Dynamic Program Analyses and Their Security Applications

Xiangyu Zhang
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)
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

<table>
<thead>
<tr>
<th>Event</th>
<th>Actor</th>
<th>Action</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Proc_A</td>
<td>recv</td>
<td>&lt;x.x.x.x&gt;</td>
<td></td>
</tr>
<tr>
<td>(2) Proc_A</td>
<td>fork</td>
<td>Malware</td>
<td></td>
</tr>
</tbody>
</table>

Audit Log
Audit Logging

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

```plaintext
(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

- : 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
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: \( \sim 40\% \) run time slow down
  - Space overhead
Dependence Explosion - Example

• Social engineering attack by phishing e-mail
Social engineering attack by phishing email.
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

Choose date range

Beginning date

Ending date

Select software format

Download Activity | Cancel
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
Download Activity

Download account transactions into your Quicken®, QuickBooks® or Intuit ProSeries® and Intuit ProSeries® ProAdvisor® software format, then click on the downloads you want. Transactions from the last 45 days are available.

*Required field

Download Information

Select a format

*Required field
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 ]
The user visited 11 web sites

**Dependence explosion!!**

*(229 IP Addresses)*

---

**Backward Analysis**
[ Backward Analysis ]
[ Backward Analysis ]

Firefox

Malware

Email client

sendmail

procmail

message1

message2

... 

sendmail

procmail

message13

message14

... 

sendmail

procmail

Backward Analysis
[ Backward Analysis ]

Dependence Explosion:
51 Processes, 15 Files,
251 Network addresses, 351 Edges

Malware
[ Backward Analysis ]

Firefox

Malware

Email client

sendmail

procmail

message1

message2

...

sendmail

procmail

message13

message14
[ Backward Analysis ]

Firefox

Malware

Email client

sendmail

procmail

message1

message2

...

sendmail

procmail

message13

message14
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
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
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
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

- **System Calls**
  - Only capture events

- **Syscall Tracepoint**
  - Efficiently transfer data

- **Memory Ring Buffer**
  - Concurrent event processing
  - Lazy flushing

- **Event Consuming threads**
  - Log Buffer

- **User Space**
  - Kernel Space
Example: Avoid *Redundant* Events

1. # vim opening a large file
2. ... 
3. while ((size = read(fd, buf)) > 0):
4. add_node(root, buf)
5. ... 
6. exit();

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

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 = read

PID = 1483, TYPE = SYSCALL: Syscall = read

PID = 1483, TYPE = SYSCALL: Syscall = read

...
Example: Lazy Flushing

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

---

### ProTracer

\[
T[ FD=8 ] = \{ \text{vim} \}
\]

LogBuffer:

\[
T[ FD=8 ] = \{ \text{vim} \}
\]

---

### Logging

\[
\ldots
\]

\[
\text{TYPE = SYSCALL: Syscall = open, FD = 8}
\]

\[
\text{TYPE = SYSCALL: Syscall = write, FD = 8}
\]

\[
\ldots
\]

\[
\text{TYPE = SYSCALL: Syscall = unlink, FD = 8}
\]

\[
\ldots
\]
Evaluation: Storage Efficiency (3 months, client)

BEEP
[NDSS’13]
168,269,688 KB

ProTracer
2,437,010 KB
Evaluation: Run time Efficiency (Client Programs)

Whole system: 7% v.s. 40%
Outline

Temporal Dynamic Analysis:
- analyze program execution history

1. Attack investigation
2. Application vetting

Dynamic Analyses

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%
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
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


Apple is secretive about the specific evaluation ... Security?
Motivation: Total number of iOS applications (2008-2015)

~100,000 apps in 6 months!
~16,666 apps per month!
~555 apps per day!
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....
Goal: Private APIs

Middleware APIs reserved only for Apple’s internal use

Middleware Libraries

Public APIs

Private APIs

Privacy-Critical Resources
- Serial Number
- IMEI
- Installed Apps
- Battery ID
- Camera ID
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

Privacy-Critical Resources
- Serial Number
- IMEI
- Installed Apps
- Battery ID
- Camera ID
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**:

```c
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 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**:  

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

**Static Analysis will not be able to detect complex private API invocations!**
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
1. char hidden[] = “\x73\x68\x75\x74\x64\x6f\x77”
2. ”\x6e\x44\x65\x76\x69\x67\xb7”;  
3. getAppsOrShutdown( void * obj, int key )
4. {
5.   char * sel;
6.   if( time() == DEC_21_2017)
7.   {
8.     sel = XOR(hidden, key);
9.   }
10.  else
11.  {
12.     sel = “allInstalledApplications”; 
13.  }
14.  objc_msgSend( obj, sel );
15. }

iRiS must resolve the value of sel at this call
Force Execution: Conditional Message Selector

1. char hidden[] = "\x73\x68\x75\x74\x64\x6f\x77"
2.                  "\x6e\x44\x65\x76\x69\x67\xb7";
3. getAppsOrShutdown( void * obj, int key )
4. {
5.    char * sel;
6.    if( time() == DEC_21_2017)
7.    {
8.      sel = XOR(hidden, key);
9.    }
10.   else
11.   {
12.      sel = "allInstalledApplications";
13.   }
14.   objc_msgSend( obj, sel );
15. }

Not Resolvable by Static Analysis
Force Execution: Conditional Message Selector

1. char hidden[] = "\x73\x68\x75\x74\x64\x6f\x77"
2.    "\x6e\x44\x65\x76\x69\x67\xb7";
3. getAppsOrShutdown( void * obj, int key )
4. {
5.    char * sel;
6.    if( time() == DEC_21_2017)
7.    {
8.        sel = XOR(hidden, key);
9.    }
10.   else
11.   {
12.        sel = "allInstalledApplications";
13.   }
14.   objc_msgSend( obj, sel );
15. }

iRiS determines if a new call target can be resolved by flipping a predicate
1. char hidden[] = "\x73\x68\x75\x74\x64\x6f\x77"
2.   "\x6e\x44\x65\x76\x69\x67\xb7";
3. getAppsOrShutdown( void * obj, int key )
4. { 
5.   char * sel;
6.   if( time() == DEC_21_2017)
7.   { 
8.     sel = XOR(hidden, key);
9.   } 
10. } else
11. { 
12.   sel = "allInstalledApplications";
13. } 
14. objc_msgSend( obj, sel );
15. }

Force Execution: Conditional Message Selector

key = 1234

sel = "shutdownPhone"
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

## Private APIs: Access User Identification

<table>
<thead>
<tr>
<th>Framework</th>
<th>API Name</th>
<th>Functionality</th>
<th>#Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpringBoardServices</td>
<td>SBSSpringBoardServerPort</td>
<td>Initialize port with Springboard</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>SBSCopyApplicationDisplayIdentifiers</td>
<td>Obtain ids of all running apps</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>SBFrontmostApplicationDisplayIdentifier</td>
<td>Obtain id of the front most app</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>SBSCopyLocalizedApplicationNameForDisplayIdentifier</td>
<td>Get app name from its id</td>
<td>33</td>
</tr>
<tr>
<td>MobileCoreServices</td>
<td>[LSApplicationWorkspace defaultWorkspace]</td>
<td>Obtain the default workspace object</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>[LSApplicationWorkspace allApplications]</td>
<td>Get all installed apps</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>[LSApplicationWorkspace allInstalledApplications]</td>
<td>Get all installed apps</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>[LSApplicationWorkspace applicationIsInstalled:]</td>
<td>Check if a specific app is installed</td>
<td>1</td>
</tr>
</tbody>
</table>
## Private APIs: More User Identification

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

Disappointingly
Simple

strcat
snprintf
[NSString stringWithFormat:]
Custom method for “concat and invoke”

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 ...
In-Memory Information Reverse Engineering

Information in memory is stored in Data Structures
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
(Prior) State of the Art: Data Structure Recovery

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

```c
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;
    ...
}
```
A Cyber-Crime

Based on true events that occurred at the authors' university...
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
The application that defined the data structure contains printing/rendering logic for it too!
**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

**Rendering Function**

```c
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

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

Present every offset of a memory image to the rendering logic & Reported the valid output
Cross-State Execution

```c
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

```
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

Memory Snapshot

Program Code

```c
struct pdf* my_pdf;
my_pdf = load_pdf_file(...);
main_loop(my_pdf); // User edits PDF
save_pdf_file_file(my_pdf);
exit(0);
```

Begin Cross-State Execution!

1. Map in memory snapshot
2. Swap my_pdf pointer
Cross-State Execution

Program Code

```c
struct pdf* my_pdf;
my_pdf = load_pdf_file(...);
main_loop(my_pdf); // User edits PDF
save_pdf_file_file(my_pdf);
exit(0);

save_pdf_file(struct pdf* ptr)
{
    char* buf = format(ptr);
    fwrite(buf, ...);
}
```
Let’s catch that criminal...

Back at the Forensics Lab...
VCR: Reverse Engineer Preview Pictures

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

Camera preview is always buffering frames
GUI: Reverse Engineer User Interfaces
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
Thank you!
Questions?