#15

Dynamic Code Patch with Frida

영화 <마션> 에서 주인공이 탐사 장비의 프로그램 코드를 직접 패치(Patch)하고 있는 장면. 코드를 패치하면 원하는 대로 프로그램을 바꿔 동작시킬 수 있다.

모든 프로그램은 개발자가 의도한 대로 동작하게 되어있다. (당연한 이야기다.) 보안의 관점에서 프로그램을 바라볼 때, 개발자의 의도를 비틀어 다른 동작을 수행하도록 만드는 공격 방법이 있다. 개발자의 의도와 상관없이 공격자의 의도대로 프로그램이 동작한다면 그 피해는 가늠하기 어렵다. 따라서 보안 점검 시 고의로 코드가 변경되지는 않는지 확인해봐야 한다.

프로그램을 변경하는 방법은 두 가지가 있다. 프로그램이 실행되지 않은 상태에서 프로그램을 변경하는 정적 코드 패치(Static Code Pathc)와 프로그램을 실행시키는 과정에서 프로그램의 코드를 변경하는 동적 코드 패치(Dynamic Code Patch)가 있다.

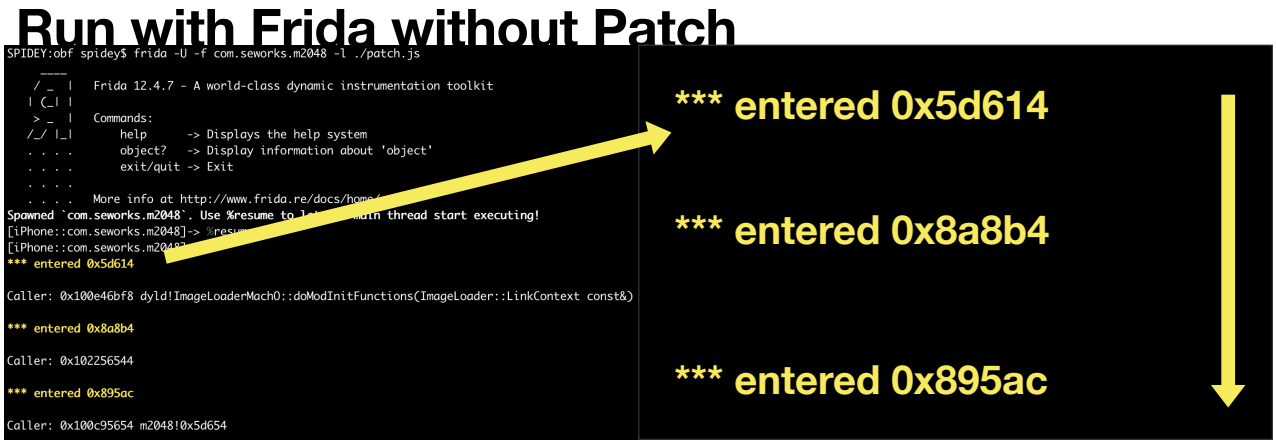
Android나 iOS는 프로그램이 만들어지기 전에 코드 서명(Code Signing)을 통해 정적(Static)으로 프로그램 코드가 변경되는 것을 막는다. 물론, 공격 과정에서 코드 서명을 임의로 다시 할 수 있다. 하지만 이번엔 동적 코드 패치에 대한 내용을 다루고자 한다. 프로그램은 전혀 건들지 않고, 실행되는 과정에서 프로그램을 어떻게 바꿀 수 있는지 알아보자.

Analyze Target Application - iOS

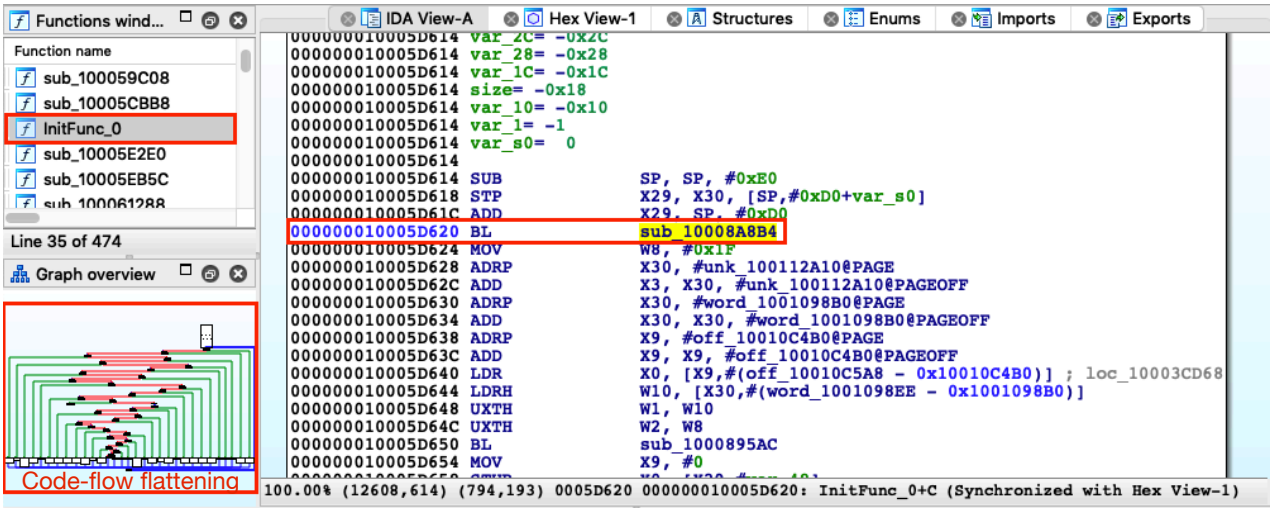
대상으로 정한 앱은 공개된 코드로 만들어진 m2048이라는 게임 앱이고, 해킹 방지를 위한 솔루션이 적용된 앱이다.

솔루션이 적용된만큼 탈옥폰에서는 앱이 정상적으로 시작되지 않는다. 하지만 솔루션이 탈옥 여부를 확인하기 직전까지는 앱이 동작하기 때문에 분석의 여지는 충분히 있다.

프로그램은 Init_Func(0x5d614) 함수부터 시작한다.



프로그램의 시작 부분을 더 자세히 확인하기 위해서 IDA를 사용해보자.



왼쪽 밑을 보면 솔루션이 적용된 탓으로 코드의 흐름을 따라가기 어렵게 되어 있다.

Control-flow flattening이라 부른다.,
코드의 흐름이 평평하게 나열된 많은 코드 블록으로 흩어진다.

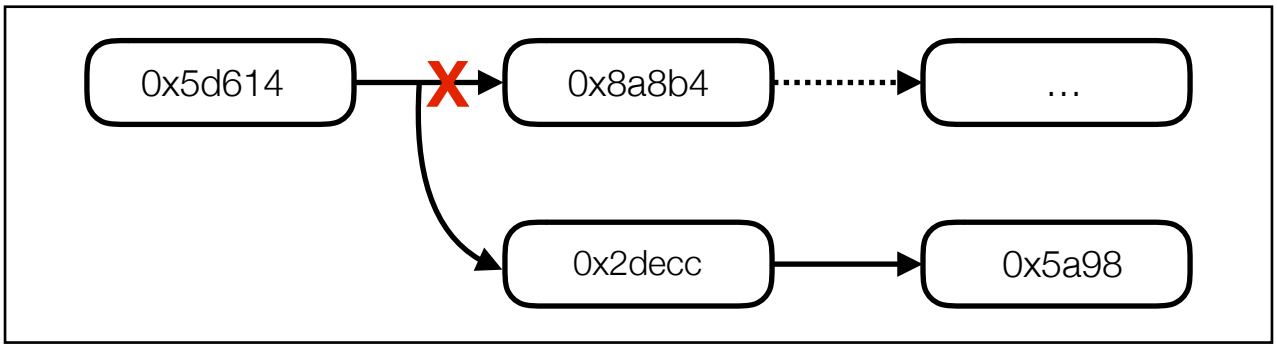
그리고 frida에서 확인한 것과 같이 Init_Func(0x5d614)이 시작된 이후 첫번째 함수 sub_10008A8B4를 호출한다.

Aiming

프로그램이 실행되는 순간 코드를 패치하여 프로그램의 흐름을 바꿔보자.

별 다른 수정을 하지 않는다면 솔루션이 적용된 프로그램이기 때문에 탈옥 탐지 관련 코드가 실행되고 비정상 종료될 것이다. 따라서 이번 코드 패치의 목적은 탈옥 탐지 관련 코드를 건너 뛰는 것이 될 것이다.

방법은 매우 다양하겠지만 간단한 패치만으로 목적을 달성해보자. branch 명령만으로 프로그램의 실행 흐름을 변경할 것이다. (ARM64 opcode를 알아야 한다. 참고: opcode converter online)



최종적으로 실행하고자 하는 목표 코드는 0x100005a98 주소에 있는 코드다. 목적지를 이곳으로 정한 이유는 IDA로 분석했을 때 앱의 Main 함수로 추측되는 부분이었기 때문이다.

Init_Func의 코드를 패치하여 원하는 코드로 실행 분기 변경.

```
0000000100005A98
0000000100005A98 loc_100005A98
0000000100005A98 MOV             X8, SP
0000000100005A9C SUBS            X8, X8, #0x10
0000000100005AA0 MOV             SP, X8
0000000100005AA4 MOV             X9, SP
                                 ...
0000000100005C98 MOV             X2, X8
0000000100005C9C SUB             X8, X29, #-var_60
0000000100005CA0 LDUR            X3, [X8,#-0x100]
0000000100005CA4 BL              _UIApplicationMain
0000000100005CA8 SUB             X8, X29, #-var_28
0000000100005CAC LDUR            X8, [X8,#-0x100]
0000000100005CB0 STR             W0, [X8]
```

<0x5a98 code block>

그런데, 굳이 중간에 0x2decc 주소를 한 번 거친 후 0x5a98로 분기하는 이유는 ARM64 명령어가 한 번에 이동할 수 있는 거리에 제한이 있기 때문이다. **즉, 0x5d614에서 0x5a98까지 거리가 너무 멀다!!**
(더 좋은 opcode가 있으면 알려주세요!)

Code Segment

코드를 패치하기 전에 프로그램의 코드 영역(Code Segment)의 특징에 대해 간단히 짚고 넘어갈 필요가 있다. 코드 영역은 메모리에 올라갔을 때 쓰기 권한이 없다. 읽기(Read)와 실행(eXecute) 권한만 있다. 코드는 프로그램이 실행되는 동안 변경될 일이 없기 때문에 코드를 읽고 실행하는 권한만 있으면 된다.

따라서 우리는 코드 패치를 위해 코드 영역의 속성을 변경해줘야 한다. 전통적으로 많이 쓰이는 Memory protect 기술을 사용해서 속성을 변경할 것이다. (Frida에서 지원한다.)

Run Target Application with Frida

공격에 쓰인 핵심 코드를 살펴보자.

```
1   const membase = Module.findBaseAddress('m2048');
2   const initFunc = memAddress(membase, '0x0', '0x5d614');
3   Memory.protect(initFunc, 4096, 'rw-');
4   Process.enumerateRanges('rw-', {
5       onMatch: function(range) {
6           console.log('base : ' + range.base + ' / size : ' + range.size);
7           var matches = Memory.scanSync(range.base, range.size, 'FF 83 03 D1');
8           if (matches.length == 0) {
9               return;
10          }
11          var match = matches[0];
12          Memory.writeByteArray(match.address, [ 0x2E, 0x42, 0xFF, 0x17 ]);
13          Memory.protect(initFunc, 4096, 'r--');
14      },
15      onComplete: function() { }
16  });
```

[Line 2] : Init_Func 함수 포인터를 가져온다.

[Line 3] : Memory.protect 함수를 이용해서 Init_Func 함수에 쓰기(Write) 속성을 추가한다.

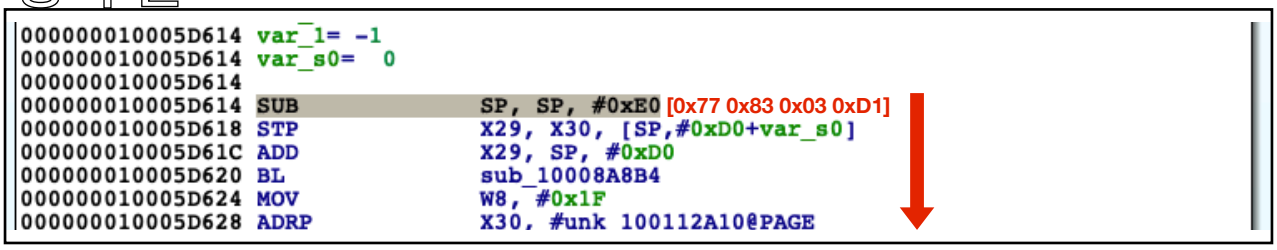
[Line 4] : 프로세스의 메모리 중 읽기(Read)/쓰기(Write) 권한이 있는 부분에

[Line 7] : [0x77, 0x83, 0x03, 0xD1] 이 있다면

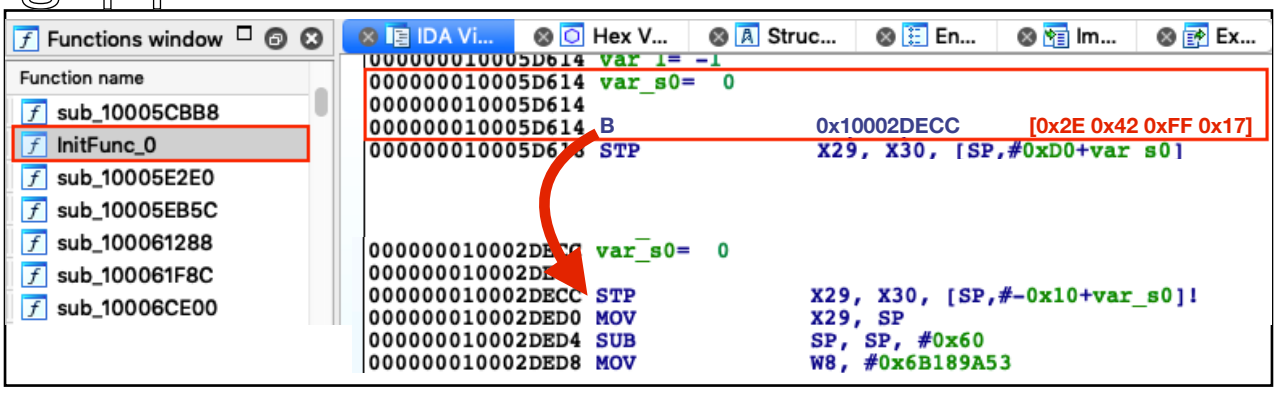
[Line 12] : [0x2E, 0x42, 0xFF, 0x17]로 바꾼다.

그림으로 설명하자면, 아래와 같다.

공격 전



공격 후



Init_Func (0x5d614) 함수의 시작 코드 *SUB SP, SP #0xE0 [0x77, 0x83, 0x03, 0xD1]*를 우리가 원하는 *B 0x10002DECC [0x2E 0x42 0xFF 0x17]* 로 변경했다. B(Branch)는 프로그램의 실행을 주소값 (0x10002DECC)으로 이동시키라는 의미다.

실행 결과를 보면 다음과 같다.

```
Spawned `com.seworks.m2048`. Use %resume to let the main thread start executing!
[iPhone::com.seworks.m2048]-> %resume
[iPhone::com.seworks.m2048]->
*** entered 0x2decc

Caller: 0x10242abf8 dyld!ImageLoaderMachO::doModInitFunctions(ImageLoader::LinkContext const&)
```

앞에서 Caller 인 dyld!ImageLoaderMachO::doModInitFunctions는 0x5d614를 호출 했지만, 코드 패치를 통해 우리가 원하는 대로 0x2decc 함수를 호출했다.

What about Android so file

위와 같은 방식으로 안드로이드의 so(shared object) 파일을 후킹하여 코드 수정을 할 수 있다. Give it a go!

iOS, Android 모두 코드 서명 과정을 거쳐 코드 패치 공격으로부터 프로그램을 보호하고 있다. 하지만 Frida를 이용한 동적 코드 패치 공격을 활용하면 코드 서명을 우회하여 우리가 원하는대로 프로그램을 수정할 수 있는 장점이 있다. 특히 탈옥/루팅 방지 보안 솔루션이 적용된 경우 많은 도움이 된다.