University of Amsterdam

Dana Geist & Marat Nigmatullin

Root/Jailbreak Detection Evasion Study on iOS and Android

Research Project 1

Motivation

- Compromised (rooted/jailbroken) devices are a major issue in the mobile security field.
- Security and business applications often attempt to identify rooted/jailbroken devices.
- Cloaking techniques are being developed as the detection counterpart.

Research questions

- **RQ1**: Which techniques are used for root/jailbreak detection and evasion on Android and iOS?
- **RQ2**: Are there any differences between the techniques used for each of the platforms? Are the controls they present effective?
- **RQ3**: What are the latest trends used for detection?
- **RQ4**: Could those latest trends be circumvented? If so, is it possible to create new evasion methods and implement them?

Related work

- Bulk of the research is focused on Android.
  - Detection methods are not effective against evasion techniques.
  - Focused on high level (Java) and native languages (C/C++).
- IOS
  - Lack of formal research that addresses iOS detection and evasion methods.
  - NESO Security Labs AppMinder developed a free prototype for jailbreak detection, based on ARM assembly code.

Detection and Evasion Methods

- Methodology
  - Study detection/evasion methods (RQ1, RQ2):
    - Primary literature
    - Existing tools and frameworks
    - Popular forums
  - Analyze collected information to detect latest trends (RQ3)

Detection and Evasion Methods

- Taxonomy of Android Root Detection Methods
  - Presence of packages, applications, files.
  - Build settings: test keys, build version.
  - File permissions.
  - Shell command execution (su, which su).
  - Runtime characteristics: mount /system partition.

Detection and Evasion Methods

- Taxonomy of iOS Jailbreak Detection Methods
  - Existence of files.
  - Directory permissions.
  - Process forking.
  - SSH loopback connections.
  - Privilege actions execution.
  - Calling dynamic library functions.
  - AppMinder Solution.

```
if ([[NSFileManager defaultManager]
fileExistsAtPath:@"/Applications/Cydia.app"])
    {
        return YES;
    }
    else
            if ([[NSFileManager defaultManager]
fileExistsAtPath:@"/Library/MobileSubstrate/MobileSubstrate.dylib"])
    {
        return YES;
    }
```

https://github.com/leecrossley/cordova-plugin-jailbreak-detection

Detection and Evasion Methods

- Root/Jailbreak evasion methods
  - Simple methods:
    - Hiding su binary (Android)
    - Runtime checks (Android)
    - Binary patching (Android and iOS)
  - Frameworks:
    - RootCloak (Android)
    - RootCloak Plus (Android)
    - xCon (iOS)

Detection and Evasion Methods

- Android vs. iOS: Method Comparison
  - Based on the same idea.
  - Detection/evasion methods implemented in different levels of abstraction:
    - High level: Java/Objective-C
    - Native level: C/C++
    - Low level: ARM assembly (No framework available)
  - Minor differences in implementation (e.g fork).

Detection and Evasion Methods

- Latest trends
  - Most applications implement detection controls in **high level** and **native** languages
  - NESO Security Labs created a jailbreak detection solution implemented in **ARM assembly** : AppMinder

AppMinder: What is it?

- Jailbreak detection tool for Apple iOS.
- Based on ARM assembly.
- Fork system call is evaluated for detection.
- Code consists of 5 functions.
- Application is terminated on jailbroken devices

```
#if !defined(DISABLE_APPMINDER) && !
(TARGET_IPHONE_SIMULATOR) && !(__arm64__)
__attribute__ ((always_inline)) static void
dFRdWsEfEaJi (unsigned int
*___lxTgdaUaxSYingsbeypmEtHgmILez, unsigned int
*___TukDsLwSvzYctQkYpXKiDfwnLvJJJ, unsigned int
*___aurUzzwAHntEjodevWkF)
{asm volatile ("sub r1, r1, r1;mov r0, r1;b
L975215;push {r0-r12};L975215:;mov r12, #32;mov r3,
r3;asr r12, #4;mov r3, r3;add r0, r0, #40;b
L975216;stmdb sp!, {r0-r12};L975216:;mov r4, pc;ldr
r4, [r4, #0];svc 0x80;ldr r3, %
[lxTgdaUaxSYingsbeypmEtHgmILez];str r4, [r3, #0];b
L975217;push {r0-r12};L975217:;sub r1, r1, r1;mov r0,
r0;mov r3, r1;mov r2, r2;add r3, r3, #1;mov r1, r1;cmp
r0, r3;b L975218;stmdb sp!, {r0-r12};L975218:;beq
L975219;mov r10, #79;mov pc, r10;L975219:;ldr r3, %
[TukDsLwSvzYctQkYpXKiDfwnLvJJJ];str r0, [r3, #0];ldr
r3, %[aurUzzwAHntEjodevWkF];str r12, [r3, #0];
...
```

Reference:http://appminder.nesolabs.de/

AppMinder

- Why is it difficult to bypass?
  - No traditional methods work on it.
  - Polymorphic.
  - Obsfuscation.
  - Self integrity checks.
  - Assembly code added ”inline”.

Experiments on iOS

- Methodology (RQ4)
  - Study AppMinder.
  - Understand its inner workings.
  - Create methods for evasion and implement them.

Experiments on iOS

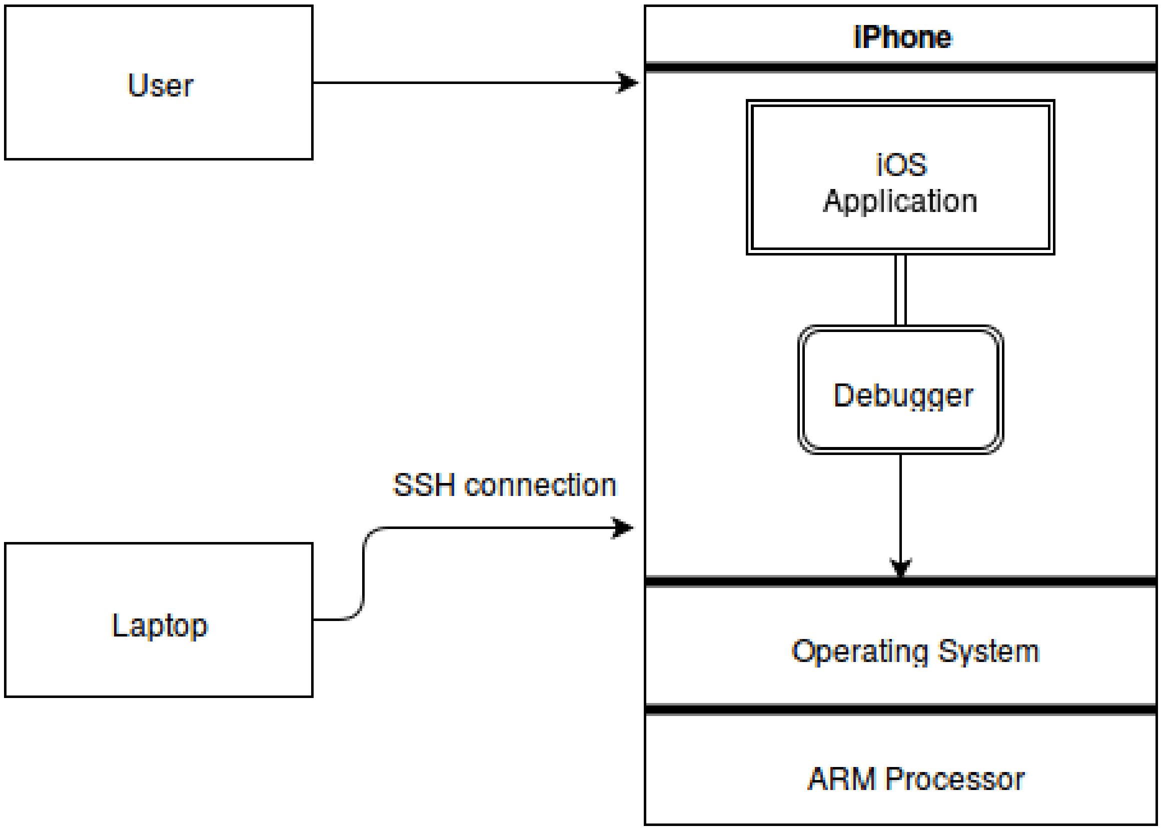
- Methodology (RQ4)
  - Create an iOS testing application with AppMinder checks.
  - Static/Dynamic analysis.
  - Identify patterns.
  - Design a strategy to bypass AppMinder's controls.
  - Implement solution.

Experiments on iOS: bypassing AppMinder

- Techniques explored:
  - Hooking tools such as Cycript.
  - Binary patching.
  - **Debbuging tools: GNU Debugger (a.k.a gdb).**

Experiments on iOS: bypassing AppMinder

- System architecture:



Experiments on iOS: bypassing AppMinder

- Code analysis: supervisor calls (SVC)
  - Fork: jailbreak detection
  - Ptrace: anti-debugging measures
  - Exit

UNIVERSITEIT VAN AMSTERDAM

Experiments on iOS: bypassing AppMinder

- Bypassing strategy: Fork
  - Normal device:r0=1
  - Jailbroken device: r0!=1

    (Child's PID)
  - Solution
    - Alter return value:

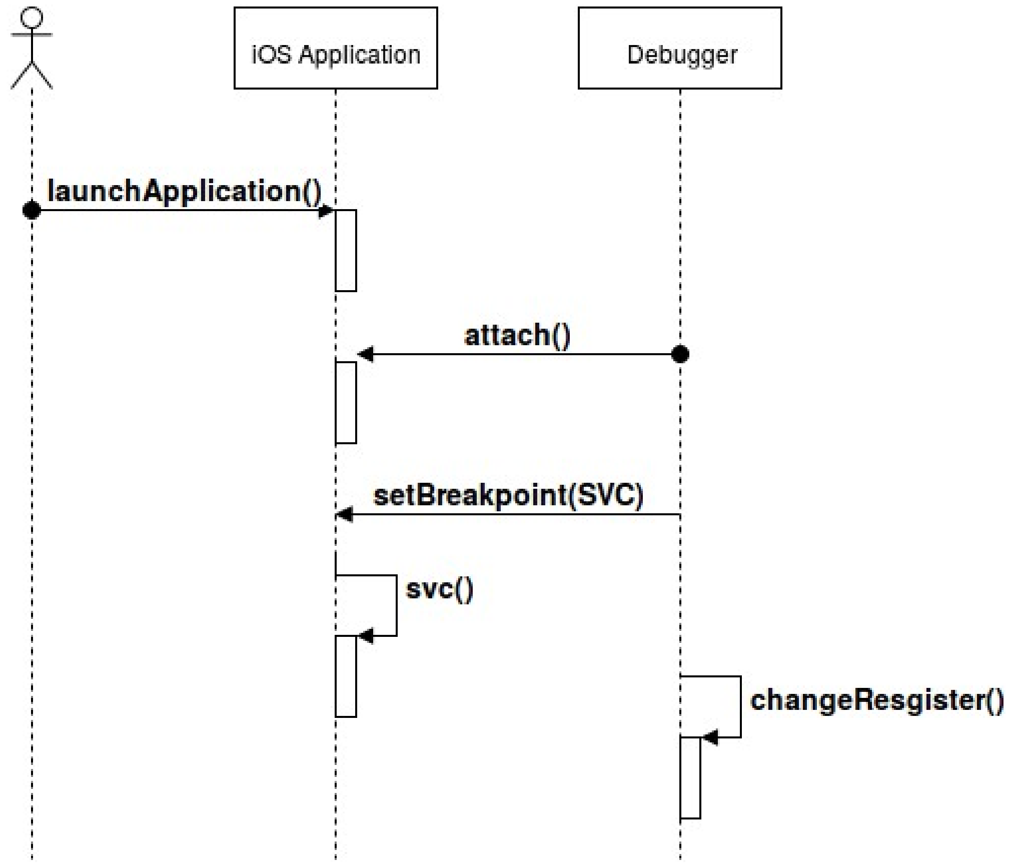
      set r0=1

Sample Code:

```
mov r1 , #2;
b L505572 ;
stmdb sp ! , { r0−r 1 2 } ;
L505572 : ;
mov r12 , r1 ;
svc 0x80;        ←Breakpoint
sub r1, r1, r1; ←Breakpoint
mov r3, r1;
add r3, r3, #1;
cmp r0, r3;
```

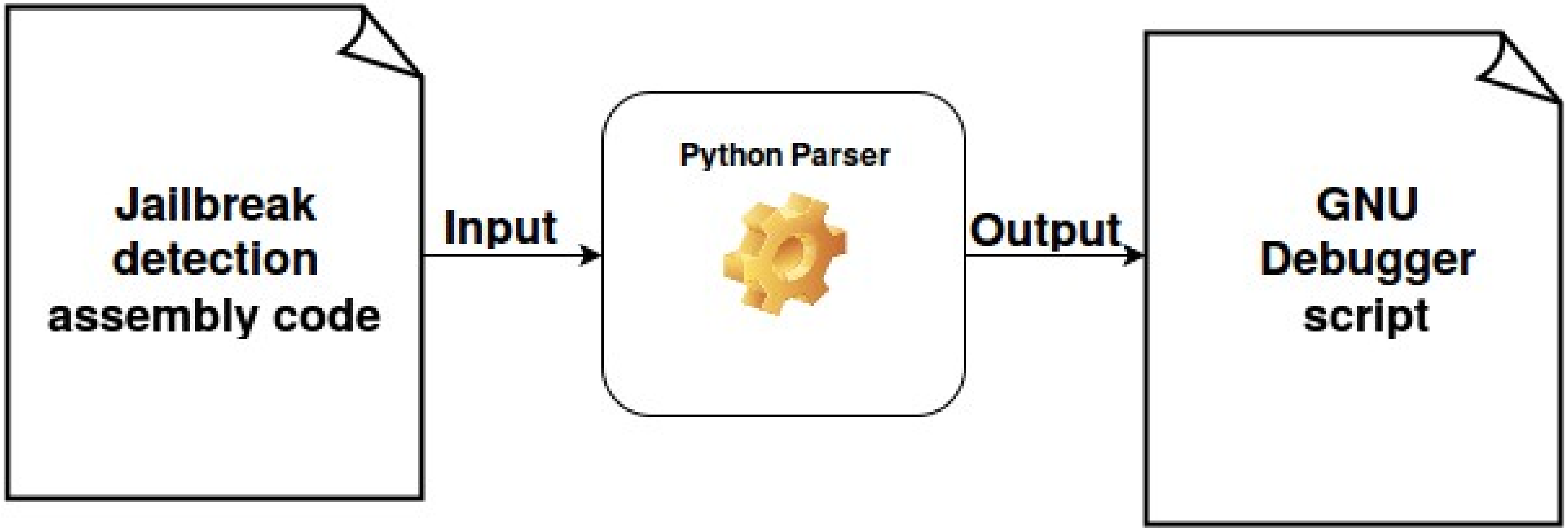
Experiments on iOS: bypassing AppMinder

- Component interaction:



Experiments on iOS: bypassing AppMinder

- Semi-automatic solution



Experiments on iOS: bypassing AppMinder

- Limitations:
  - We studied AppMinder’s variant B.
  - We worked with our own testing application.
  - Fifth function call exhibits different behavior.

Experiments on iOS: alternative jailbreak detection methods

- Cordova jailbreak detection plugin:
  - Implemented in Objective-C.
  - Detection methods:
    - Check for existing directories, files or packages.
    - Execute privileged actions like writing outside of the sandbox.

Experiments on iOS: alternative jailbreak detection methods

- Cordova bypassing:
  - Focus on if statements.
  - Target assembly compares.
  - Change register values.

Objective-C	ARM Assembly
if ([[NSFileManager defaultManager] fileExistsAtPath: @"/Applications/Cydia.app"])	Check for file existence
{return YES;}	cmp r1, #0
else if ...(next check)	

Results & Analysis

- AppMinder controls were evaded.
- Bypassing mechanisms were successfully implemented.
- Assembly level techniques can be used to evade methods at different abstraction levels.
- Attaching a debugger affects performance.

Conclusions

- Android and iOS use similar detection and evasion methods.
- Detection trends are moving controls to lower level languages. AppMinder is an example of that.
- Even low level techniques can be bypassed.
- With enough time and resources an attacker will be able to evade all detection controls.

Future Work

- Address limitations of our current study:
  - Implement an efficient fully automated solution to evade AppMinder's controls.
  - Study evasion of different detection mechanisms for both Android and iOS.

DEMO

Any questions?