

1. Attack investigation

2. Application vetting

Spatial Dynamic Analysis:

analyze a snapshot of program execution state

3. Memory forensics

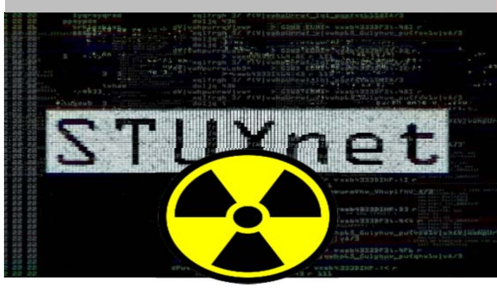
Dynamic Analyses

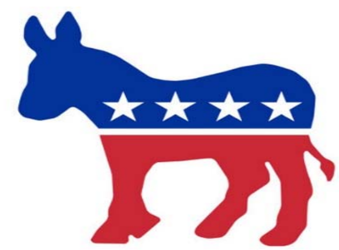
Cyberattacks are becoming more sophisticated

- **Advanced Persistent Threat (APT)**

  Targeted: Targets specific organizations to exfiltrate information or disrupt the systems.

Infrastructure (Nuclear plants)	Business (Target® Data Breach)	Government (OPM: Office of Personnel Management)	Politics (DNC email hack)





PURDUE UNIVERSITY

Multiple stages of APTs

1. Reconnaissance*:* Learn the target organization

2. Infiltration*:* Enter into the victim via social-engineering (e.g., phishing emails) or vulnerabilities

3. Discovery and capture*:* Stay *low* and operate *slowly* to avoid detection while discovering critical machines and/or information.

4. Exfiltration/Disruption*:* Send the captured secret information to attackers or destroy the systems

Combatting APTs is challenging

3. Discovery and capture*:* Stay *low* and operate *slowly* to avoid detection while discovering critical machines and/or information.

(Whitelisted) benign built-in software

APT actors often leverage *benign built-in software (e.g., web-browsers and email clients* that are *already whitelisted)* to avoid detection

Low and slow (Stealthy)

Incidents are often detected after a *few months*.

Audit Logging

- Audit logging records system level events during system execution
  - For attack investigation: forensic analysis
    - Identify the source of an attack
    - Understand the damage to a victim system

Audit Logging

- Analyze audit logs to generate a causal graph

(1) Proc_A recv <x.x.x.x>
(2) Proc_A fork **Malware**

Audit Log

PURDUE UNIVERSITY

Audit Logging

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

(1) Proc_A recv <x.x.x.x>
(2) Proc_A fork Malware

Audit Log

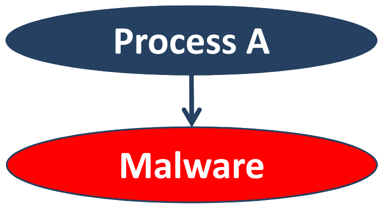
: Process : File : Network Socket : Event

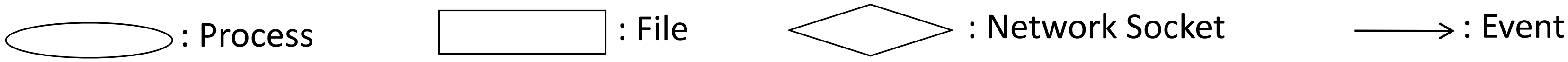
Audit Logging

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

(1) Proc_A recv <x.x.x.x>
(2) Proc_A fork Malware

Audit Log



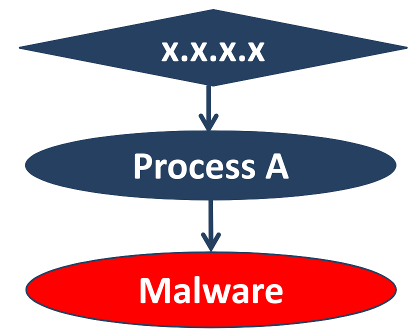


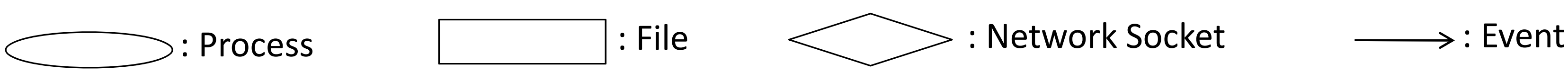
Audit Logging

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

(1)	**Proc_A**	**recv**	**<x.x.x.x>**
(2)	Proc_A	fork	Malware

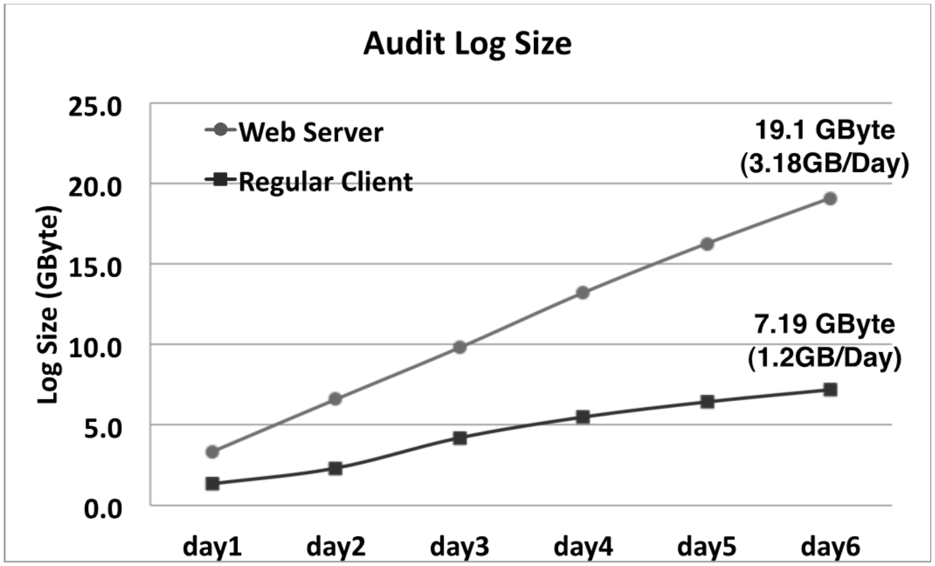
Audit Log





Limitations of Traditional Audit Logging

- Dependence explosion problem
  - Long running programs generate a lot of system level dependences during their lifetime
- High overhead
  - Linux Audit Framework: ***~40%*** run time slow down
  - Space overhead



Dependence Explosion - Example

- Social engineering attack by phishing e-mail

- Social engi

From: Chase Online [mailto:smrfs@chaseonline.com]
Sent: Wednesday, July 11, 2012 8:27 PM
Subject: Verification of Recent Activities Required

CHASE

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 70.43.95.130. 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 8:30PM.

3) Sign in Error Attempt was noticed and registered at 74.11.185.43 Delray Beach, Florida United State on or around 2012-07-11 at 8:20PM.

4) Sign in Error Attempt was noticed and registered at 68.46.148.86, Egg Harbor Township, New Jersey, United States on or around 2012-07-11 at 6:39AM.

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

PURDUE
UNIVERSITY

CHASE

Chase.com | Contact Us | Privacy Notice | En Español | LOG OFF

My Accounts | Payments & Transfers | Products & Services | Customer Center

CHASE ONLINE[SM] Monday, January 13, 2014

My Accounts > Download Activity

Download Activity

Help with this page

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**.

***Required field**

Download Information

Select account* Select Account

Choose date range* All transactions available (Limited to 45 days)

Specify a date range

Beginning date (mm/dd/yyyy)

Ending date (mm/dd/yyyy)

Select software format* -- Select Download Type --

***Required field**

Download Activity | Cancel

Security | Terms of Use | Legal Agreements and Disclosures

CHASE

Chase.com | Contact Us | Privacy Notice | En Español | LOG OFF

My Accounts | Payments & Transfers | Products & Services | Customer Center

CHASE ONLINE℠ Monday, January 13, 2014

My Accounts > Download Activity

Download Activity

Help with this page

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**.

***Required field**

Download Information

Select account* Select Account

Choose date range* All transactions available (Limited to 45 days)

Specify a date range

Beginning date (mm/dd/yyyy)

Ending date (mm/dd/yyyy)

Select software format* -- Select Download Type --

***Required field**

Download Activity **Cancel**

Security | Terms of Use | Legal Agreements and Disclosures

CHASE

Chase.com | Contact Us | Privacy Notice | En Español | LOG OFF

My Accounts | Payments & Transfers | Products & Services | Customer Center

CHASE ONLINE[SM] Monday, January 13, 2014

My Accounts > Download Activity

Download Activity

Download account trans
into your Quicken®, QuickBooks® or I
select your software format, then click
from the last 45 days, leave the begin

Attention! **WaMu credit card custom**
your Quicken, QuickBooks or Microso
to help you update your software on o

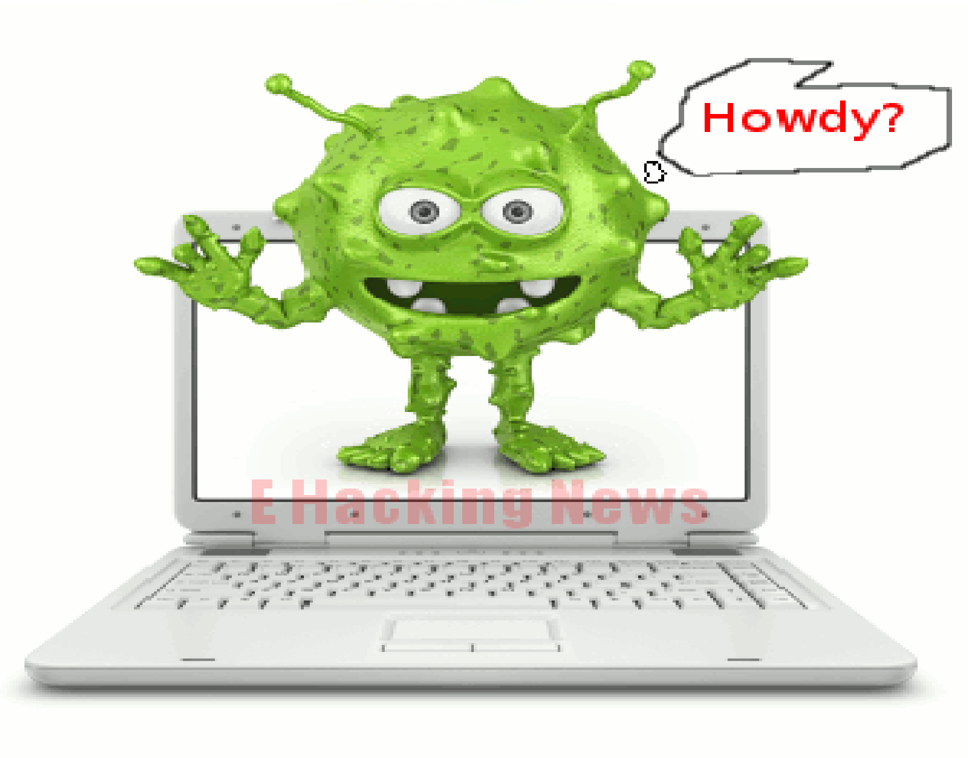
*Required field

Download Information

C

Selec

*Required field



Security | Terms of Use | Legal Agreements and Disclosures


Antivirus Alert
A malicious item has been detected!
Name:
Malware@#1xtkkvqax9gzy
Location:
/home/johnsmith/Virus samples_...
More information:
Unavailable
How should I answer?
Clean
Ignore

[*Backward Analysis*]

Malware

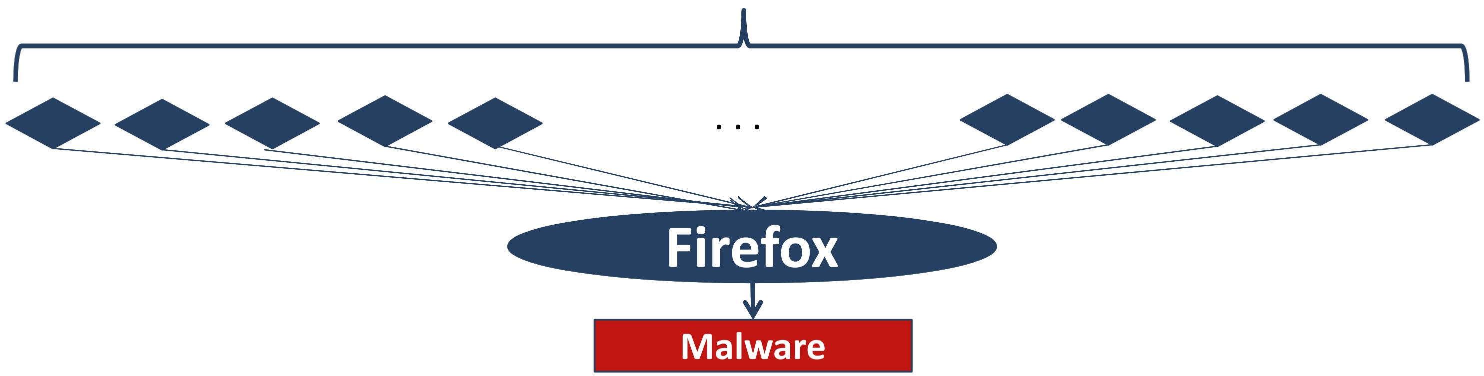
[*Backward Analysis*]

[*Backward Analysis*]

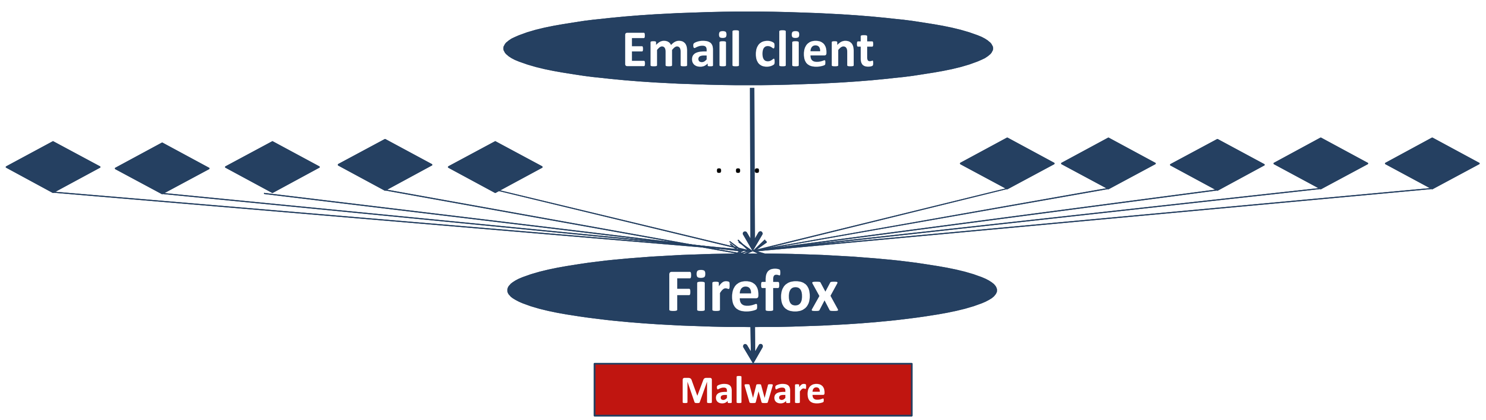
The user visited 11 web sites

Dependence explosion!!

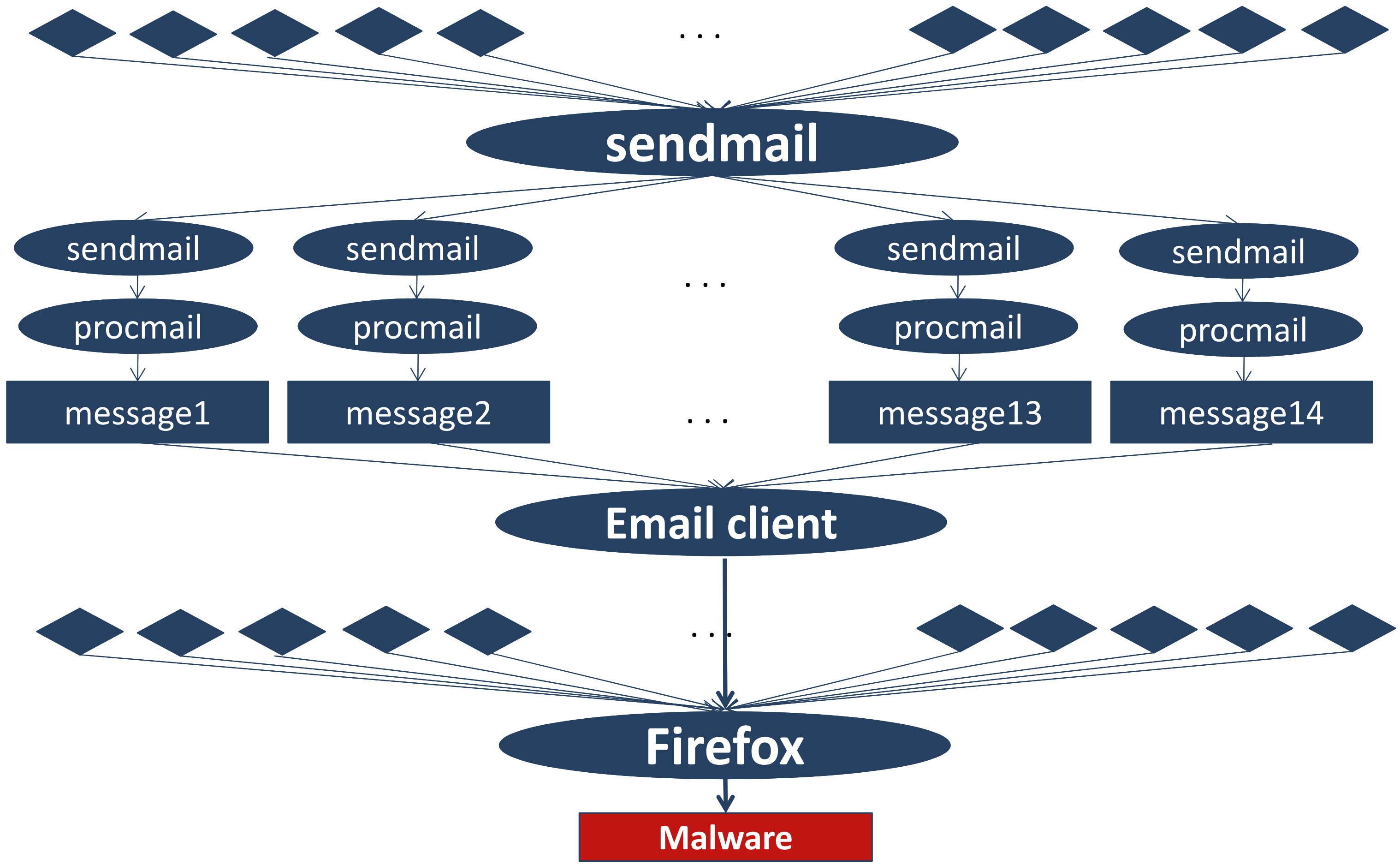
(229 IP Addresses)



[*Backward Analysis*]



[*Backward Analysis*]



[*Backward Analysis*]

. . .

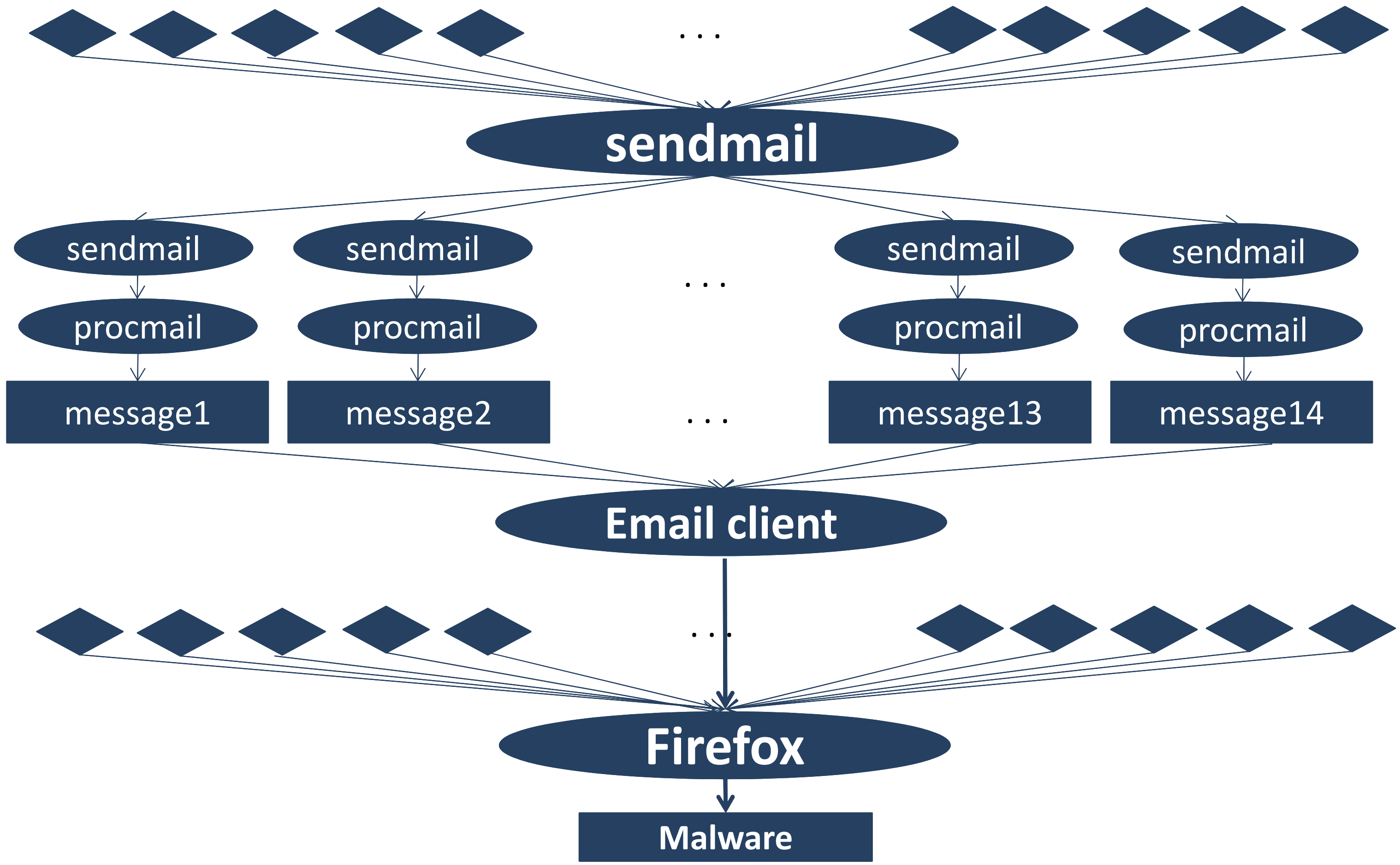
Dependence Explosion :

51 Processes, **15** Files,

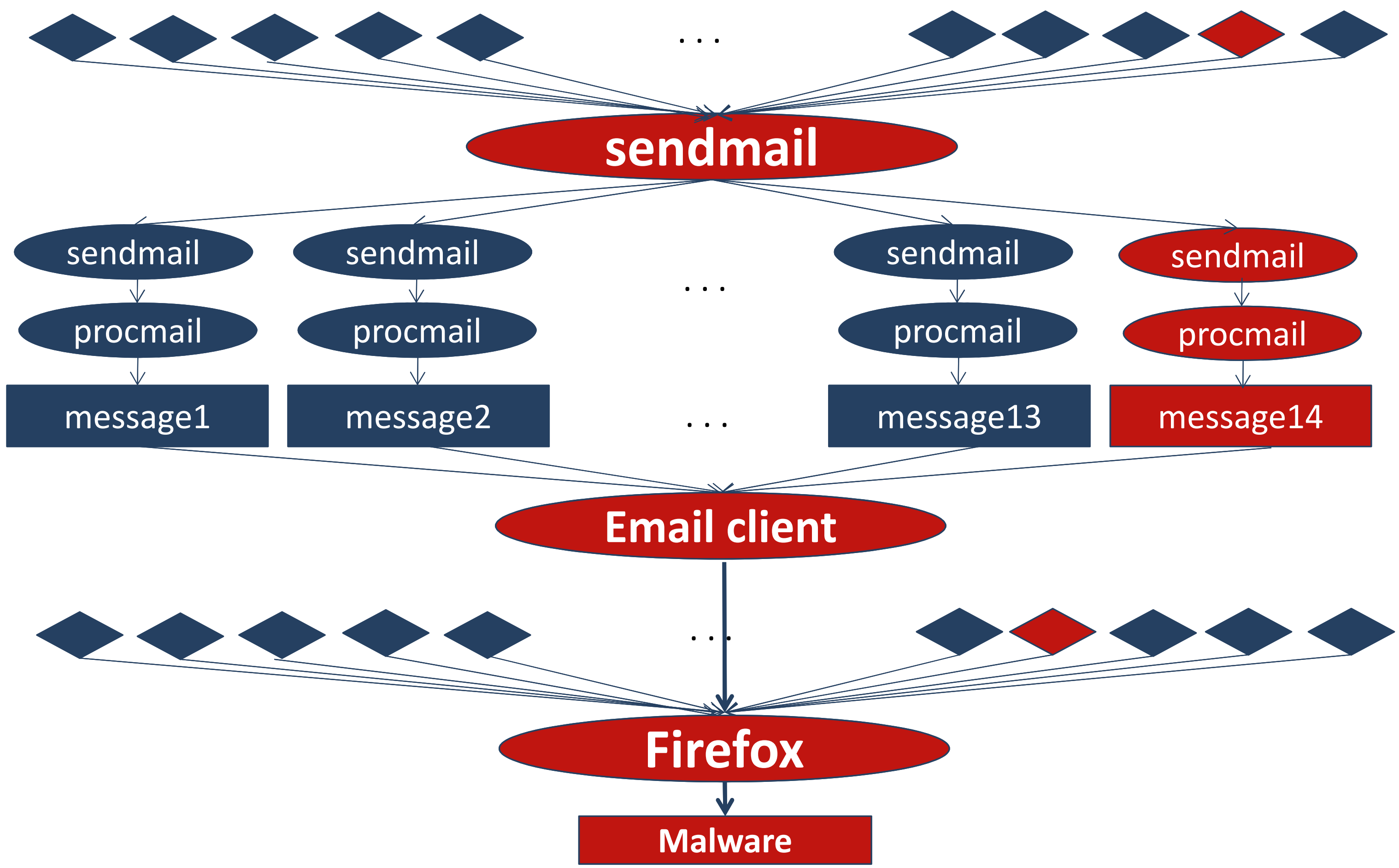
251 Network addresses, **351** Edges

Malware

[*Backward Analysis*]

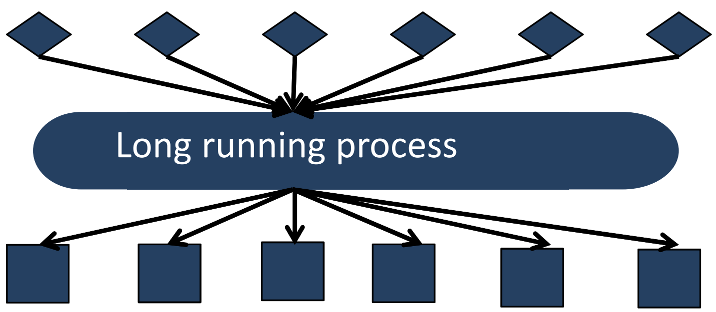


[*Backward Analysis*]



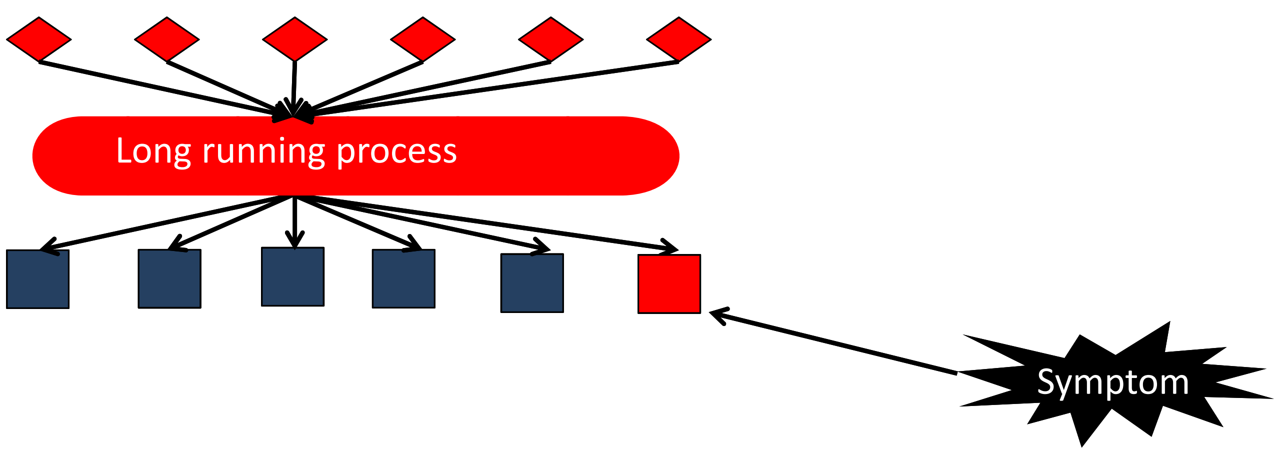
Dependence Explosion – Root Cause

- Caused by **long-running processes**
  - Receive many inputs and produces many outputs



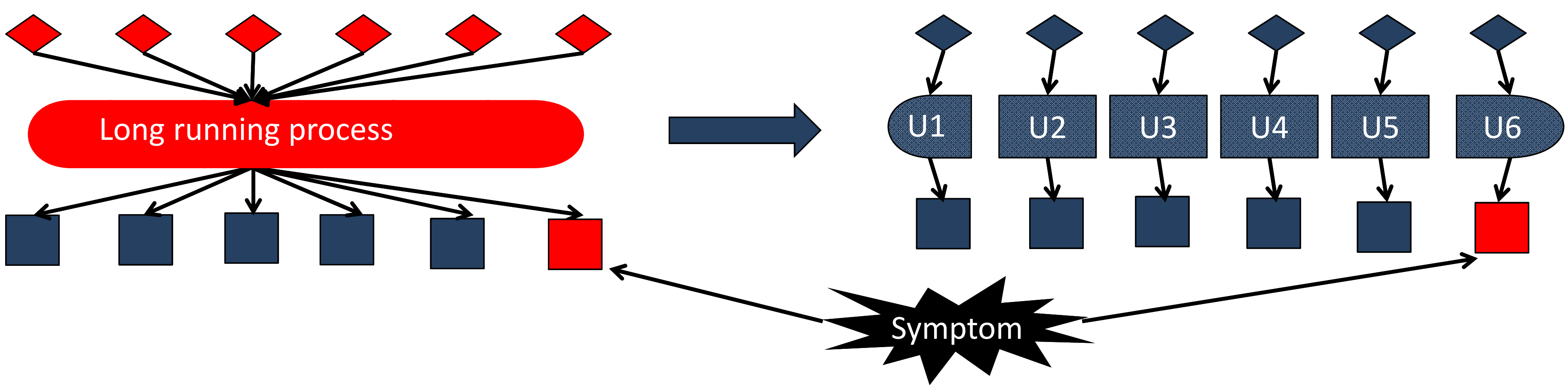
Dependence Explosion – Root Cause

- Caused by long-running processes
  - Receive many inputs and produces many outputs
  - Any output is potentially related to all preceding inputs



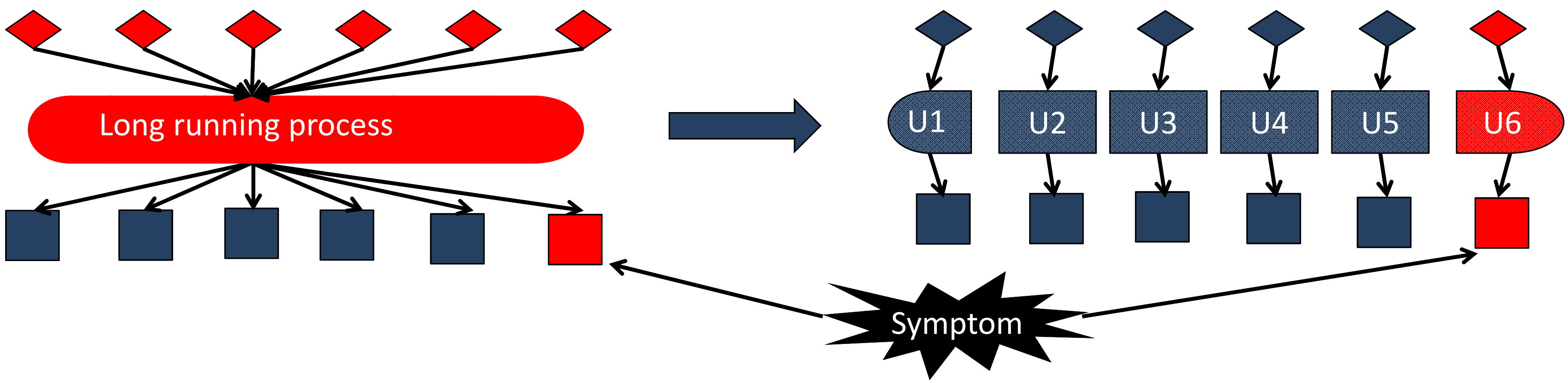
BEEP : Binary-based ExEcution Partition

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



BEEP : Binary-based ExEcution Partition

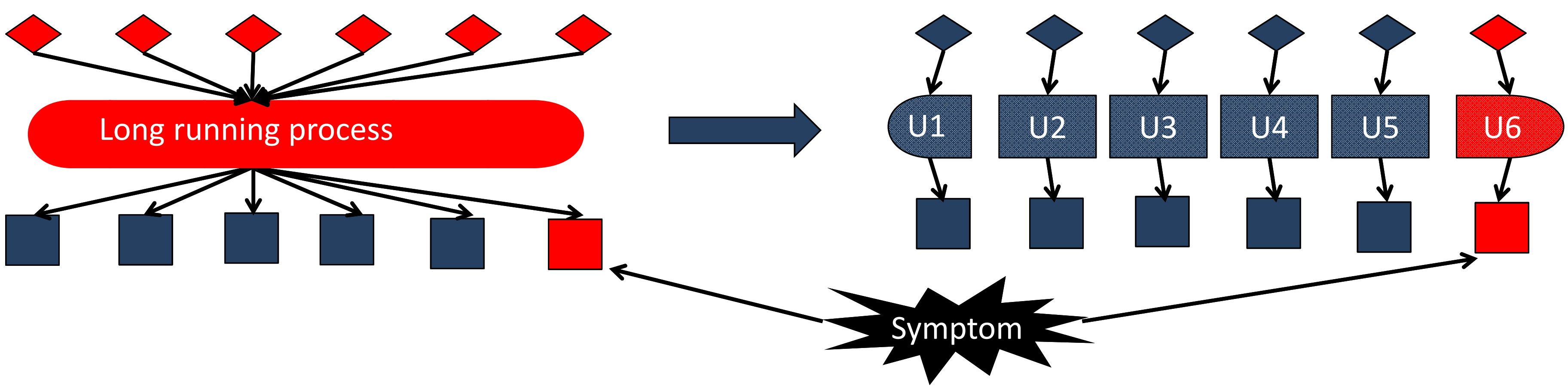
- Finer-grained subject : Execution "UNIT"
  - Dynamically partition the execution of a process into **autonomous** execution segments



PURDUE UNIVERSITY

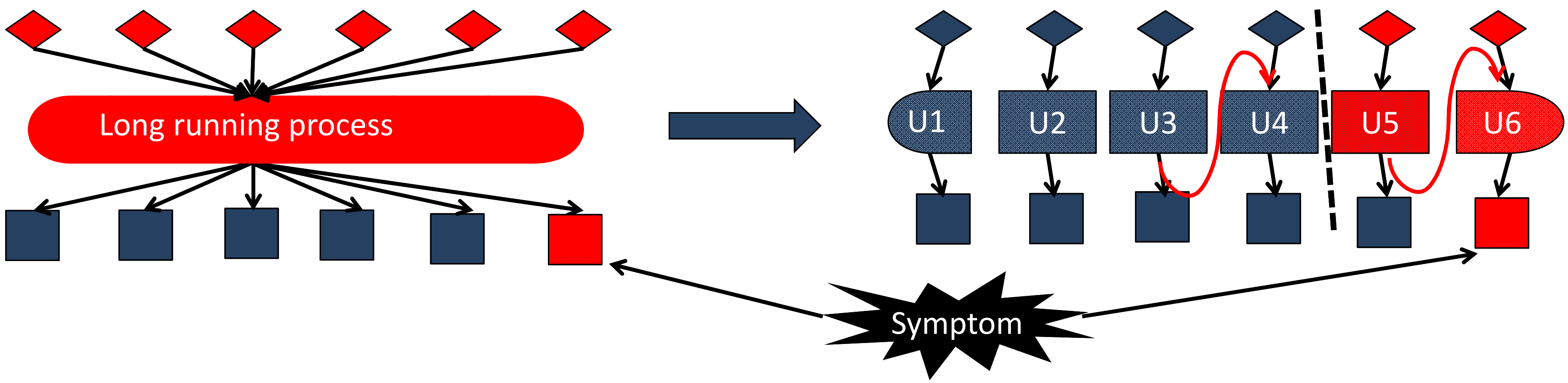
BEEP : Binary-based ExEcution Partition

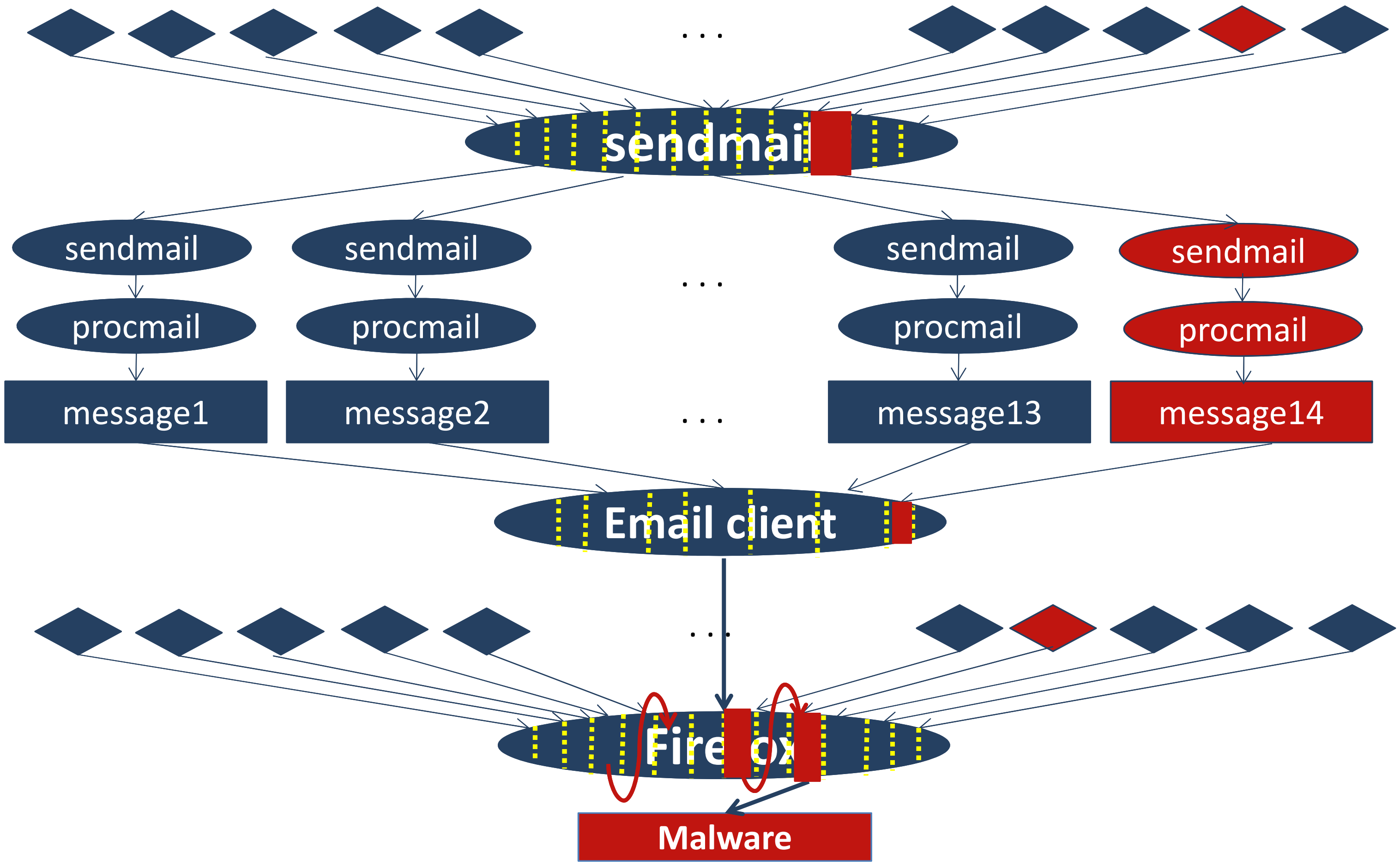
- Finer-grained subject : Execution "UNIT"
  - Dynamically partition the execution of a process into **autonomous** execution segments
  - Units are not always independent
    - Detect causality between units



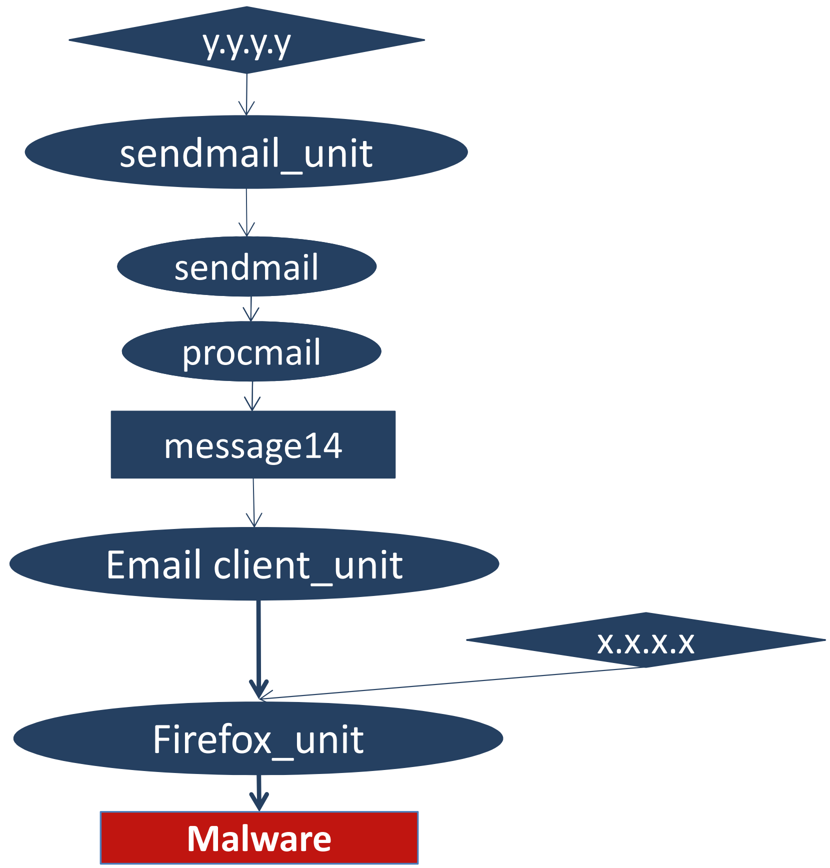
BEEP : Binary-based ExEcution Partition

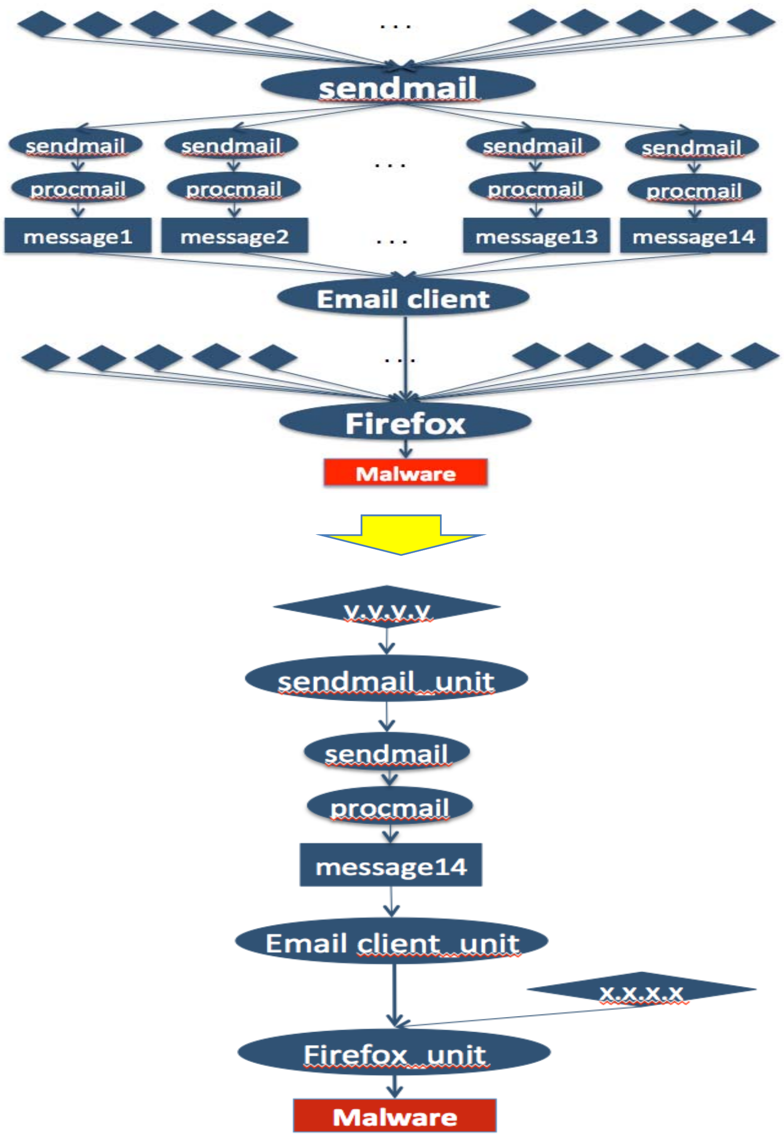
- Finer-grained subject : Execution "UNIT"
  - Dynamically partition the execution of a process into **autonomous** execution segments
  - Units are not always independent
    - Detect causality between units





. . .
sendmail
sendmail
sendmail
. . .
sendmail
sendmail
procmail
procmail
procmail
procmail
message1
message2
. . .
message13
message14
Email client
. . .
Firefox
Malware


y.y.y.y
sendmail_unit
sendmail
procmail
message14
Email client_unit
x.x.x.x
Firefox_unit
Malware



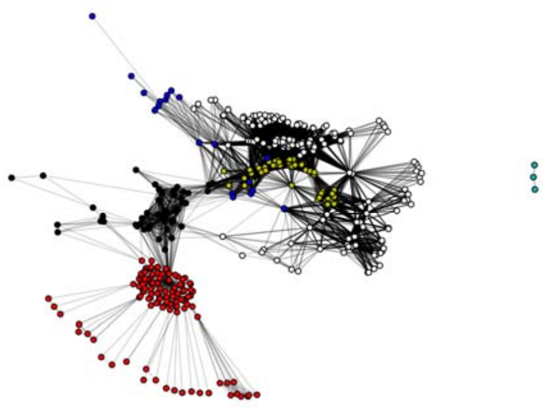
Previous approaches [SOSP'03, SOSP'05] **:**
51 Processes**, 15** Files**,**
251 Network addresses**, 351** Edges

16.3 times smaller

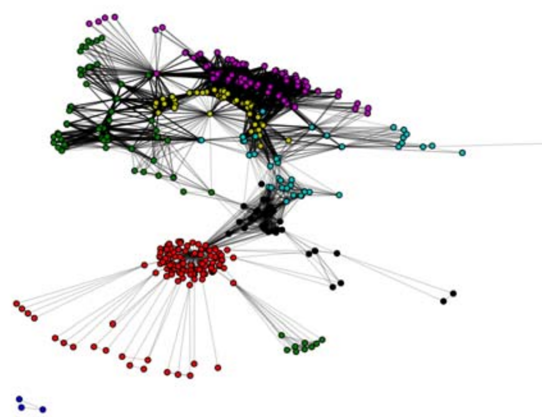
BEEP :
10 Processes**, 2** Files**,**
6 Network addresses**, 23** Edges

Evaluation on Real APT Attacks

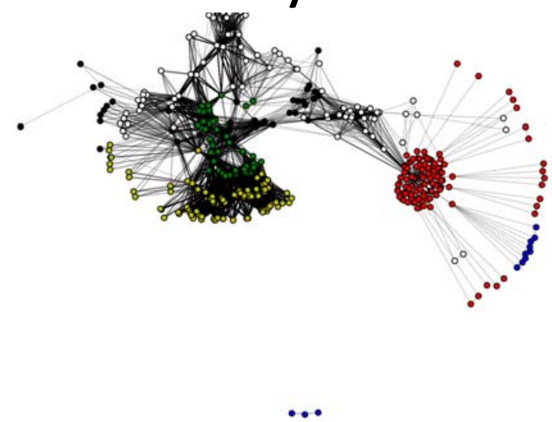
- Reproduce about 15 real attacks
- Our graphs are usually one-two orders of magnitude smaller, and cover whole attack paths with over 90% accuracy.



“Russian Campaign”



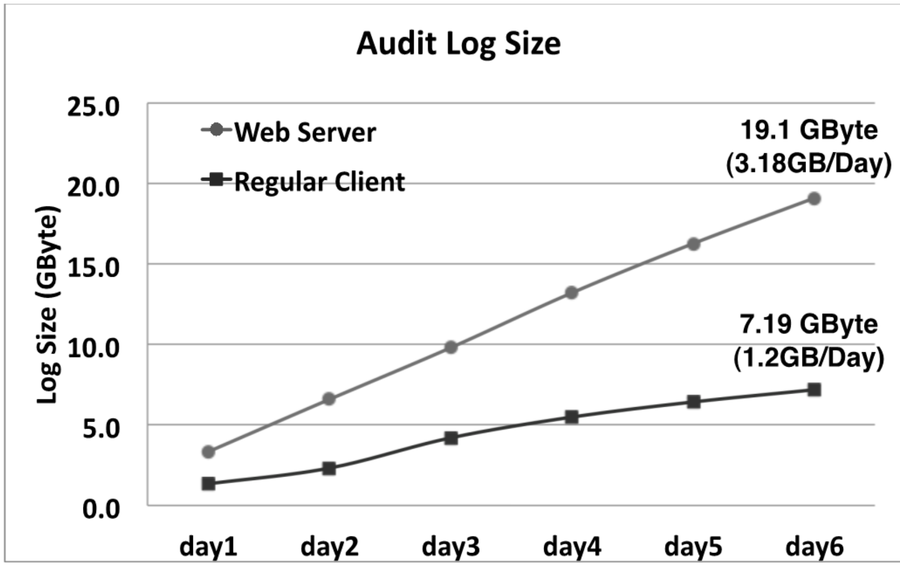
“Black Vine”



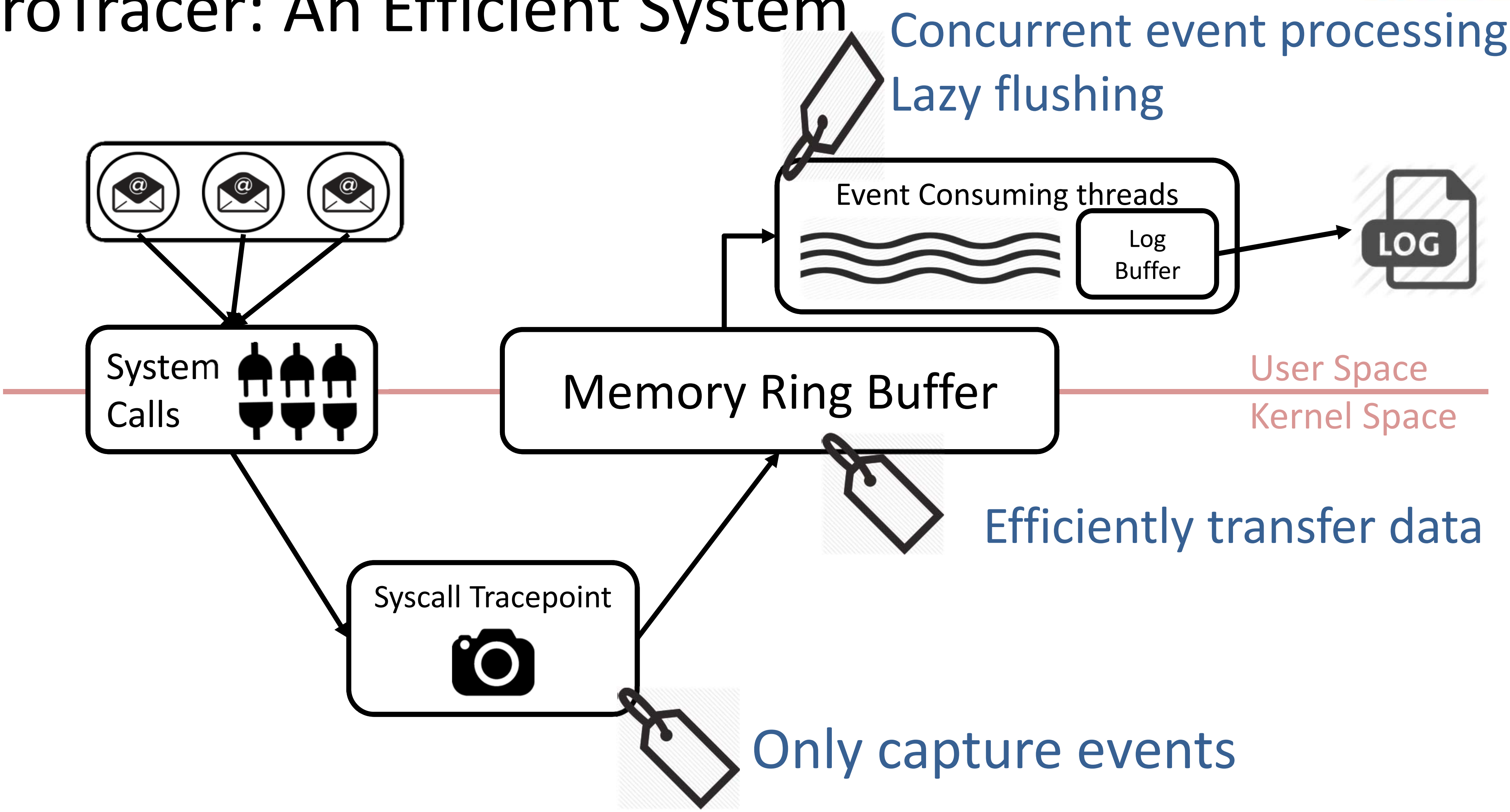
“Op-DeputyDog”

Limitations of Traditional Audit Logging

- Dependence explosion problem (NDSS'13)
  - Long running programs generate a lot of system level dependences during their lifetime
- High overhead
  - Linux Audit Framework: ***~40%*** run time slow down
  - Space overhead
  - Our solution
    - ProTracer (NDSS'16 Distinguished Paper)



ProTracer: An Efficient System



Example: Avoid ***Redundant*** Events

```
# vim opening a large file
    ...
    while((size = read(fd, buf)) > 0):
          add_node(root, buf)
    ...
    exit();
```

Logging

...
PID = 1483, TYPE = SYSCALL: Syscall = read
PID = 1483, TYPE = SYSCALL: Syscall = read
PID = 1483, TYPE = SYSCALL: Syscall = read
PID = 1483, TYPE = SYSCALL: Syscall = read
PID = 1483, TYPE = SYSCALL: Syscall = read
PID = 1483, TYPE = SYSCALL: Syscall = read
...
PID = 1483, TYPE = SYSCALL: Syscall = exit

ProTracer

...
T[PID=1483] = *{ vim }*
T[PID=1483] = T[PID=1483] ***V*** *{ fd }* = *{ vim, fd }*
T[PID=1483] = T[PID=1483] ***V*** *{ fd }* = *{ vim, fd }*
T[PID=1483] = T[PID=1483] ***V*** *{ fd }* = *{ vim, fd }*
T[PID=1483] = T[PID=1483] ***V*** *{ fd }* = *{ vim, fd }*
T[PID=1483] = T[PID=1483] ***V*** *{ fd }* = *{ vim, fd }*
T[PID=1483] = T[PID=1483] ***V*** *{ fd }* = *{ vim, fd }*
...
LogBuffer: T[PID=1483] = *{ vim, fd }*

PURDUE UNIVERSITY

Example: Lazy Flushing

```
1.  # temporary files
2.    f = open(fname, create | write)
3.  # File manipulation on the file
4.    while (not done)
5.      edit(f)
6.  # delete temporary file
7.    delete(f)
```

Logging

...
TYPE = SYSCALL: **Syscall** = open, FD = 8
TYPE = SYSCALL: **Syscall** = write, FD = 8
......
TYPE = SYSCALL: **Syscall** = write, FD = 8
TYPE = SYSCALL: **Syscall** = unlink , FD = 8
...

ProTracer

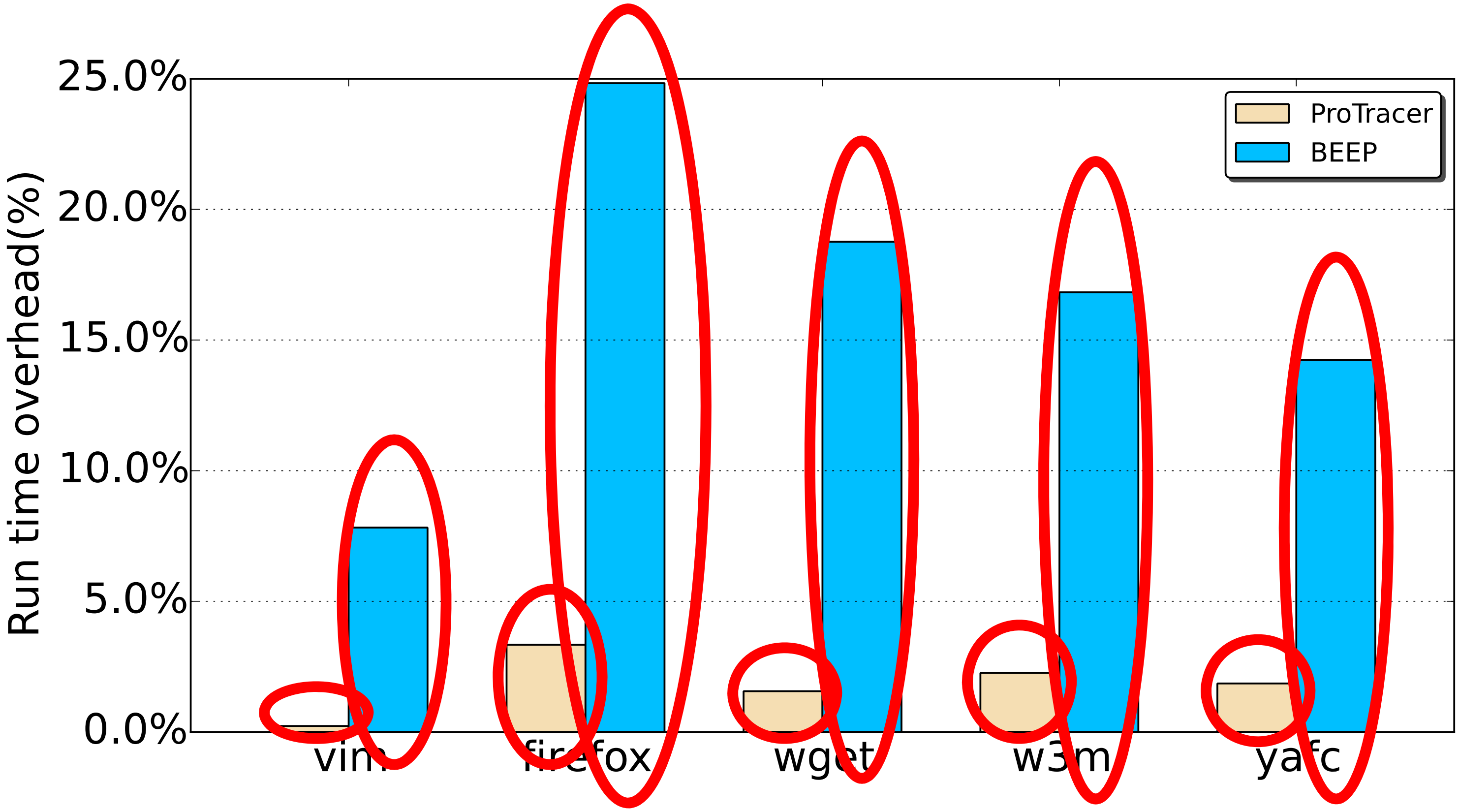
...
T[**FD=8**] = *{ }*
T[**FD=8**] = *{ vim }*
LogBuffer: T[**FD=8**] = *{ vim }*
T[**FD=8**] = T[FD=8] V *{ vim }* = *{ vim }*
LogBuffer: T[**FD=8**] = *{ vim }*
DEL: T[**FD=8**]
...

LogBuffer

~~T[FD=8] = { vim }~~
~~T[FD=8] = { vim }~~

Evaluation: Storage Efficiency (3 months, client)

Evaluation: Run time Efficiency (Client Programs)



1.9% v.s. 16.5%

Whole system: 7% v.s. 40%

Purdue University

Outline

Temporal Dynamic Analysis:

analyze program execution history

1. Execution partitioning overcomes the dependence explosion problem

2. ProTracer reduces space consumption by from over 1GB per day to less than 20MB per day, and runtime overhead from 40% to 7%

…on state

1. Attack investigation

2. Application vetting

Dynamic Analyses

Outline

Temporal Dynamic Analysis:

analyze program execution history

1. Attack investigation

2. Application vetting

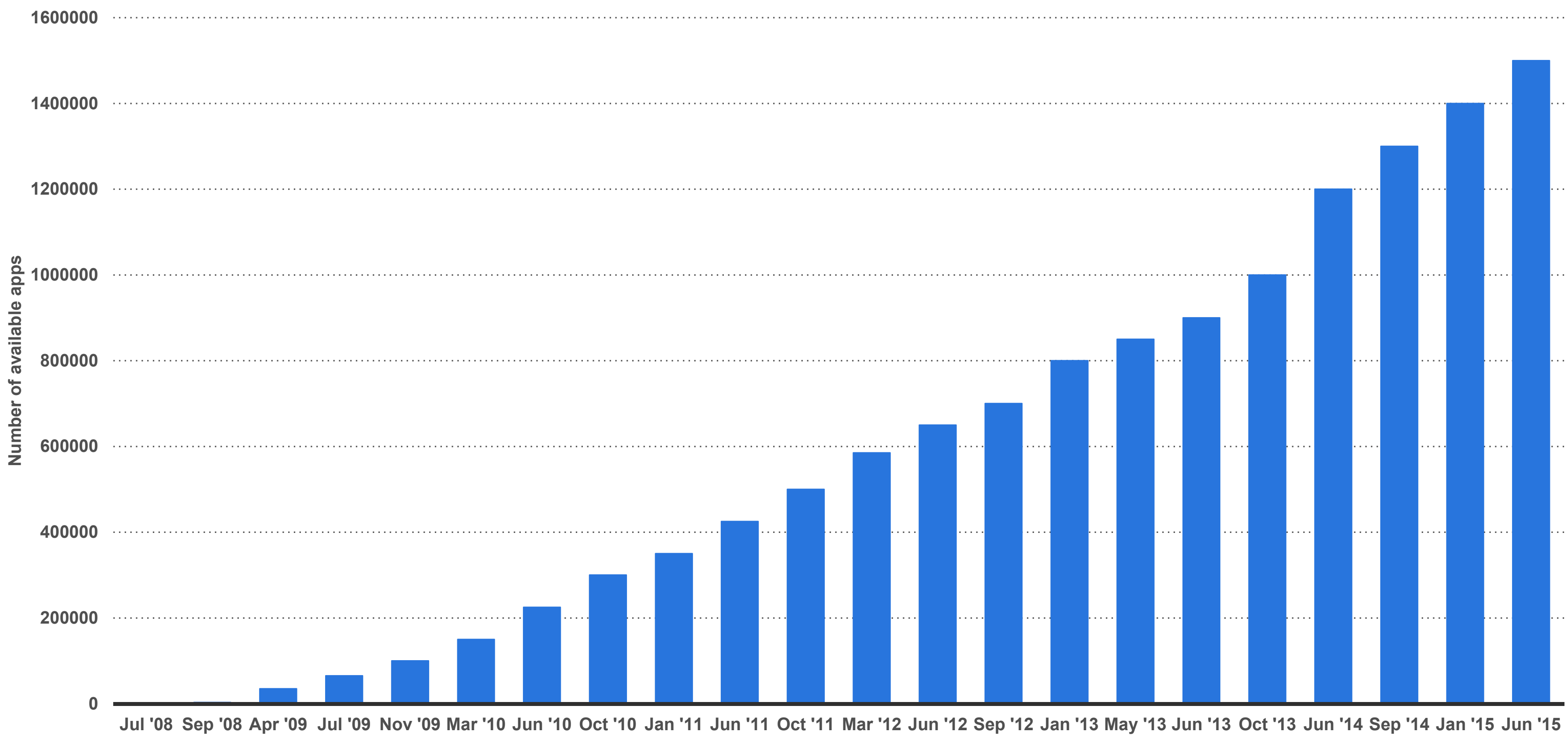
Spatial Dynamic Analysis:

analyze a snapshot of program execution state

3. Memory forensics

Dynamic Analyses

Motivation: Total number of iOS applications (2008-2015)



Motivation: …but Apple is watching out for you

Apple has a pre-release app vetting procedure call **App Review**

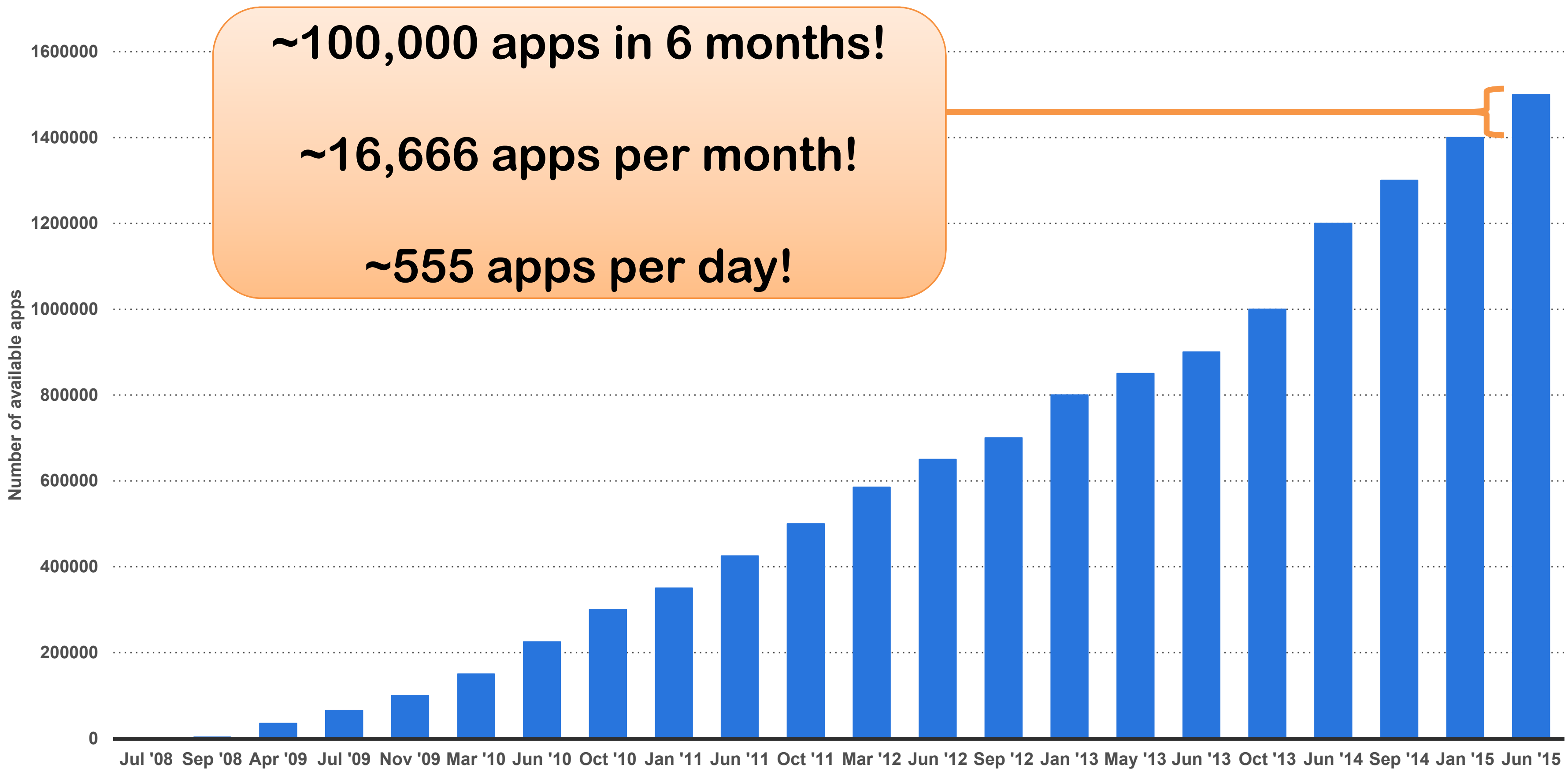
"We review all apps […] to ensure they are reliable, perform as expected, and are free of offensive material."

- Apple App Review Team

https://developer.apple.com/app-store/review/

Apple is secretive about the specific evaluation … Security?

Motivation: Total number of iOS applications (2008-2015)



Observations…

App Review vetting procedure is:

1) Very fast and lightweight
2) “Almost” fully automated
3) Static Program Analysis

Observations…

App Review vetting procedure is:

1) Very fast and lightweight
2) “Almost” fully automated
3) Static Program Analysis

Easily Subverted!

Control Flow & Data Obfuscation

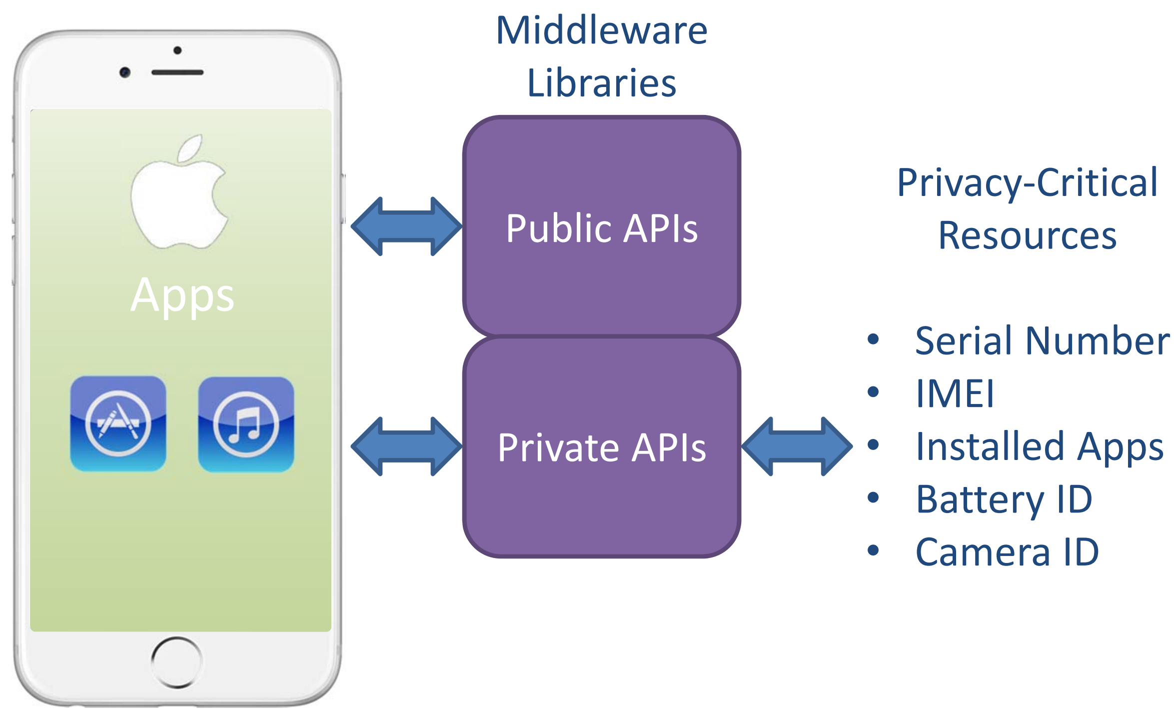
Wang et.al [USENIX Security’13]
Han et.al [ACNS’13]
Zheng et.al [ASIACCS’15]

InstaStock, FindAndCall, PawnStorm.A, LBTM, FakeTor, XCodeGhost, etc....

PURDUE UNIVERSITY

Goal: Private APIs

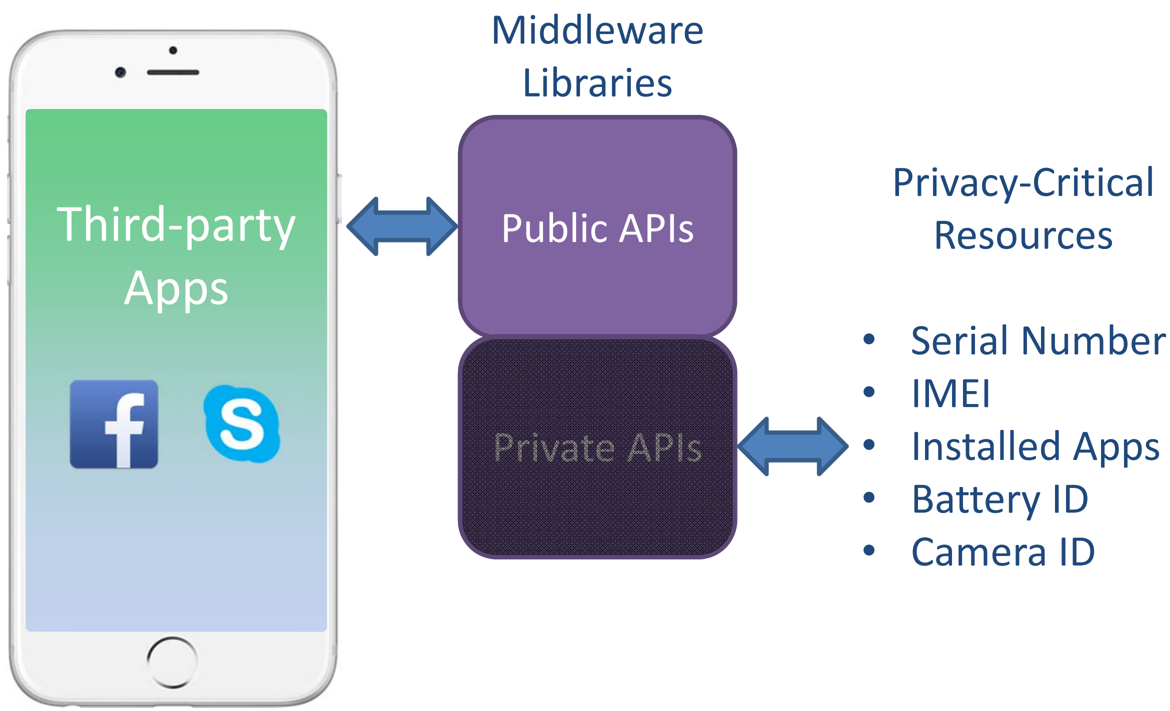
Middleware APIs reserved only for Apple's internal use



Goal: Private APIs

Middleware APIs reserved only for Apple's internal use

Private APIs are not exported for third-party code to use

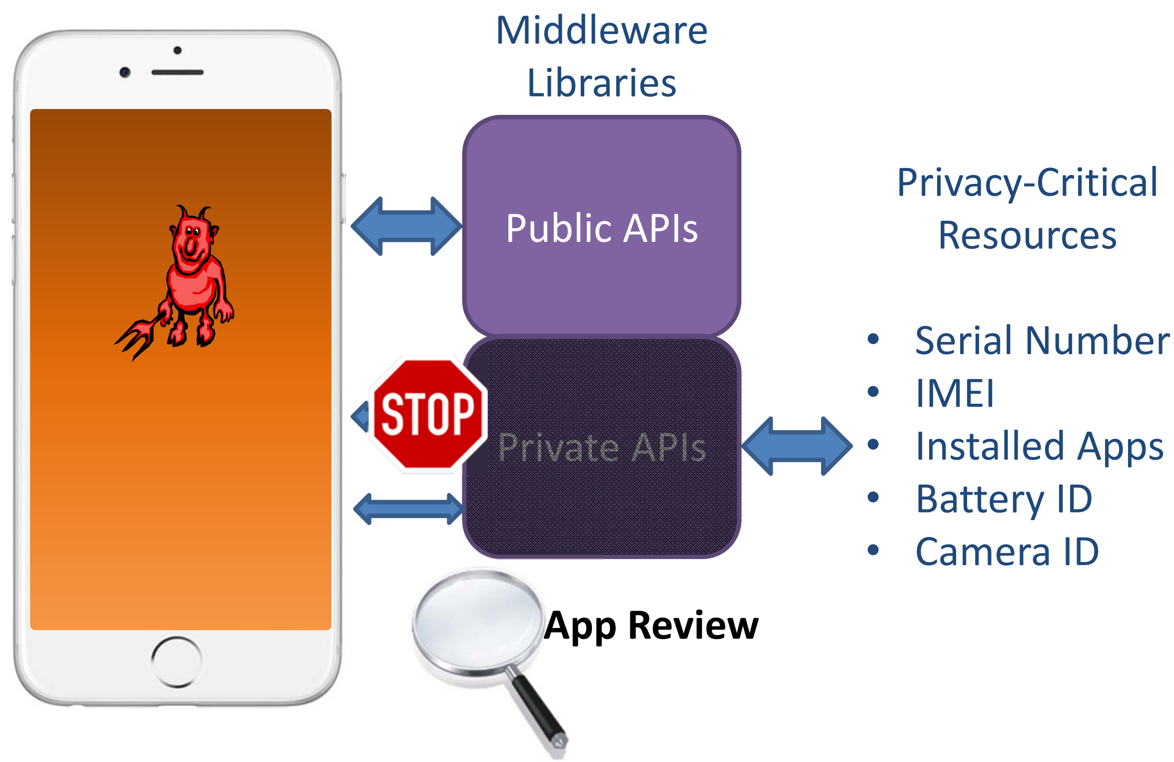


Goal: Private APIs

Middleware APIs reserved only for Apple's internal use

Private APIs are not exported for third-party code to use

Malware are still finding and invoking the Private API despite vetting



Private API Abuse is Difficult To Detect

iOS applications are mostly written in Objective-C &

Objective-C functions can be called sending a message to object

Invoke method **foo** on object **obj** with parameter **param:**

```
objc_msgSend ( obj, “foo”, param );
```

Private API Abuse is Difficult To Detect

iOS applications are mostly written in Objective-C &

Objective-C functions can be called sending a message to object

Invoke method **foo** on object **obj** with parameter **param:**

```
objc_msgSend ( obj, "foo", param );
```

Message Selector = String

Private API Abuse is Difficult To Detect

iOS applications [C &

Objective-C func[ ]essage to object

Static Analysis will not be able to detect complex private API invocations!

Invoke method **foo** on object **obj** with parameter **param:**

```
char sel[3];   strcpy ( sel, "f" );   strcat ( sel, "oo" );
objc_msgSend ( objc, sel, param );
```

+ Obfuscation
+ Encryption

iRiS - Automated Vetting of Private API Abuse

Forced Execution

Drives the app's execution to unresolved API call sites using **forced execution**

Forced Execution

- Dynamic analysis engine that forces a binary to execute
  - Provide no inputs or any environment setup
    - Use random inputs
  - Force branch outcomes at a few places

Force Execution: Conditional Message Selector

```
char hidden[] = “\x73\x68\x75\x74\x64\x6f\x77”
                “\x6e\x44\x65\x76\x69\x67\xb7”;
getAppsOrShutdown( void * obj, int key )
{
  char * sel;
  if( time() == DEC_21_2017)
  {
    sel = XOR(hidden, key);
  }
  else
  {
     sel = “allInstalledApplications”;
  }
  objc_msgSend( obj, sel );
}
```

iRiS must resolve the value of **sel** at this call

Force Execution: Conditional Message Selector

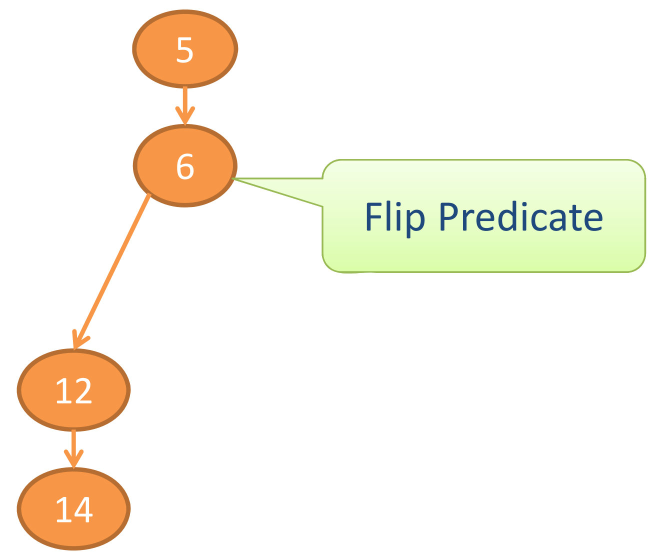
```
char hidden[] = “\x73\x68\x75\x74\x64\x6f\x77”
                “\x6e\x44\x65\x76\x69\x67\xb7”;
getAppsOrShutdown( void * obj, int key )
{
  char * sel;
  if( time() == DEC_21_2017)
  {
    sel = XOR(hidden, key);
  }
  else
  {
     sel = “allInstalledApplications”;
  }
  objc_msgSend( obj, sel );
}
```

Not Resolvable by Static Analysis

Force Execution: Conditional Message Selector

```
char hidden[] = “\x73\x68\x75\x74\x64\x6f\x77”
                “\x6e\x44\x65\x76\x69\x67\xb7”;
getAppsOrShutdown( void * obj, int key )
{
  char * sel;
  if( time() == DEC_21_2017)
  {
    sel = XOR(hidden, key);
  }
  else
  {
     sel = “allInstalledApplications”;
  }
  objc_msgSend( obj, sel );
}
```

Most Executions follow the false branch

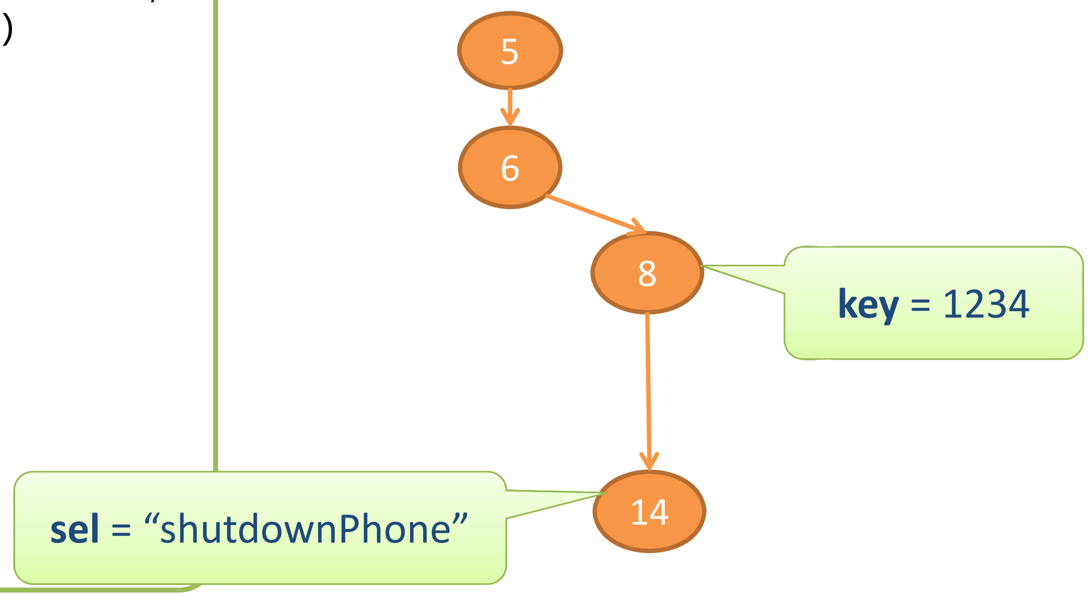


iRiS determines if a new call target can be resolved by flipping a predicate

Force Execution: Conditional Message Selector

```
char hidden[] = “\x73\x68\x75\x74\x64\x6f\x77”
                “\x6e\x44\x65\x76\x69\x67\xb7”;
getAppsOrShutdown( void * obj, int key )
{
  char * sel;
  if( time() == DEC_21_2017)
  {
    sel = XOR(hidden, key);
  }
  else
  {
     sel = “allInstalledApplications”;
  }
  objc_msgSend( obj, sel );
}
```

Force Execution
Resolved Code Path



PURDUE UNIVERSITY

Crash-free Execution Model

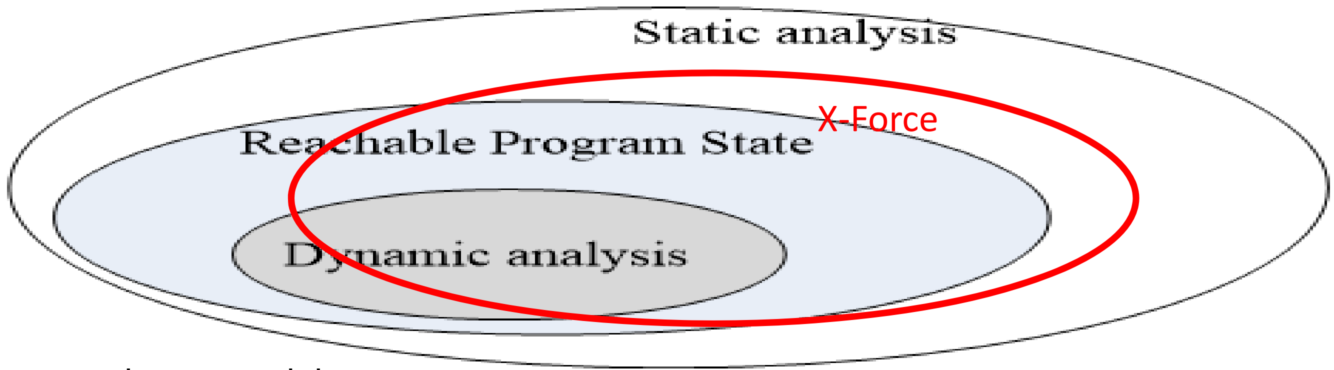
- Ideas on memory access exception
  - Skip it?
    - A lot of following exceptions, cascading effect on program state corruption
    - Lose heap data
  - Allocate a piece of memory on demand
    - It is not sufficient by just fixing the corrupted pointer itself
    - Fix the other correlated pointers

```
1. if (time() == DEC_21_2017)
2.     p=0;
3. q=p+5;
4. *q=…
```

Flipped to true

Crash on *q

The Essence of X-Force



- X-Force is neither sound nor complete
  - Unsound due to path infeasibility and violation of input precondition
  - Incomplete due to the prohibitively large search space
- A pragmatic solution for certain applications
  - Fast
  - Path feasibility may be violated at only a small number of places
    - Less than 10 dynamic predicates out of hundreds of thousands.
  - Naturally handle packed, obfuscated, and even self-modifying binaries
  - Existing dynamic analysis can be easily ported to X-Force
  - Good at exposing behavior, cannot generate exploit inputs

iRiS Evaluation

Vetting platform: iPad 4 + Our ported Valgrind (iOS)

2019 free apps from official App Store

- 9 categories
- Popular apps listed in iTunes Preview
- Crawled in March 2015
- App size: 1-80MB, median 3MB

iRiS Evaluation

149 (7%) applications use private APIs

- Identified a total number of 153 private APIs

Apple brought down most of these apps from their store after our paper was published

- http://www.digitaljournal.com/technology/apple-blacklists-hundreds-of-apps-that-stole-personal-user-data/article/446995

Private APIs: Access User Identification

Framework	API Name	Functionality	#Apps
SpringBoardServices	SBSSpringBoardServerPort	Initialize port with Springboard	3
	SBSCopyApplicationDisplayIdentifiers	Obtain ids of all running apps	3
	SBFrontmostApplicationDisplayIdentifier	Obtain id of the front most app	3
	SBSCopyLocalizedApplicationNameForDisplayIdentifier	Get app name from its id	33
MobileCoreServices	[LSApplicationWorkspace defaultWorkspace]	Obtain the default workspace object	2
	[LSApplicationWorkspace allApplications]	Get all installed apps	1
	[LSApplicationWorkspace allInstalledApplications]	Get all installed apps	1
	[LSApplicationWorkspace applicationIsInstalled:]	Check if a specific app is installed	1

PURDUE UNIVERSITY

Private APIs: More User Identification

Framework	API Name	Functionality	#Apps
AppleAccount	[AADeviceInfo appleIDClientIdentifier]	Obtain the Apple ID of the device user	1
AdSupport	[ASIdentifierManager sharedManager]	Obtain a reference to the ASID manager	25
	[ASIdentifierManager advertisingIdentifier]	Obtain the device's ASID	25
	[ASIdentifierManager advertisingTrackingEnabled]	Check if advertising tracking is enabled	23
IOKit	IOMasterPort	Initialize communication with IOKit	21
	IOServiceMatching	Find and open the specified IOService object	21
	IOServiceGetMatchingService		21
	IORegistryEntryCreateCFProperty	Locate the specified property (e.g. S/N)	19

PURDUE UNIVERSITY

Range of Obfuscation Tactics

strcat

snprintf

[NSString stringWithFormat:]

Custom method for “concat and invoke”

Disappointingly Simple

Somewhat Interesting

Range of Obfuscation Tactics

strcat

snprintf

[NSString stringWithFormat:]

Custom method for “concat and invoke”

XOR Encryption

RC4 Encryption

Disappointingly Simple

Somewhat Interesting

Outline

Temporal Dynamic Analysis:

analyze program execution history

1. Attack Investigation

2. Application vetting

Spatial Dynamic Analysis:

analyze a snapshot of program execution state

3. Memory forensics

Dynamic Analyses

Outline

Temporal Dynamic Analysis:

analyze program execution history

1. Audit logging

2. Forced execution

Spatial Dynamic Analysis:

analyze a snapshot of program execution state

3. Memory forensics

Dynamic Analyses

Execution State (in Memory) Contains Rich Information

Running Processes

Volatile IPC Data

Executing Malware

Decrypted Private Data

Open Network Connections

Encryption Keys

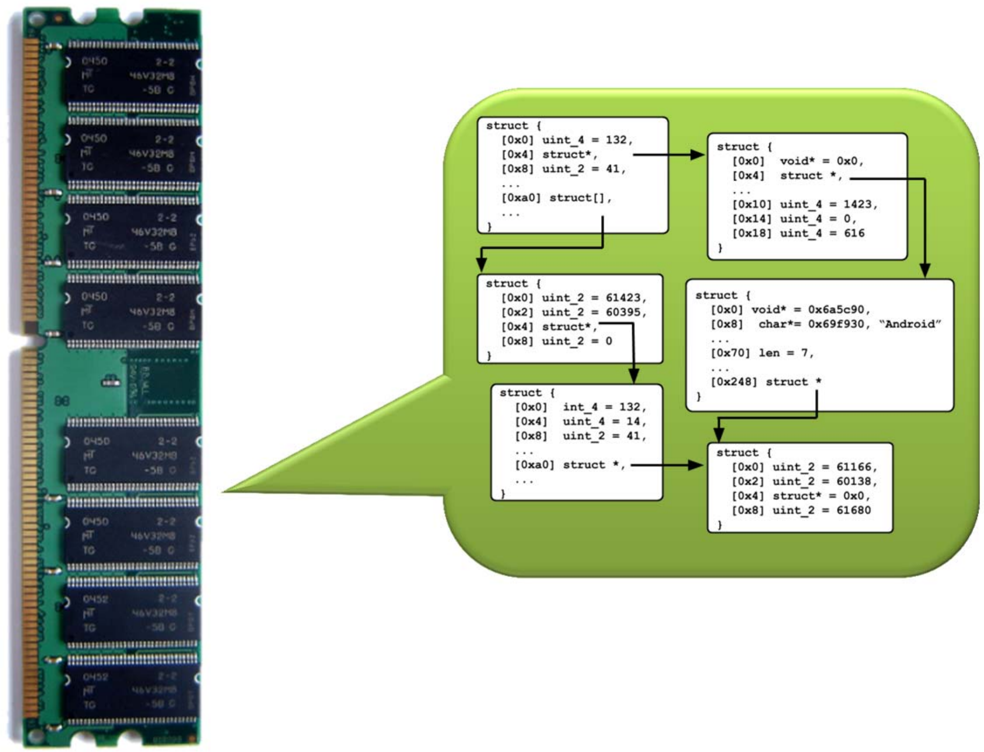
Application Data
(Browser History, Chat Logs, …)

Much More …

PURDUE UNIVERSITY

In-Memory Information Reverse Engieering

Information in memory is stored in Data Structures



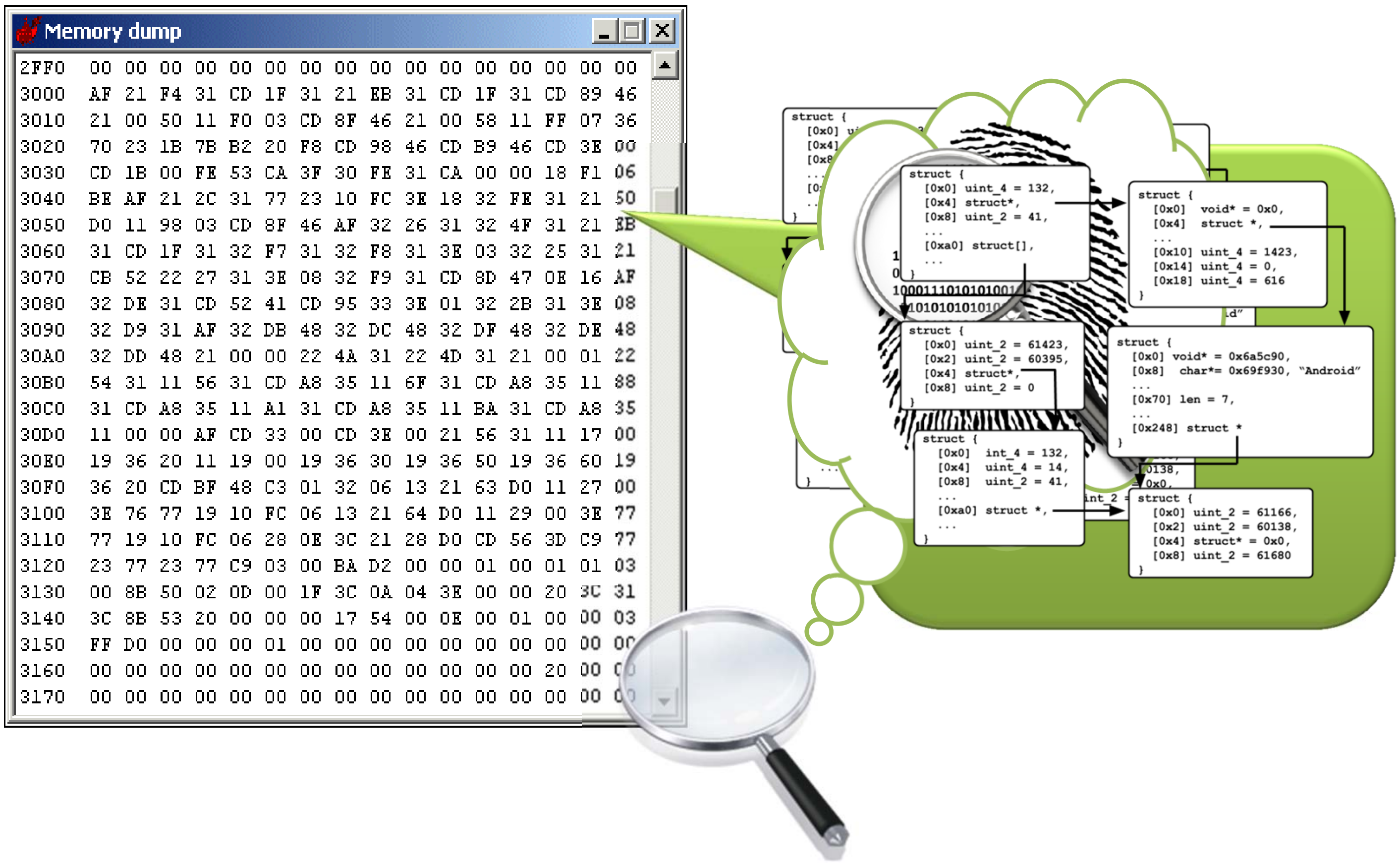
PURDUE UNIVERSITY

In-Memory Information Reverse Engineering

Investigators collect a memory image

Build signatures of data structures to recover

Scan the memory image to find data structure instances

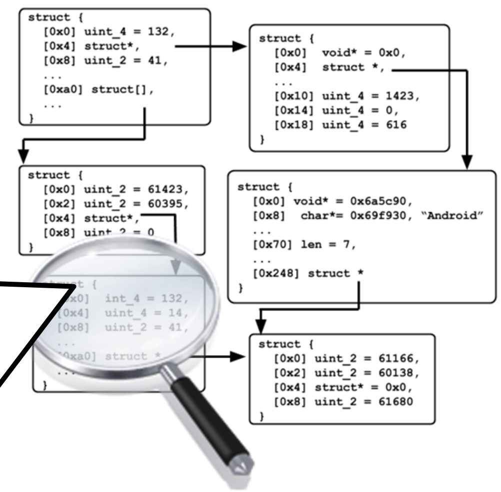


PURDUE UNIVERSITY

(Prior) State of the Art: Data Structure Recovery

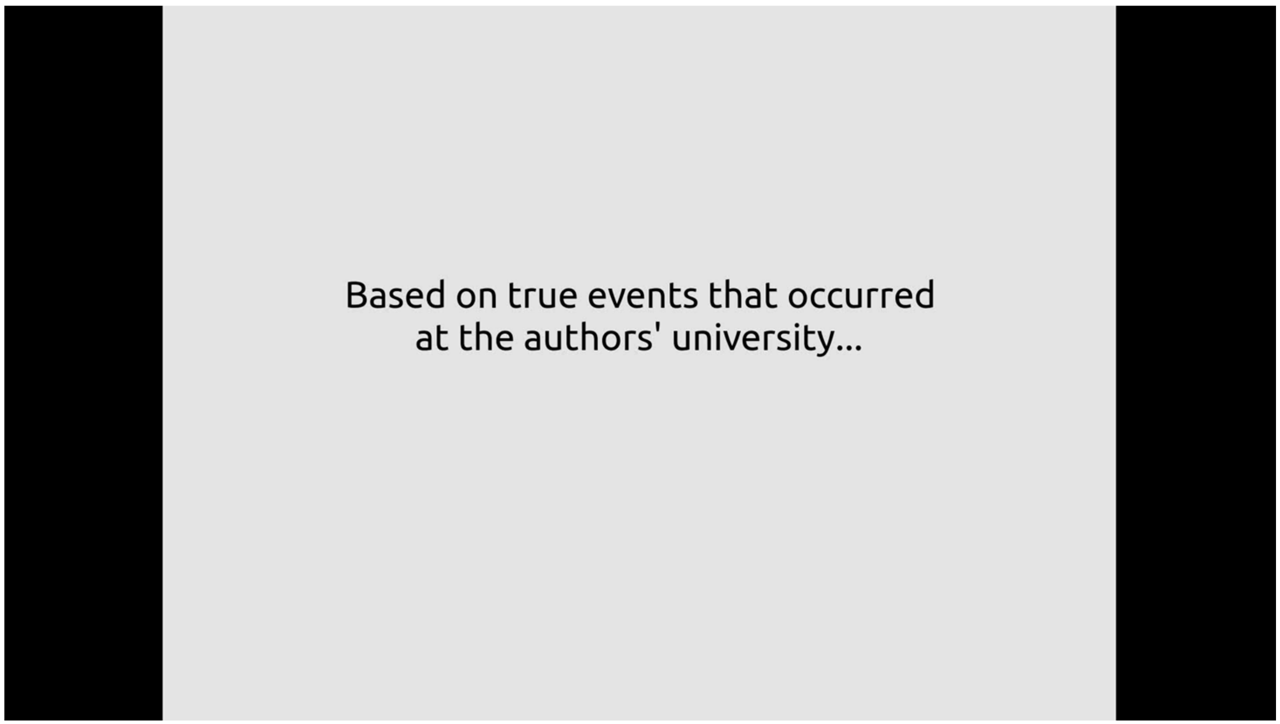
Evidence is recovered from plain-text or self-contained fields

```
struct user_account {
[0x00]  short int u_type;
[0x04]  pid_t u_pid;
[0x08]  char u_line[32];
[0x28]  char uid[4];
[0x2C]  char user[32];
[0x4C]  char password[128];
[0xCC]  char u_host[128];
[0x14C] short e_termination;
...
}
```



PURDUE
UNIVERSITY

A Cyber-Crime

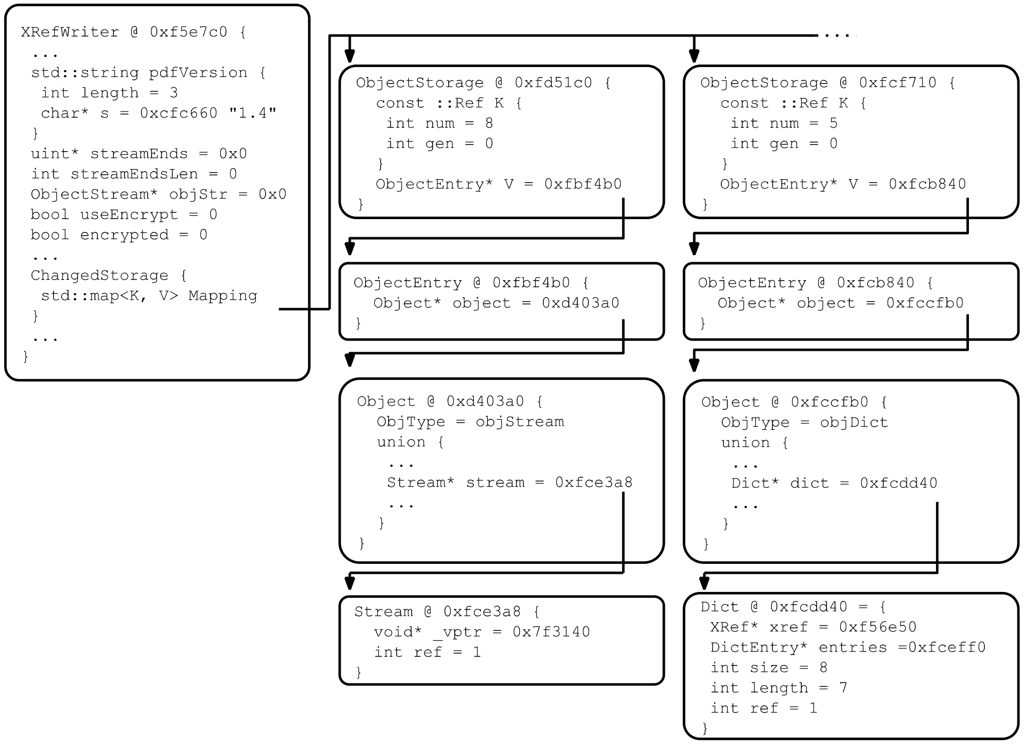


State of the Art … but Limited

Finds raw data structure instances in memory image

Still cannot understand the ***content*** of the data structure!

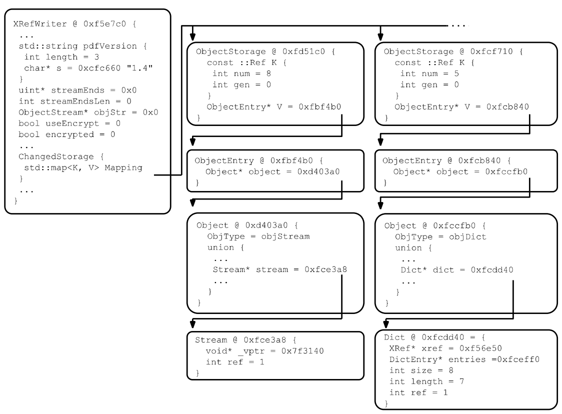
- E.g., images, documents, formatted/encoded data



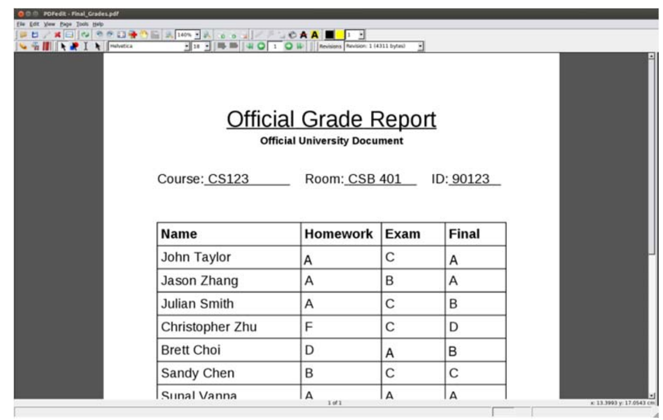
DSCRETE: Content Reverse Engineering

[Usenix Security Best Student Paper]

The application that defined the data structure contains printing/rendering logic for it too!



Rendering Logic Reuse



DSCRETE: Content Reverse Engineering

Scanner+Renderer

[Usenix Security Best Student Paper]

Input: Data Structure Instance

Output: Formatted Content

Printing/Rendering Logic

Program Code

```
struct pdf* my_pdf;
my_pdf = load_pdf_file(…);
main_loop(my_pdf); // User edits PDF
save_pdf_file(my_pdf);
exit(0);
```

Rendering Function

```
save_pdf_file(struct pdf* ptr)
{
  char* buf = format_pdf(ptr);
  fwrite(buf, …);
}
```

Scanner+Renderer

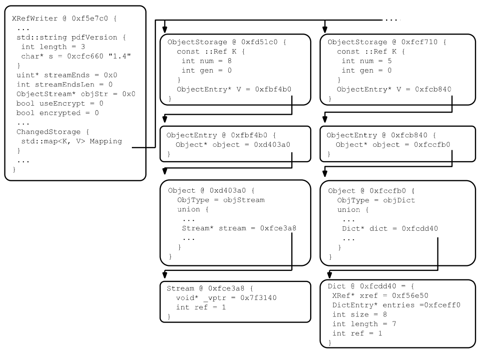
110100000101010
111101001011100
011010010100101
001000100100111

Rendering Function

```
save_pdf_file(struct pdf* ptr)
{
  char* buf = format_pdf(ptr);
  fwrite(buf, …);
}
```

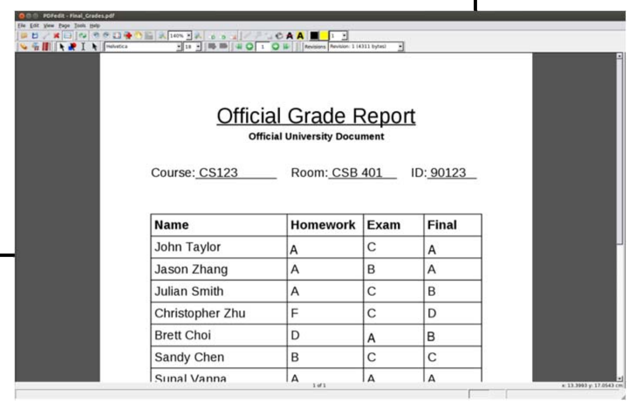
Intuition: Invalid input will crash the rendering logic

Scanner+Renderer



Rendering Function

```
save_pdf_file(struct pdf* ptr)
{
  char* buf =
  fwrite(buf,
}
```



Present every offset of a memory image to the rendering logic &

Reported the valid output

Cross-State Execution

Identified Candidate

Program Code

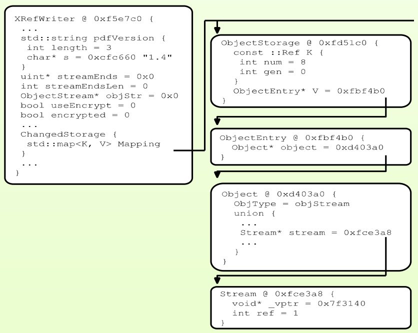
```
struct pdf* my_pdf;
my_pdf = load_pdf_file(…);
main_loop(my_pdf); // User edits PDF
save_pdf_file(my_pdf);

exit(0);


save_pdf_file(struct pdf* ptr)
{
  char* buf = format(ptr);
  fwrite(buf, …);
}
```

Cross-State Execution

App’s Memory



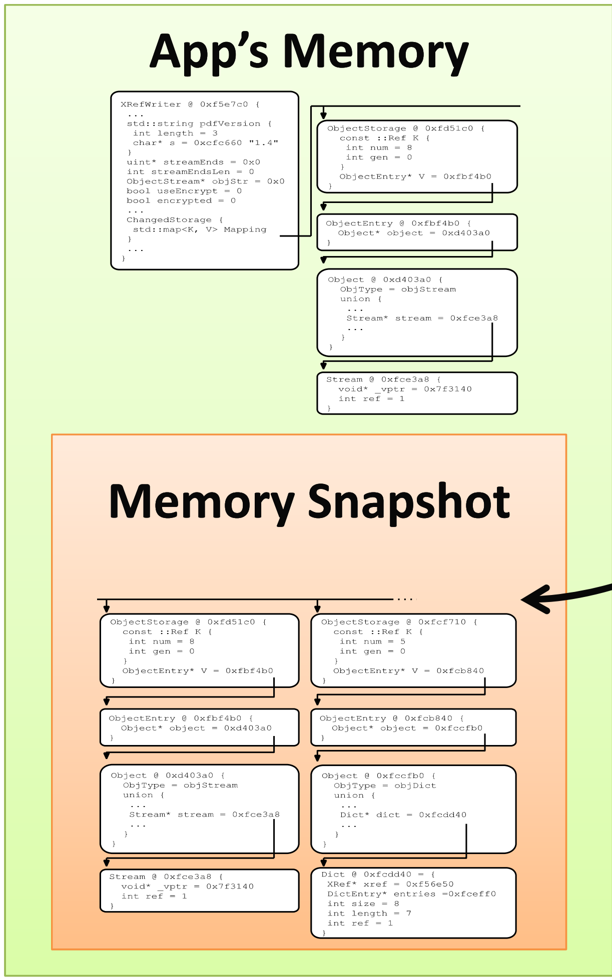
Program Code

```
struct pdf* my_pdf;
my_pdf = load_pdf_file(…);
main_loop(my_pdf); // User edits PDF
save_pdf_file(my_pdf);

exit(0);


save_pdf_file(struct pdf* ptr)
{
  char* buf = format(ptr);
  fwrite(buf, …);
}
```

Cross-State Execution



Program Code

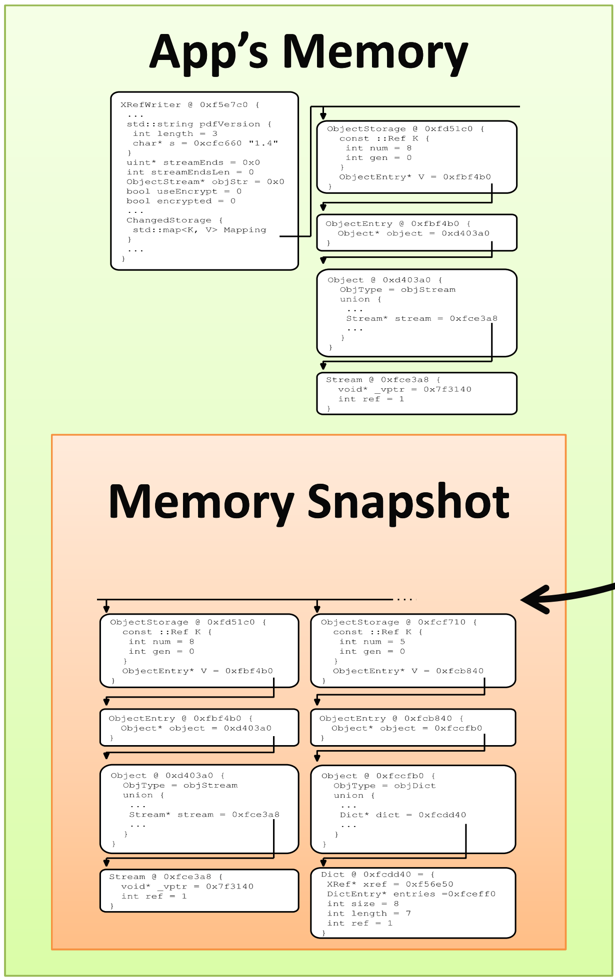
```
struct pdf* my_pdf;
my_pdf = load_pdf_file(…);
main_loop(my_pdf); // User edits PDF
save_pdf_file(my_pdf);

exit(
```

Begin Cross-State Execution!

1. Map in memory snapshot
2. Swap my_pdf pointer

Cross-State Execution



Program Code

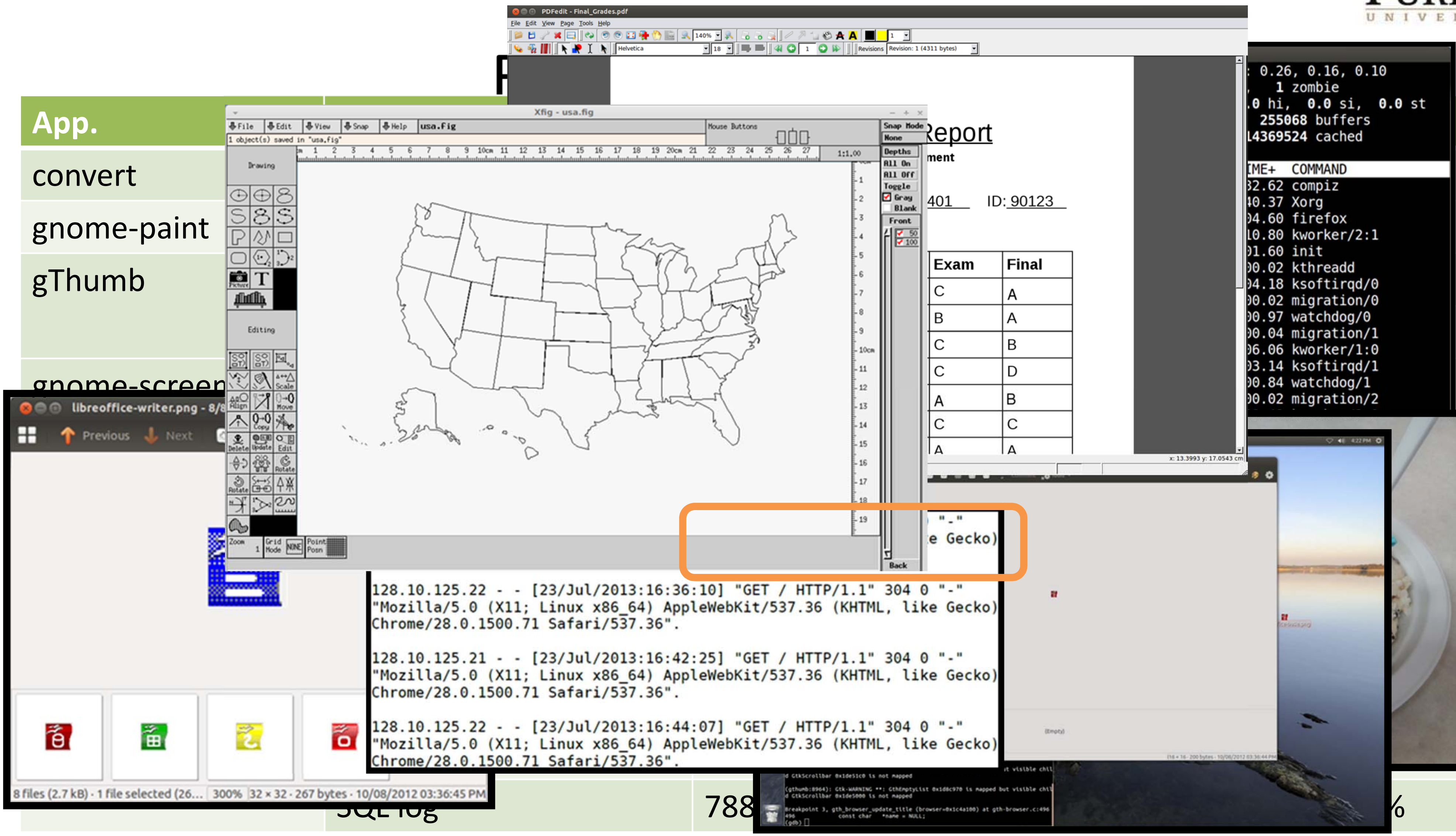
```
struct pdf* my_pdf;
my_pdf = load_pdf_file(…);
main_loop(my_pdf); // User edits PDF
save_pdf_file(my_pdf);

exit(0);


save_pdf_file(struct pdf* ptr)
{
  char* buf = format(ptr);
  fwrite(buf, …);
}
```

Let’s catch that criminal…

App.		
convert		
gnome-paint		
gThumb		
gnome-screen...		
	SQL log	788 ... %

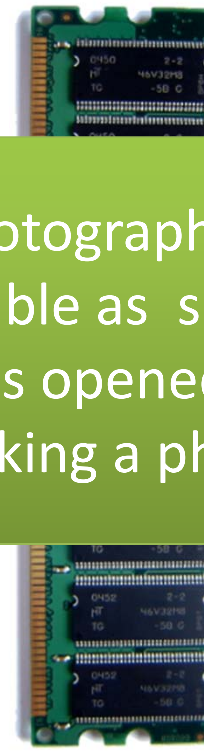


PURDUE UNIVERSITY

PURDUE UNIVERSITY

VCR: Reverse Engineer Preview Pictures

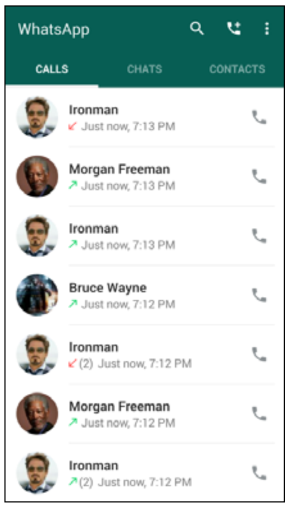
Camera preview is always buffering frames



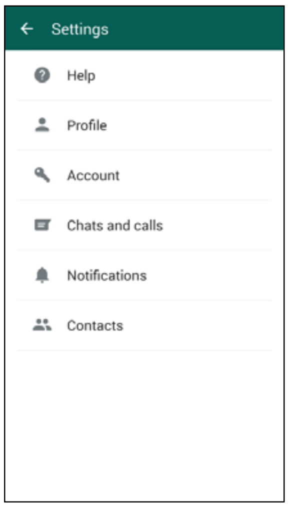
Photographic Evidence is available as soon as a camera app is opened, even without taking a photo or video

PURDUE UNIVERSITY

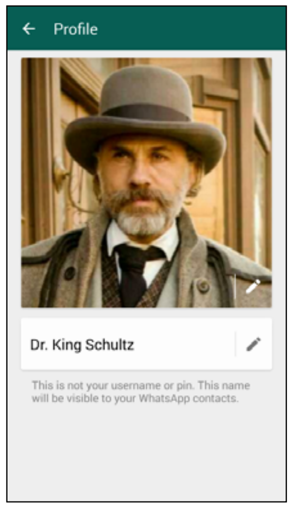
GUI: Reverse Engineer User Interfaces



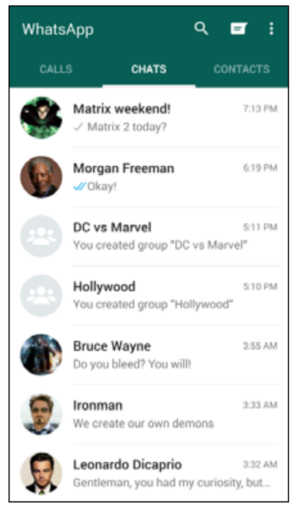
Screen -6



Screen -5

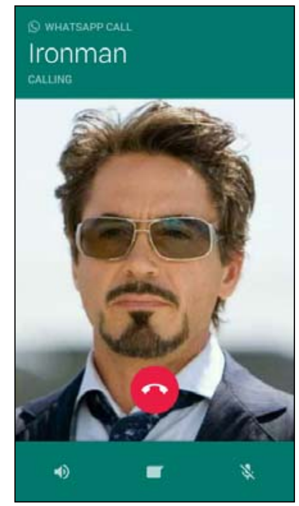


Screen -4



Screen -3

Screen -2



Screen -1

Screen 0

PURDUE UNIVERSITY

Outline

Temporal Dynamic Analysis:

analyze program execution history

1. Attack Investigation

2. Application Vetting

Spatial Dynamic Analysis:

analyze a snapshot of program execution state

3. Memory forensics

Dynamic Analyses

Outline

Temporal Dynamic Analysis:

analyze program execution history

1. Attack investigation

2. Application vetting

Spatial Dynamic Analysis:

analyze a snapshot of program execution state

3. Memory forensics

Dynamic Analyses

Thank you!

Questions?