Forced-Execution of Binaries and iOS Apps to Disclose Malicious Behavior

Xiangyu Zhang
Background & Motivation

- What is binary analysis - analysis on compiled binary software
  - No source code or symbolic information

- Binary analysis has many security applications
  - Exposing malicious/hidden behavior
  - Identifying and patching security vulnerabilities

- Binary analysis is highly challenging
  - No variables
  - No array indexing or data structure field accesses
  - jmp EAX, call EAX
  - Obfuscation, packing, self-modifying, and anti-analysis
Background & Motivation

- **Existing approaches**
  - **Static analysis (IDA)**
    - Examining the code without executing it
  - **Dynamic analysis (Valgrind, PIN)**
    - Testing and evaluation of an application during runtime
  - **Symbolic analysis (BitBlaze, S2E)**
    - Determine what inputs cause each part of the program to execute

<table>
<thead>
<tr>
<th></th>
<th>Good Coverage</th>
<th>Packing &amp; Obfuscation</th>
<th>Precision</th>
<th>Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>✔️</td>
<td>✗</td>
<td>✗</td>
<td>✔️</td>
</tr>
<tr>
<td>Dynamic</td>
<td>✗</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Symbolic</td>
<td>✔️</td>
<td>✗</td>
<td>✔️</td>
<td>✗</td>
</tr>
</tbody>
</table>
Outline

- X-Force: Forced Execution of x86 Binaries [Usenix’14]
  - Design
  - Implementation Challenges
  - Evaluation

- iRiS: Forced Execution of iOS Apps [CCS’15]
  - iOS security
  - Forced execution on iOS platform
Design

- **What is X-Force?**
  - Dynamic analysis engine that forces an x86 binary to execute
    - Provide no inputs or any environment setup
      - Force branch outcomes at a few places
      - Tolerate exceptions
    - Explore different paths
Example  - Hijack the name resolution for a specific domain

```
1 DNSentry *p;
2 void main () {
3    int x = inputInt ();
4    if (C (x))
5        p = (DNSentry *)malloc (...);
6    if (x & CODE_RED) {
7        genName (x, p);
8        hashTablePut (x, p);
9    }
10   ...  
11   hashTablePut (... , o); // o is of type ...
12   ...  
13   s = hashTableGet (y); // y == x through execution
14   if (s)
15      // redirection for the domain specified by s
16      redirection ();
17 }
18 
19 void genName (int x, DNSentry *q) {
20    inputDirectionary ();
21    *(q->name) = ...Lookup (x, date ())...;
22 }
```

- Reads an integer x
- If condition satisfied, a DNS object is allocated to p
- If CODE_RED bit set is in x, get the domain name for the object in p
- Other objects are put into the hash table
- Fetch an object using key y == x
- If an object is fetched successfully, malicious payload triggered
Example – Static Analysis

```c
1  DNSentry *p;
2  void main () {
3      int x = inputInt ();
4      if (C (x))
5          p = (DNSentry *)malloc (...);
6      if (x & CODE_RED) {
7          genName (x, p);
8          hashTablePut (x, p),
9      }
10     ...
11    hashTablePut (... 0); // o is of type T
12    ...
13  s = hashTableGet (y); // y == x through execution
14  if (s)
15      // redirection for the domain specified by s
16      redirection ();
17 }
18
19  void genName (int x, DNSentry *q) {
20      inputDirectionary ();
21      *(q->name) = ... Lookup (x, date ())...;
22 }
```
Example – Dynamic Analysis

```c
1  DNSentry *p;
2  void main () {
3      int x = inputInt ();
4      if (C (x))
5          p = (DNSentry *)malloc (...);
6      if (x & CODE_RED) {
7          genName (x, p);
8          hashTablePut (x, p);
9      }
10     ...
11     hashTablePut (... , o); // o is of type T
12     ...
13     s = hashTableGet (y); // y == x through execution
14     if (s)
15         // redirection for the domain specified by s
16         redirection ();
17  }
18
19  void genName (int x, DNSentry *q) {
20      inputDirectionary ();
21      *(q->name) = ...Lookup (x, date ())...;
22  }
```

If CODE_RED bit is not set in x, an object with key x is not put into hash table. Malicious payload is not triggered.
Example – Symbolic Analysis

```
1 DNSentry *p;
2 void main () {
3     int x = inputInt ();
4     if (C (x))
5         p = (DNSentry *)malloc (...);
6     if (x & CODE_RED) {
7         genName (x, p);
8         hashTablePut (x, p);
9     }
10     ...
11     hashTablePut (... , o); // o is of type T
12     ...
13     s = hashTableGet (y); // y == x through execution
14     if (s)
15         // redirection for the domain specified by s
16         redirection ();
17 }
18 void genName (int x, DNSentry *q) {
19     inputDirectionary ();
20     *(q->name) = ...Lookup (x, date ())...;
21 }
```
Example – X-Force

```c
1 DNSentry *p;
2 void main () {
3    int x = inputInt ();
4    if (C (x))
5       p = (DNSentry *)malloc (...);
6    if (x & CODE RED) {
7       genName (x, p);
8       hashTablePut (x, p);
9    }
10   ...
11  hashTablePut (... , o); // o is of type T
12  ...
13  s = hashTableGet (y); // y == x through execution
14  if (s)
15     // redirection for the domain specified by s
16     redirection ();
17 }
18 }
19 void genName (int x, DNSentry *q) {
20    inputDirectionary ();
21    *(q->name) = ...Lookup (x, date () ) ...
22 }
```
Example – X-Force

```c
1. DNSentry *p;
2. void main () {
3.    int x = inputInt ();
4.    if (C (x)) {  
5.        p = (DNSentry *)malloc (...);
6.        if (x & CODE_RED) {
7.            genName (x, p);
8.            hashTablePut (x, p);
9.        }
10.   ...  
11.   hashTablePut (..., o); // o is of type T
12.   ...
13.   s = hashTableGet (y); // y == x through execution
14.   if (s)
15.      // redirection for the domain specified by s
16.      redirection ();
17.   }
18. 
19. void genName (int x, DNSentry *q) {
20.    inputDirectionary ();
21.    *(q->name) = ...Lookup (x, date ())...
22. }
```
Example – X-Force

```c
1 DNSentry *p;
2 void main () {
3     int x = inputInt ();
4     if (C (x))
5         p = (DNSentry *)malloc (...);
6     if (x & CODE RED) {
7         genName (x, p);
8         hashTablePut (x, p);
9     }
10    ...
11    hashTablePut (... , o); // o is of type T
12    ...
13    s = hashTableGet (y); // y == x through execution
14    if (s)
15        // redirection for the domain specified by s
16        redirection ();
17 }
18
19 void genName (int x, DNSentry *q) {
20     inputDirectory ();
21     *(q->name) = ...Lookup (x, date (...));
22 }
```
Crash-free Execution

- 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
Example – Dataflow

```c
1 DNSSentry *p;
2 void main () {
3    int x = inputInt ();
4    if (C (x)) {
5        p = (DNSSentry *)malloc (...);
6        if (x & CODE RED) {
7            genName (x, p);
8            hashTablePut (x, p);
9        }
10    ...
11    hashTablePut (... , o); // o is of type T
12    ...
13    s = hashTableGet (y); // y == x through execution
14    if (s)
15        // redirection for the domain specified by s
16        redirection ();
17    }
18    }
19 void genName (int x, DNSSentry *q) {
20    inputDirectionary (y );
21    *(q->name) = ...Lookup (x, date ())...;
22    }
```
Crash-free Execution

Observations

- Some pointers are correlated
- Correlated pointers are only linearly correlated
  - No multiplication/division

Solution – Linear set tracing

1. Memories/registers that are linearly correlated are put into a set
   - Copying (e.g. b = a)
   - Adding or subtracting (e.g. q = p +/- 4)
2. When memory exception occurs, recover values for elements based on maintained linear sets
Example – Linear Set Tracing

```c
1 DNSentry *p;
2 void main () {
3     int x = inputInt ();
4     if (C (x))
5         p = (DNSentry *)malloc (...);
6     if (x & CODE RED){
7         genName (x, p);
8         hashtablePut (x, p);
9     }
10     ...  
11     hashtablePut (... , o); // o is of type T
12     ...  
13     s = hashtableGet (y); // y == x through execution
14     if (s)
15         //redirection for the domain specified by s
16         redirection ()
17 }  
18  
19 void genName (int x, DNSentry *q) {
20     inputDirectionary ();
21     *(q->name) = ...Lookup (x, date (...);
22 }  
```

Memory write exception, crash!

p
q
q->name
Path Exploration

- Exploration algorithms
  - Branch coverage driven algorithm
    - Number of executions - $O(n)$
      - $n$ denotes the number of predicates
  - Exponential search algorithm - $O(2^n)$

- Implement a taint analysis subsystem
  - Determine branches that are input related
The Essence of X-Force

- Reachable program state
  - Ideal coverage
- Static analysis
  - Over-approximate coverage
- Dynamic analysis
  - Under-approximate coverage
- X-Force
  - Practicality
The Essence of X-Force

- X-Force is important in practice
  - Results are not affected much by infeasible paths
    - Only a small number of predicates are switched
  - Fast
  - Naturally handle packed, obfuscated, and even self-modifying binaries
  - Existing dynamic analysis can be easily ported to X-Force
Outline

- Design
- Implementation Challenges
- Evaluation
Implementation Challenges

- Loops
  - If the loop bound is computed from input, it may be a corrupted value
    - Use taint analysis subsystem to determine if it’s input related
    - If so, set the loop bound to a pre-defined constant

- Recursions
  - Maintain call stack during execution to detect recursion
  - If recursion is too deep, skip calling into it by simulating a return instruction
Outline

- Design
- Technical Challenges
- Evaluation
Evaluation: Malware Analysis

- X-Force discovers more lib calls than IDA for packed/obfuscated malware
- X-Force beats dynamic native run for all the programs
X-Force Demo

- Using X-Force to analyze real-world APT malware
  - Revealed APT payloads *missed by a SANS Institute Report on the same malware*
Outline

- X-Force: Forced Execution of x86 Binaries [Usenix’14]
  - Design
  - Implementation Challenges
  - Evaluation

- iRiS: Forced Execution of iOS Apps [CCS’15]
  - iOS security
  - Forced execution on iOS platform
Motivation

Growth of the total number of iOS applications (2008-2015)
iOS Security

- App Review
- Runtime protection
  - Sandboxing
  - Mandatory Access Control (MAC)
    - Entitlement
  - Data execution prevention (DEP)
  - Address Space Layout Randomization (ASLR)
Motivation: Private API

- In Android, access to system resources is through IPC with Service Manager
  - Permission based security

- In iOS, access to system resources is through APIs that directly interact with the kernel
  - Many of them grant access to security-critical resources
    - Device serial number, IMEI, battery ID, camera ID
Motivation: App Review

- **App Review**
  - A mandatory process enforced by Apple
  - Checks for private API uses in third-party applications before putting them on App Store

- Few malicious applications have been reported so far
  - App Review effective? **No!**
  - Can be easily bypassed with API obfuscation
    - Wang et.al [USENIX Security’13]
    - Han et.al [ACNS’13]
    - Zheng et.al [ASIACCS’15]
iRiS Overview

- An automated system for vetting private API uses in iOS applications

- Uses a combination of static and dynamic analysis
  - iOS applications are usually large (>1MB)
  - Static analysis
    - Fast, resolves most API calls
    - Vulnerable to obfuscation
  - Dynamic analysis by forced execution
    - Slower, but more powerful to resolve the few remaining ones

- Ported Valgrind to iOS to enable dynamic binary instrumentation for the dynamic analysis
Challenges: Objective-C Method Calls

- iOS applications are mostly written in Objective-C
  - Objective-C function call is sending message to object

- Invoke method $foo$ on object $x$ with parameter $p$
  - $\Rightarrow$ `objc_msgSend(x, "foo", p);`

- The method name (i.e. message selector) could be constructed dynamically at runtime
  - char sel[16]; strcpy(sel, "hello"); strcat(sel, "World");
  - `objc_msgSend(objc, sel, param);`
Challenges: iOS Application Execution

- Driven by external events
  - Execution of event handlers

- Loading of UI objects from Next Interface Builder (NIB) resource files
  - Involves many implicit API invocations
    - Initialization of objects
    - Event handler registrations
    - Connections between UI objects and the corresponding controller objects
  - Such invocations do not appear in the application’s binary
Overview

iOS Packaged Apps

NIB Resources

Resource Analysis

Implicit Call Targets

Iterative Forced Execution

Intra-procedural CFGs

Static Analysis

Call Graph

Resolved Call Targets

Encrypted Binary

Binary Executable
Static Analysis

- Resolves call targets as many as possible
  - Objective-C messages

- Generates graphs to guide dynamic analysis in later stage
  - Intra-procedural CFGs (by IDA Pro)
  - Call graph

- Intra-procedural backward slicing + forward constant propagation
Static Analysis: Backward Slicing

- Example: [s addSubview:v]

```
    __text:0000AD1A       MOVW            R1, #0xB7D8
    __text:0000AD1E       ADD             R0, PC
    __text:0000AD20       MOVT.W          R1, #0x2E
    __text:0000AD24       ADD             R1, PC
    __text:0000AD26       LDR             R0, [R0]
    __text:0000AD28       LDR             R6, [R1]
    __text:0000AD2A       LDR             R2, [R4, R0]
    __text:0000AD2C       MOV             R0, R5
    __text:0000AD2E       MOV             R1, R6
    __text:0000AD30       BLX             _objc_msgSend
```
Static Analysis: Backward Slicing

- Example: [s addSubview:v]

Selector in R1, Slicing Criterion = (0x0000AD30, {R1})
Example: `[s addSubview:v]`

```
.text:0000AD1A        MOVW          R1, #0xB7D8
.text:0000AD1E        ADD            R0, PC
.text:0000AD20        MOVT.W        R1, #0x2E
.text:0000AD24        ADD            R1, PC
.text:0000AD26        LDR            R0, [R0]
.text:0000AD28        LDR            R6, [R1]
.text:0000AD2A        LDR            R2, [R4,R0]
.text:0000AD2C        MOV            R0, R5
.text:0000AD2E        MOV            R1, R6
.text:0000AD30        BLX            _objc_msgSend

{R1}
```
Static Analysis: Backward Slicing

Example: `[s addSubview:v]`

```assembly
__text:0000AD1A   MOVW          R1, #0xB7D8
__text:0000AD1E   ADD           R0, PC
__text:0000AD20   MOVT.W        R1, #0x2E
__text:0000AD24   ADD           R1, PC
__text:0000AD26   LDR           R0, [R0]
__text:0000AD28   LDR           R6, [R1]
__text:0000AD2A   LDR           R2, [R4, R0]
__text:0000AD2C   MOV           R0, R5
__text:0000AD2E   MOV           R1, R6
__text:0000AD30   BLX           _objc_msgSend

{R6}
```
Static Analysis: Backward Slicing

Example: `[s addSubview:v]`

```
__text:0000AD1A     MOVW         R1, #0xB7D8
__text:0000AD1E     ADD          R0, PC
__text:0000AD20     MOVT.W       R1, #0x2E
__text:0000AD24     ADD          R1, PC
__text:0000AD26     LDR          R0, [R0]
__text:0000AD28     LDR          R6, [R1]
__text:0000AD2A     LDR          R2, [R4,R0]
__text:0000AD2C     MOV          R0, R5
__text:0000AD2E     MOV          R1, R6
__text:0000AD30     BLX           _objc_msgSend

{R6}
```
Static Analysis: Backward Slicing

Example: [s addSubview:v]

```
.text:0000AD1A     MOVW            R1, #0xB7D8
.text:0000AD1E     ADD             R0, PC
.text:0000AD20     MOVT.W          R1, #0x2E
.text:0000AD24     ADD             R1, PC
.text:0000AD26     LDR             R0, [R0]
.text:0000AD28     LDR             R6, [R1]
.text:0000AD2A     LDR             R2, [R4,R0]
.text:0000AD2C     MOV             R0, R5
.text:0000AD2E     MOV             R1, R6
.text:0000AD30     BLX             _objc_msgSend

{R6}
```
Static Analysis: Backward Slicing

Example: [s addSubview:v]

```
__text:0000AD1A           MOVW            R1, #0xB7D8
__text:0000AD1E           ADD             R0, PC
__text:0000AD20           MOVT.W          R1, #0x2E
__text:0000AD24           ADD             R1, PC
__text:0000AD26           LDR             R0, [R0]
__text:0000AD28           LDR             R6, [R1]
__text:0000AD2A           LDR             R2, [R4,R0]
__text:0000AD2C           MOV             R0, R5
__text:0000AD2E           MOV             R1, R6
__text:0000AD30           BLX             _objc_msgSend

{R1}
```
Static Analysis: Backward Slicing

Example: `[s addSubview:v]`

```
__text:0000AD1A   MOVW          R1, #0xB7D8
__text:0000AD1E   ADD           R0, PC
__text:0000AD20   MOVT.W        R1, #0x2E
__text:0000AD24   ADD           R1, PC
__text:0000AD26   LDR           R0, [R0]
__text:0000AD28   LDR           R6, [R1]
__text:0000AD2A   LDR           R2, [R4,R0]
__text:0000AD2C   MOV           R0, R5
__text:0000AD2E   MOV           R1, R6
__text:0000AD30   BLX           _objc_msgSend

{R1}
```
Static Analysis: Backward Slicing

Example: [s addSubview:v]

```
__text:0000AD1A           MOVW            R1, #0xB7D8
__text:0000AD1E           ADD             R0, PC
__text:0000AD20           MOVT.W          R1, #0x2E
__text:0000AD24           ADD             R1, PC
__text:0000AD26           LDR             R0, [R0]
__text:0000AD28           LDR             R6, [R1]
__text:0000AD2A           LDR             R2, [R4,R0]
__text:0000AD2C           MOV             R0, R5
__text:0000AD2E           MOV             R1, R6
__text:0000AD30           BLX             _objc_msgSend
```

{R1}
Static Analysis: Backward Slicing

Example: `[s addSubview:v]`

```
__text:0000AD1A       MOVW            R1, #0xB7D8
__text:0000AD1E       ADD             R0, PC
__text:0000AD20       MOVT.W          R1, #0x2E
__text:0000AD24       ADD             R1, PC
__text:0000AD26       LDR             R0, [R0]
__text:0000AD28       LDR             R6, [R1]
__text:0000AD2A       LDR             R2, [R4,R0]
__text:0000AD2C       MOV             R0, R5
__text:0000AD2E       MOV             R1, R6
__text:0000AD30       BLX             _objc_msgSend

{R1(lower half)}
```
Static Analysis: Backward Slicing

- Example: `[s addSubview:v]`

```assembly
__text:0000AD1A          MOVW          R1, #0xB7D8
__text:0000AD1E          ADD           R0, PC
__text:0000AD20          MOVT.W        R1, #0x2E
__text:0000AD24          ADD           R1, PC
__text:0000AD26          LDR           R0, [R0]
__text:0000AD28          LDR           R6, [R1]
__text:0000AD2A          LDR           R2, [R4,R0]
__text:0000AD2C          MOV           R0, R5
__text:0000AD2E          MOV           R1, R6
__text:0000AD30          BLX           _objc_msgSend

{R1(lower half)}
```
Static Analysis: Backward Slicing

- Example: [s addSubview:v]

```assembly
__text:0000AD1A  MOVW            R1, #0xB7D8
__text:0000AD1E  ADD             R0, PC
__text:0000AD20  MOVT.W          R1, #0x2E
__text:0000AD24  ADD             R1, PC
__text:0000AD26  LDR             R0, [R0]
__text:0000AD28  LDR             R6, [R1]
__text:0000AD2A  LDR             R2, [R4, R0]
__text:0000AD2C  MOV             R0, R5
__text:0000AD2E  MOV             R1, R6
__text:0000AD30  BLX             _objc_msgSend

{}
```
Static Analysis: Forward Const Propagation

Example: [s addSubview:v]

```
__text:0000AD1A           MOVW            R1, #0xB7D8
__text:0000AD20           MOVT.W          R1, #0x2E
__text:0000AD24           ADD             R1, PC
__text:0000AD28           LDR             R6, [R1]
__text:0000AD2E           MOV             R1, R6
__text:0000AD30           BLX             _objc_msgSend
...
__objc_selrefs:002F6500 0x248DEF
...
__objc_methname:00248DEF 'addSubview:'
```
Static Analysis: Forward Const Propagation

- Example: [s addSubview:v]

```
__text:0000AD1A           MOVW            R1, #0xB7D8
__text:0000AD20           MOVT.W          R1, #0x2E
__text:0000AD24           ADD             R1, PC
__text:0000AD28           LDR             R6, [R1]
__text:0000AD2E           MOV             R1, R6
__text:0000AD30           BLX             _objc_msgSend
...
__objc_selrefs:002F6500  0x248DEF
...
__objc_methname:00248DEF   'addSubview:'
```

R1 = 0x????B7D8
Static Analysis: Forward Const Propagation

Example: [s addSubview:v]

R1 = 0x2EB7D8
Static Analysis: Forward Const Propagation

Example: [s addSubview:v]

```
MOVW R1, #0xB7D8
MOVT.W R1, #0x2E
ADD R1, PC
LDR R6, [R1]
MOV R1, R6
BLX _objc_msgSend
```

```
__objc_selrefs:002F6500 0x248DEF

__objc_methname:00248DEF 'addSubview:'
```

```
R1 = 0x2EB7D8 + 0xAD28 = 0x2F6500
```
Static Analysis: Forward Const Propagation

Example: `[s addSubview:v]`

```
__text:0000AD1A          MOVW            R1, #0xB7D8
__text:0000AD20          MOVT.W          R1, #0x2E
__text:0000AD24          ADD             R1, PC
__text:0000AD28          LDR             R6, [R1]
__text:0000AD2E          MOV             R1, R6
__text:0000AD30          BLX             _objc_msgSend
...
__objc_selrefs:002F6500   0x248DEF
...
__objc_methname:00248DEF  ‘addSubview:’
```

R1 = 0x2F6500, R6 = [0x2F6500] = 0x248DEF
Static Analysis: Forward Const Propagation

Example: [s addSubview:v]

R1 = 0x248DEF ('addSubview'), R6 = 0x248DEF
Dynamic Analysis

- Resolve the call targets that are not statically resolvable

- Guided forced execution
  - Start with a natural execution
  - For each unresolved call site, we guide the force-execution engine to follow a path that leads to the site
  - Handling exceptions in a way similar to X-Force

- Iterative
  - Newly found call edges are merged back to CG
Dynamic Analysis: Initial CG & CFGs

```
foo () {
    ...  
    if (...) 
    a () 
    else {
        if (...) 
        b () 
        else 
        b () 
    }
}
```
Dynamic Analysis: Natural Execution

Foo () {
    ...
    if (…) {
        A()
    } else {
        if (…) {
            B()
        } else {
            B()
        }
    }
}
Dynamic Analysis: Natural Execution

Given a target call site, we determine if the target can be reachable from a predicate outcome. If not, we switch the predicate.

Foo () {
1   ...
2   if (...) 
3      A()  
4   else {
5      if(...) 
6         B()  
7          else
8          B()  
9      }
}
We determine if an untaken branch can lead to a new call site that is can eventually lead to the target call site. If so, we force along the branch.
Given a target call site, we determine if an untaken branch can lead to a new call site that is eventually lead to the target call site. If so, we force along the branch.
Dynamic Analysis: Forced Execution 2

```plaintext
Foo () {
  ...  
  if (...) {
    A()
    else {
      if (...) {
        B()
      } else {
        B()
      }
    }
  }
}
```
Force Executing Event Handlers

- When encountering an NIB loading statement
  - Post the events of all the implicit handlers to the main event dispatch queue
    - MouseMove, ButtonClick
  - iOS will eventually call the corresponding handlers
Evaluation

- **Setup**
  - iOS 7
  - iPad 3 and iPad 4

- **2019 free applications from an official App Store**
  - 9 categories
  - Popular apps listed in iTunes Preview
  - Crawled in March 2015
Results

- **App size:** 1-80MB, median 3MB
- **Analysis time:** 257sec to 30hours (avg: 2439 sec)
- **Static analysis results**
  - 135 million call sites can be resolved (99%)
- **Dynamic results (the remaining 40,052 call sites)**
  - iRiS can resolve 35,427 of them (88%)
  - Note that these unresolved sites distribute in 1859 (92%) of the benchmarks
  - 18671 (47%) were resolved by the first natural run, 16756 resolved by forced execution

- 149 (7%) applications use private APIs
  - Identified a total number of 153 private APIs
## Private APIs: Access User Identification

<table>
<thead>
<tr>
<th>Framework</th>
<th>API Name</th>
<th>Functionality</th>
<th>#apps using</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 bundle ids of all running apps</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>SBFrontmostApplicationDisplayIdentifier</td>
<td>Obtain bundle id of the front most app</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>SBSCopyLocalizedApplicationNameForDisplayIdent</td>
<td>Get app name from its bundle 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: Access User Identification

<table>
<thead>
<tr>
<th>Framework</th>
<th>API Name</th>
<th>Functionality</th>
<th>#apps using</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>
<tr>
<td>IOKit</td>
<td>IORegistryEntryCreateCFProperties</td>
<td>Iterate through all properties to find information (e.g. Battery Id, IMEI)</td>
<td>2</td>
</tr>
<tr>
<td>IOKit</td>
<td>IORegistryGetRootEntry</td>
<td>Release IOService object</td>
<td>2</td>
</tr>
<tr>
<td>IOKit</td>
<td>IORegistryEntryGetChildIterator</td>
<td>Release IOService object</td>
<td>2</td>
</tr>
<tr>
<td>IOKit</td>
<td>IOIteratorNext</td>
<td>Release IOService object</td>
<td>2</td>
</tr>
<tr>
<td>IOKit</td>
<td>IORegistryEntryGetNameInPlane</td>
<td>Release IOService object</td>
<td>2</td>
</tr>
<tr>
<td>IOKit</td>
<td>IOObjectRelease</td>
<td>Release IOService object</td>
<td>2</td>
</tr>
</tbody>
</table>
Conclusion

- Forced execution is a new kind of program analysis different from traditional analysis
  - Practical for binaries, neither sound nor complete
- Forced execution is very effective at disclosing hidden/malicious behavior
Thank you!

Q & A