
iOS Application (In)Security

(번역 문서)

2012-06-15

해당 번역문서는 연구 목적으로 진행된 프로젝트입니다.
상업적으로 사용하여 법적인 문제가 생기는 경우는 사용자 자신에게
책임이 있음을 경고합니다.

모바일 위협 분석 nam_daehyeon PM 님

- 보안프로젝트 (www.boanproject.com) -

목 차

1. 소개	1
2. 배경	2
2.1. iOS 보안 기능	2
2.2. iOS Overview	4
2.3. 먼저 읽어야 할 것	5
3. 블랙 박스(Black Box) 방식 진단	6
3.1. AppStore 바이너리 암호 해체	6
3.2. Locating the Position Independent Executable	9
3.3. Identifying the use of Stack Smashing Protection	11
3.4. Identifying the use of Automatic Reference Counting	12
3.5. inspecting the Binary	12
3.6. Manipulating the Runtime	13
3.7. Defending the Binary	18
3.8. 결론	20
4. 안전하지 않은 API 사용 진단	21
4.1. Evaluating Transport Security	21
4.2. Abusing Protocol Handlers	27
4.3. Locating InSecure Data Storage	31
4.4. Attacking the iOS Keychain	37
4.5. Conduction Cross-Site Scripting (XSS) through UIWebViews	40
4.6. Attacking XML Processors	42
4.7. SQL Injection	44
4.8. Filesystem Interacion	46
4.9. GEO-Location	49
4.10. Logging	50
4.11. Backgrounding	50
5. 대응 방안	52
5.1. Format String	52
5.2. Object Use-after-Free	55
6. 결론	57
7. iOS App Compliance Checklist	58

8. About MDSec.....	60
9. Acknowledgements	61
10. References.....	62

www.boanproject.com

그림 목차

그림 1. cripid 값 확인.....	6
그림 2. 아키텍처 컴파일 정보 확인.....	7
그림 3. doModInitFuctions 브레이크 포인트 설정.....	8
그림 4. dump memory 사용한 검색.....	8
그림 5. 암호화된 세그먼트와 대체 - 원본에 저장.....	9
그림 6. crpyid 값 수정.....	9
그림 7. 간단한 예제 작성.....	9
그림 8. main 함수 고정주소 값.....	10
그림 9. main 함수 동적 주소로 실행.....	10
그림 10. PIE 값 확인 가능.....	10
그림 11. -fstack-protector-all Flag 사용.....	11
그림 12. otool 을 이용하여 dump.....	11
그림 13. ARC 와 관련된 심벌 확인.....	12
그림 14. class-dump-z 사용.....	13
그림 15. SpringBoard 프로세스에 코드 삽입 등 작업 가능.....	14
그림 16. jailbreak 관련 파일시스템 체크.....	14
그림 17. MobileSubstrate tweak 사용.....	15
그림 18. DynamicLibraries 폴더에 저장.....	15
그림 19. 매번 실행 될 때마다 로드.....	16
그림 20. 특정 어플리케이션 실행 될 때 로드.....	16
그림 21. class-dump 을 이용하여 관련된 메서드 확인.....	16
그림 22. 경고 메시지 확인.....	17
그림 23. Jailbreaking 우회.....	17
그림 24. isJailBrokenDevice Hook.....	18
그림 25. class_getMethodImplementation 메서드를 제공.....	19
그림 26. jailbreak 감지 기능 정상 실행.....	19
그림 27. isJailBroken 메소드 변경.....	19
그림 28. keyword inline 사용.....	20
그림 29. HTTPS 접속 구현.....	22
그림 30. 4.3 버전 - TLS 1.0 버전 사용.....	23
그림 31. 29 개 암호군 중에 사용.....	23
그림 32. SDK 5.0 or 5.1 - TLS 1.2 세션 사용.....	24
그림 33. 37 개 암호군 중 사용.....	24

그림 34. allowsAnyHTTSPCertificateForHost 메서드를 사용	25
그림 35. didReceiveAuthenticationChallenge 메서드 구현	26
그림 36. onSocket delegate 메서드	26
그림 37. custom URL 스키마를 등록한 VulnerableiPhoneApp 프로젝트	27
그림 38. handleOpenURL 이나 openURL 을 이용	28
그림 39. handleOpenURL 메서드의 구현	28
그림 40. setHomeURL 객체를 저장	29
그림 41. iframe 을 사용해서 어플리케이션의 설정파일을 재설정	29
그림 42. 요청된 URL 을 확인	29
그림 43. 호출된 URL 을 검사	30
그림 44. UIView 확인	30
그림 45. 페이스북 어플리케이션에서 확인한 protocol	31
그림 46. com.kik.chat.plist 파일	33
그림 47. sqlite database 에 저장	33
그림 48. 첨부파일 그림	34
그림 49. NSFileProtectionComplete 속성을 설정	36
그림 50. 잠긴 모바일기기에 파일 접근 시도	36
그림 51. provisioning profile 만이 app 의 keychain 에 접근	38
그림 52. 모바일기기가 잠겨있지 않을 때에만 아이템에 접근	39
그림 53. SQLite database keychain	40
그림 54. UIWebView DOM	41
그림 55. XSS 예제	42
그림 56. 취약한 NSXMLParser 의 구현	43
그림 57. setShouldResolveExternalEntities 파서 옵션의 설정	43
그림 58. 웹 서버로 강제로 HTTP 요청	44
그림 59. 공격 벡터의 구현	44
그림 60. SQLite statement	45
그림 61. 상태 메시지 고려	45
그림 62. SQL 쿼리 실행 예제	45
그림 63. sqllite3 에서 SQL Injection 해결 방안	46
그림 64. 파일 매니저 구현	47
그림 65. 파일 매니저 구현	48
그림 66. 파일 시스템 접근 취약점 검사	48
그림 67. string 객체를 일찍 단축(terminate)	48
그림 68. "Hidden"속성을 사용	51

그림 69. applicationDidBecomeActive delegate 를 사용.....	51
그림 70. 취약한 예	53
그림 71. NSLogv.....	53
그림 72. format string 의 취약점이 적용 여부 확인.....	53
그림 73. console log 에 dump	54
그림 74. 예제	54
그림 75. crash 를 발생 여부 확인.....	55
그림 76. use-after-free 취약점 예제	55
그림 77. heap 이 사용자 제어 데이터와 함께 뿌려진 데이터.....	56
그림 78. 엑세스 위반 에러가 발생	56

표 목차

표 1. SDK 버전별 전송 체계	21
표 2. 어플리케이션 디렉터리 구조	32
표 3. 보호 레벨	35
표 4. 확장 속성	35
표 5. KeyChain 보호 레벨.....	37
표 6. 파일시스템 상호작용 인스턴스 메서드를 포함.....	47
표 7. 평가 가이드.....	59

1. 소개

작년에, MDSec 의 컨설턴트들은 iOS 어플리케이션들과 그들이 지원하고 있는 구조 가운데서 특히 소매기업금융 분야에 대해서 급증하고 있는 많은 보안 평가들을 수행하였고, 그 중에서 특히 데이터 보안분야가 단연 최고였다. 스마트 폰들은 소비자시장 뿐만 아니라 기업시장에도 일반화 되었다.

스마트폰들은 전통적인 전화기의 기능들과 컴퓨터의 기능들이 합쳐졌다. 최신 스마트 폰의 향상된 처리능력과 메모리는 모바일 어플리케이션 개발의 급격한 증가를 이끌고 개발자는 플랫폼의 다양한 기능을 활용할 수 있다는 기대를 걸고 있다.

어플리케이션개발은 너무나 인기 있어서 "그것에 대한 App 은 있다."라는 애플의 트레이드마크 슬로건은 실제로 현실화 되고 있다.

2011 년에 목격했던 트렌드는 계속 증가하고 있고, iOS 나 Android 앱들이 선두주자가 되고 있는 모바일 어플리케이션들 보안평가(security assessments)에 대한 수요는 계속 증가하고 있다.

NetApplications 에서 시장조사한 결과를 보면 iOS 기기가 국제 모바일 시장의 52%를 차지하는 것을 보여준다. MDSec 의 iOS 어플리케이션 보안의 실무 교육과정 백서는 iOS 어플리케이션들 에 악영향을 끼치는 이슈들에 대한 카타고리들에 대해서 보안관계자들 뿐만 아니라 최상의 보안을 유지하고 싶은 개발자들에게 하나의 기준점을 제시 하는 방향으로 초점을 맞추고 있다.

2. 배경

2.1. iOS 보안 기능

Code Signing 기능

Code signing 은 플랫폼의 실시간 보안기능으로 모바일 기기에서 매번 프로그램이 실행 될 때마다 검증된 어플리케이션의 서명(signature)에 의해서 허가되지 않은 어플리케이션들의 실행 금지를 시도한다.

게다가 어플리케이션들은 오직 유효하고, 신뢰할만한 서명(signature)에 의해 sign 되어진 코드만을 실행한다.

어플리케이션은 먼저 신뢰할만한 인증서에 의해서 sign 되어야 기기에서 실행 할 수 있다.

개발자들은 Apple 에 의해서 허가된 provisioning profile 을 통해서 신뢰받은 인증서를 기기에 설치할 수 있다.

provisioning profile 은 내장된 개발자의 인증서 그리고 개발자가 어플리케이션들에 허용 한 것들에 권한의 모음들(set of entitlements)포함하고 있다

완성된 어플리케이션들, 모든 코드들은 애플에 의해 허가 되어 지는데 이 과정은 AppStore 등록과정을 통해서 이루어진다.

등록과정은 개발자들에 의해 사용되어진 기능, 예를 들면 app 들의 불법 API 사용 또는 설치를 위해서 실행코드를 다운로드 하는 등의 기능 그리고 API 들을 준수하기 위해서 app 들에 대한 Apple 의 어느 정도의 통제를 허용한다. [3]

Exploit 완화 기능

주소 영역 레이아웃 무작위화(ASLR)은 프로세스 주소 영역 안에 매핑 되어 있는 코드와 데이터를 무작위화함으로써 exploitation 취약점의 복잡도를 증가시키는 보안기능이다

ASLR 기능은 iOS v4.3b 버전부터 최초로 소개되어졌다. 그리고 시작 이후로 각각의 공개버전마다 점점 개선 되어졌다.

ASLR 구현에서의 주요한 취약점은 dyld(다이나믹 링커) 재할당의 부족이고 이것은 iOS 5.0 버전에서 알려졌다.

어플리케이션들은 두 가지 다른 특징의 ASLR 을 적용 할 수 있다. 위치 독립적인 실행(Position Independent Execution(PIE))이 지원되는 컴파일을 했는지 여부에 따라 부분적 ASLR 또는 전체 ASLR(full ASLR)를 사용 할 수 있다.

iOS Application (In)Security 번역

전체 ASLR 시나리오에서, 모든 어플리케이션 메모리 영역들은 무작위화 되고 그리고 iOS 는 PIE 가능한 바이너리가 임의의 주소를 매번 어플리케이션이 실행 할 때 마다 적재 할 것이다.

부분적 ASLR 로 컴파일된 어플리케이션은 고정주소로 기본바이너리를 적재 할 것이다 그리고 다이내믹링커(dynamic linker(dyld))에 대해 정적 위치를 사용한다.

iOS 의 ASLR 의 심층평가는 Sefan Esser 에 의해서 연구되었고 좀 더 자세한 내용을 이해하고 싶으면 다음의 책을 읽기를 권한다.[5]

ASLR 은 지식의 부족으로 인해서 exploitation 을 어렵게 하기 위해서 ASLR 이 디자인되었고, 공격자는 대상이 되는 것에 대한 메모리 안의 프로세스 레이아웃을 가질 것이고 또 주소를 갖게 될 것이다. 하지만 그것들을 효과적으로 약화 시킬 수 있는 많은 방법들이 존재한다.

가장 공통적인 부분들의 기술은 memory revelation 이다.

이곳은 별도의 취약점으로 공격자가 사전에 임의의 실행코드를 실행 시킬 수 있는 취약점을 이용해서 보여(leak)주거나 메모리 레이아웃을 확인하는데 사용된다.

다국어 취약점들의 추가적인 exploitation 완화의 시도로 iOS 는 "W^X" 비 실행 메모리 정책("W^X" non-executable memory policy)으로 구현된 ASLR 과 결합했다. "W^X" 비 실행 메모리 정책이란 메모리 페이지들은 쓰기와 실행을 동시에 표시 할 수 없다는 것을 의미한다.

이와 같은 정책의 일환으로, 쓰기 가능하도록 표시된 실행 가능한 메모리페이지들은 나중에 실행 가능하다는 표시를 할 수 없다.

이와 유사한 많은 방식의 데이터실행보호(DEP) 기능들은 Microsoft Windows, Linux, 그리고 Mac OS X 데스크탑 OS 에서 구현되어졌다

반면에 비 실행메모리는 단독 Return Oriented Programming(ROP)기반의 페이로드를 사용함으로써 하찮게(trivially) 우회 할 수 있고, ASLR 과 의무적 Code Signing 과 결합 시에 exploitation 의 복잡도는 급격하게 증가 되어졌다.

iOS 어플리케이션들은 컴파일 시에 stack smashing protection 을 통해서 추가적인 exploit 완화를 추가했다고 볼 수 있다. Stack canaries 특히 지역 변수 앞에 알려진 값을 임의로 위치함으로써 버퍼 오버 플로우 (buffer overflows)에 대비한 몇 가지 보호기술들을 소개한다.

stack canary 는 함수의 리턴 시에 체크되는데, 만약에 오버 플로우가 발생하고 canary 가 손상되면, 어플리케이션은 오버 플로우를 찾아내고 보호 할 수 있게 된다.

SandBoxing

iOS 에서 구동되는 모든 third party 어플리케이션들은 sanbox 내에서 작동된다. sandbox 는 개개의 어플리케이션들 뿐만 아니라 운영시스템으로부터 독립적인 환경이다.

어플리케이션이 모두 "모바일" 운영시스템 사용자로서 실행되는 동안 그것들은 파일시스템에서의 고유한 디렉토리 내에 위치하고, 분리되며 XNU Sandbox kernel extension 에 의해서 관리 된다.

SandBox 안에서 수행 할 수 있는 일들은 seatbelt 프로파일을 준수하고 Third party 어플리케이션들은 일반적으로 어플리케이션의 홈디렉토리에 대한 제한된 파일접근, 미디어파일 읽기, 주소록 읽기 또는 쓰기, 뿐만 아니라 launchd 의 네트워크 소켓의 예외로 아웃바운드 네트워크 무제한적 접근이 가능한 "컨테이너"라는 프로파일이 할당된다.

자세한 것은 "The Apple SandBox"참고 [6]

암호(Encryption) 기능

기본적으로, iOS 파일시스템의 모든 데이터들은 flash 에 저장되어 저있는 File SystemKey 를 이용한 Block-based encryption (AES)에 의해 암호화되어 있다.

파일시스템의 암호화는 다음과 같다.

디바이스가 켜지면 하드웨어 기반의 crypto accelerator 는 filesystem 을 잠금 해제한다. 게다가 하드웨어 암호화는 개인적인 파일과 키체인(keychain) 아이템 들을 디바이스 패스코드로부터 생성된 키를 사용하는 Data Protection(DP) API 을 이용해서 암호화 할 수 있다.

결과적으로, 디바이스가 잠겨있을때, DP API 를 이용해서 암호화한 항목들은 메모리에서 접근 할 수가 없다.

중요한 데이터를 암호화 하고 싶은 Third party 어플리케이션들은 Data Protection API 를 사용 함으로서 암호화 할 수가 있다. 하지만 고려해야 할 사항은 디바이스가 잠겨 지는것 때문에 이용 불가능한 경우가 생기기 때문에 백그라운드 프로세스에게 어떻게 행동 해야 하는지 열려 주어 져야 한다.

2.2. iOS Overview

Third party iOS 어플리케이션들은 Cocoa Touch API 를/ 디바이스와 상호작용하기 위해서 Cocoa Touch API 를 사용한다. 이 프레임웍은 OS 로부터의 추상화 수단을 제공하는데 C 의 superset 으로 Object-c 로 작성되었다.

iOS 용 어플리케이션들은 무료로 이용 가능한 OS X 용 IDE 인 XCode 를 사용해서 개발 할 수 있다. XCode 는 컴파일과, 어플리케이션의 모바일 환경에서의 작동여부를 확인 할 수 있는 시뮬레이터를 제공해준다. 하지만 그것은 시뮬레이션이라기 보다는 에뮬레이션이라는 사실을 알아야 한다.

비탈옥(non-jailbraken) 디바이스에서 어플리케이션을 실행하기 위해서는, 사전등록 절차를 기반한 iOS 개발자 프로그램에 가입되어 있어야 하고 개발자용 인증서가 필요하다.

2.3. 먼저 읽어야 할 것

우리가 아는 바, 우리가 주목 해야 할 iOS 어플리케이션 보안 평가에 관한 발표가 있다. iOS 어플리케이션 평가를 하는 사람 이라면 다음의 두 개의 발표 모두 읽기를 추천한다.

"iPhone 과 iPad 의 어플리케이션 검사" Ilja van Sprundel[7]

"iOS 보안개발" David Thiel[8]

3. 블랙 박스(Black Box) 방식 진단

조직에 의해서 만들어진 일반적인 가정(assumption)은 어플리케이션의 내부작업은 어떻게든(in some way) 소스코드에 접속할 수 없는 공격자로부터의 보호이다.

실제로, 그것들은 key method 가 포함하고 있는 위치를 찾거나, 실행 시에 후킹 하거나, 변수를 바꾸거나 실행순서를 변경 하는 등 암호 해제 된 어플리케이션에 접근 하려 하는 공격자에게 비교적 직접적인 과정이다

이것들은 일반적으로 다음의 절차들을 따른다.

3.1. AppStore 바이너리 암호 해체

AppStore 에 올라와있는 App 들은 Apple 사의 바이너리 암호화 방식으로 보호된다. 이러한 app 들은 kernel mach loader 에 의해서 실시간으로 암호해제 될 것이다.

암호 해제 된 파일들의 복구 같은 작업들은 비교적 직접적 절차를 거친다. 이러한 암호화의 삭제는 공격자로 하여금 리버스 엔지니어링을 통해서 내부 클래스의 구조와 적합한 바이너리 상태를 얻기 위해 어떻게 바이너리가 행동하게 하는지 좋은 정보를 제공해준다.

AppStore 의 암호화의 삭제는 loader 가 app 을 암호해제 한 후에 debugger 를 이용해서 암호해제 된 이미지를 추출함으로써 (by letting) 달성 할 수 있다.

이 과정은 Cydia 와 AppCrack 두 개의 어플리케이션을 이용해서 자동화 할 수 있지만 이러한 과정을 역시 GDB 를 이용해서도 할 수 있다.

암호화 된 바이너리들은 LC_ENCRYPTION_INFO 라는 필드 값의 "cryptid" 안의 값을 이용해서 알아낼 수 있다. 명령어 사용은 다음과 같다.

```
mdsec-iPhone:/var/mobile/Applications/E938B6D0-9ADE-4CD6-83B8-712D0549426D/99Bottles.app root# otool -l 99Bottles | grep -A 4 LC_ENCRYPTION_INFO
    cmd LC_ENCRYPTION_INFO
    cmdsize 20
    cryptoff 4096
    cryptsize 12288
    cryptid 1
```

그림 1. cryptid 값 확인

어떤 경우에는 app 들은 여러 개의 아키텍처로 컴파일 되는데, 대표적인 것이 FAT 바이너리다. 아래의 그림처럼 otools 를 이용해서 App 들이 어떠한 아키텍처로 컴파일 되어졌는지 알 수 있다.

iOS Application (In)Security 번역

```
mdsec-iPhone:/var/mobile/Applications/68E3B644-9203-4B8F-A707-
A52E23B793B6/Kik.app root# otool -f Kik

Fat headers
fat_magic 0xcafebabe
nfat_arch 2
architecture 0
  cputype 12
  cpusubtype 6
  capabilities 0x0
  offset 4096
  size 865152
  align 2^12 (4096)
architecture 1
  cputype 12
  cpusubtype 9
  capabilities 0x0
  offset 872448
  size 867488
  align 2^12 (4096)
```

그림 2. 아키텍처 컴파일 정보 확인

위의 예제에서 cputype 12, cpusubtype 6 은 ARM v6 에 해당하고, cputype12, cpusubtype 9 는 ARM v7 에 해당한다. lipo 를 사용해서 원하는 아키텍처에 따라 바이너리가 "경량화" 될 수 있다.

App 의 암호 해제 된 세그먼트를 검색하기 위해서는 우선 해야 할 일은 loader 를 실행시켜야 한다. 이것은 모든 객체가 전부 로딩 된 후에 호출되는 "doModInitFunction"에 브레이크 포인트(break point)를 설정해 줌으로써 얻을 수 있다. (can be achieved)

iOS Application (In)Security 번역

```
mdsec-iPhone:/var/mobile/Applications/E938B6D0-9ADE-4CD6-83B8-712D0549426D/99Bottles.app root# gdb --quiet -e ./99Bottles

Reading symbols for shared libraries . done

(gdb) set sharedlibrary load-rules ".*" ".*" none
(gdb) set inferior-auto-start-dyld off
(gdb) set sharedlibrary preload-libraries off
(gdb) rb doModInitFunctions
Breakpoint 1 at 0x2fe0ce36
<function, no debug info>
__dyld__ZN16ImageLoaderMachO18doModInitFunctionsERKN11ImageLoader11LinkContextE;
(gdb) r

Starting program: /private/var/mobile/Applications/E938B6D0-9ADE-4CD6-83B8-712D0549426D/99Bottles.app/99Bottles

Breakpoint 1, 0x2fe0ce36 in
__dyld__ZN16ImageLoaderMachO18doModInitFunctionsERKN11ImageLoader11LinkContextE
()
(gdb)
```

그림 3. doModInitFuctions 브레이크 포인트 설정

이 단계에서 로더(loader)는 App 을 암호해제하고, 우리는 메모리에 있는 텍스트 세그먼트들(text segments)을 명확하게 Dump 해 낼 수 있다.

암호화된 세그먼트의 위치는 헤더에 대응되는 offset 값을 제공해주는 Load command 인 LC_ENCRYPTION_INFO 라는 cryptoff 라는 값에 의해서 명시 된다.

따라서 암호화된 세그먼트의 시작은 offset 0x2000(0x1000 의 cryptoff(4096) + 0x1000 의 시작주소) 이다. 메모리덤프를 위한 크기주소는 암호화된 세그먼트의 시작주소 + cryptsize(12288, 0x3000)에 명시된 암호화된 세그먼트의 크기가 되며, 결과적으로 (0x2000+0x3000)이 되어서 끝 주소는 0x5000 이 된다.

암호 해제된 세그먼트는 GDB 의 명령어인 "dump memory"를 사용해서 검색 할 수 있다.

```
(gdb) dump memory 99bottles.dec 0x2000 (0x2000 + 0x3000)
(gdb) kill
Kill the program being debugged? (y or n) y
(gdb) q

mdsec-iPhone:/var/mobile/Applications/E938B6D0-9ADE-4CD6-83B8-712D0549426D/99Bottles.app root# ls -al 99bottles.dec
-rw-r--r-- 1 root mobile 12288 Mar  4 16:31 99bottles.dec
```

그림 4. dump memory 사용한 검색

결과파일은 cryptsize 의 값과 같은 크기 여야 한다. 암호 해제된 부분은 원래의 암호화된 세그먼트와 대체시켜 원본 바이너리에 저장할 수 있다.

iOS Application (In)Security 번역

```
mdsec-iPhone:/var/mobile/Applications/E938B6D0-9ADE-4CD6-83B8-712D0549426D/99Bottles.app root# dd seek=4096 bs=1 conv=notrunc  
if=./99bottles.dec of=99Bottles  
  
12288+0 records in  
12288+0 records out  
12288 bytes (12 kB) copied, 0.471737 s, 26.0 kB/s
```

그림 5. 암호화된 세그먼트와 대체 - 원본에 저장

끝으로, cryptid 값은 0 으로 설정함으로써 더 이상 암호화되지 않고, 따라서 loader 는 그것에 대한 암호해제를 시도하지 않게끔 한다.

vbindiff 를 사용해서, LC_ENCRYPTION_INFO 명령어의 위치를 찾고, (2100000014000000 hex bytes 값을 검색해서 찾을 수 있다) 이 위치에서 cmdsize(0x21000000)의 16 바이트에 앞에 위치에 해있는 cryptid 값을 0 으로 바꿔준다.

```
99Bottles  
0000 06A0: 21 00 00 00 14 00 00 00 00 10 00 00 00 30 00 00 !..... 0..  
0000 06B0: 01 00 00 00 0C 00 00 00 54 00 00 00 18 00 00 00 ..... T.....  
0000 06C0: 02 00 00 00 00 33 A6 02 00 00 2C 01 2F 53 79 73 .....3.. ../Sys  
0000 06D0: 74 65 6D 2F 4C 69 62 72 61 72 79 2F 46 72 61 6D tem/Library/Fram
```

그림 6. crpyid 값 수정

이 단계에서, App 은 암호해제 되고, 정상적으로 실행 될 것이고 다시금 code signed 하게 될 것이다.

3.2. Locating the Position Independent Executable

위치독립 실행파일(PIE)는 Exploit 완화 보안기능으로 어플리케이션이 FULL ASLR 을 이용 할 수 있도록 해준다. 이것을 이용하려면 XCode 를 이용해서 컴파일 할 때 "-fPIE -pie" Flag 를 이용하면 된다.

이것은 compiler code generation build setting 의 "Generate Position-Dependent Code" 옵션을 사용함으로써 활성화/비활성화 할 수 있다.

이전에도 언급 했듯이 PIE 없이 컴파일 된 App 들은 고정된 주소에서 실행하게 될 것이다.

아래의 간단 예제는 main 함수의 주소를 보여 줄 것 이다.

```
int main(int argc, const char* argv[])  
{  
    NSLog(@"Main: %p\n", main);  
    return 0;  
}
```

그림 7. 간단한 예제 작성

iOS Application (In)Security 번역

위의 어플리케이션을 PIE 없이 컴파일하고, iPhone 에서 실행시켜보면 System wide ASLR 임에도 불구하고 main 함수가 고정주소에서 load 되는 것을 볼 수가 있다.

```
mdsec-iPhone:~ root# for i in `seq 1 5`; do ./nopie-main;done
2012-03-01 16:56:17.772 nopie-main[8943:707] Main: 0x2f3d
2012-03-01 16:56:17.805 nopie-main[8944:707] Main: 0x2f3d
2012-03-01 16:56:17.837 nopie-main[8945:707] Main: 0x2f3d
2012-03-01 16:56:17.870 nopie-main[8946:707] Main: 0x2f3d
2012-03-01 16:56:17.905 nopie-main[8947:707] Main: 0x2f3d
```

그림 8. main 함수 고정주소 값

같은 어플리케이션을 PIE 옵션을 이용해서 컴파일 하면 아래의 그림처럼 main 함수가 동적 주소에서 실행되는 것을 볼 수 있을 것 이다.

```
mdsec-iPhone:~ root# for i in `seq 1 5`; do ./pie-main;done
2012-03-01 16:57:32.175 pie-main[8949:707] Main: 0x2af39
2012-03-01 16:57:32.208 pie-main[8950:707] Main: 0x3bf39
2012-03-01 16:57:32.241 pie-main[8951:707] Main: 0x3f39
2012-03-01 16:57:32.277 pie-main[8952:707] Main: 0x8cf39
2012-03-01 16:57:32.310 pie-main[8953:707] Main: 0x30f39
```

그림 9. main 함수 동적 주소로 실행

blackbox 과점에서, PIE 는 Mach-O Header 검사기능을 갖고 있는 otools 를 이용해서 확인 할 수 있다. 예를 들면, 위의 두 개의 바이너리파일을 비교하면 쉽게 PIE 파일을 찾아 낼 수 있다.

```
mdsec-iPhone:~ root# otool -hv pie-main nopie-main
pie-main:
Mach header
      magic cputype cpusubtype  caps   filetype ncmds sizeofcmds      flags
      MH_MAGIC   ARM           9    0x00   EXECUTE   18      1948    NOUNDEFS
DYLDLINK TWOLEVEL PIE

nopie-main:
Mach header
      magic cputype cpusubtype  caps   filetype ncmds sizeofcmds      flags
      MH_MAGIC   ARM           9    0x00   EXECUTE   18      1948    NOUNDEFS
DYLDLINK TWOLEVEL
```

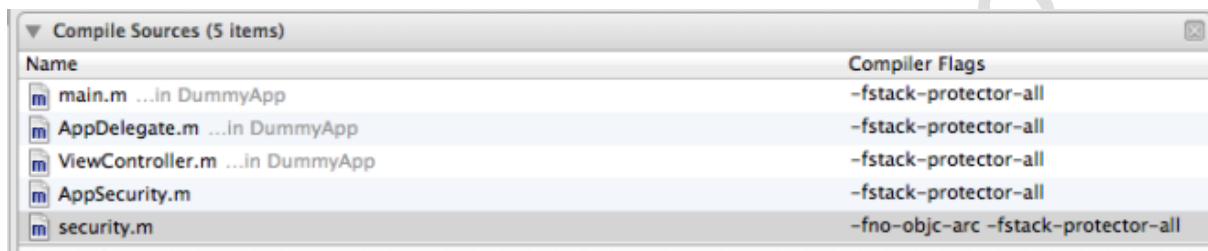
그림 10. PIE 값 확인 가능

iOS Application (In)Security 번역

iOS 5 에서, 기본적으로 설치되어 있는 모든 어플리케이션들은 기본적으로 PIE 컴파일 된 것들이다. 하지만 실제로 Third-party 어플리케이션들은 공통적으로 이러한 보호기능을 이용하지 못하고 있다.

3.3. Identifying the use of Stack Smashing Protection

앞에서도 언급했듯이, iOS 어플리케이션들은 컴파일 시에 Stack smashing protection 을 사용할 수 있다. 이과정의 아래와 같이 "-fstack-protector-all" 이라는 컴파일 Flag 를 사용하면 적용된다.



Name	Compiler Flags
main.m ...in DummyApp	-fstack-protector-all
AppDelegate.m ...in DummyApp	-fstack-protector-all
ViewController.m ...in DummyApp	-fstack-protector-all
AppSecurity.m	-fstack-protector-all
security.m	-fno-objc-arc -fstack-protector-all

그림 11. -fstack-protector-all Flag 사용

stack smashing protection 으로 app 을 컴파일 할 때 "canary" 저장된 기본 포인터(base pointer), 저장된 명령 포인터(instruction pointer) 와 함수의 인자를 보호하기 위해서 지역변수 전에 스택에 직접 적재된다. "canary" 의 값은 함수가 그것이 덮어 쓰였는지 확인하기 위해서 리턴 될 때 검증된다.

컴파일러는 함수에 지능적 스택 보호기법을 적용하기 위해서 발견적 기법(Huristic)을 사용한다. 블랙박스 관점에서 스택 "canary" 의 존재는 바이너리의 심벌테이블을 검사함으로써(by examining) 확인 할 수 있다.

만약 stack smashing protection 이 어플리케이션에 컴파일 되어있다면 두 개의 정의되지 않은 심벌들 예컨대 "__stack_chk_fail", "__stack_chk_guard"가 보일 것이다.

app 으로부터의 심벌 테이블은 otool 을 이용해서 dump 를 뜯 수 있다.

```
$ otool -I -v DummyApp | grep stack
0x00003fc4    14  __stack_chk_fail
0x0000400c    14  __stack_chk_fail
0x0000406c    15  __stack_chk_guard
```

그림 12. otool 을 이용하여 dump

3.4. Identifying the use of Automatic Reference Counting

자동 참조 카운팅(ARC)는 iOS 5.0 SDK 에서 소개되었는데 개발자로부터의 메모리관리부분을 컴파일러로 옮기기 위해서 도입되었다.

결과적으로 ARC 는 마찬가지로 몇몇 보안적 혜택을 제공한다. 그것은 개발자들로 하여금 메모리사용상의 문제들 (특히 객체의 사용 후 메모리 해제문제나, 이중해제), app 들의 취약점의 가능성을 줄일 수 있다.

ARC 는 XCode 의 컴파일러 설정 중 "Objective-C Automatic Reference Counting"을 "YES"로 설정해줌으로써 사용할 수 있다.

컴파일된 App 에서의 BlockBox 리뷰에서 ARC 의 존재를 확인하기 위해서는 아래와 같이 심벌테이블에 있는 ARC 와 관련된 심벌들의 확인을 통해서 가능하다.

```
$ otool -I -v DummyApp-ARC | grep "_objc_release"
0x00003fe8 181 _objc_release
0x00004030 181 _objc_release
$
```

그림 13. ARC 와 관련된 심벌 확인

위 그림에서 빨간색으로 표시되어 있는 ARC 심벌은 다음과 같다.

```
_objc_retainAutoreleaseReturnValue
_objc_autoreleaseReturnValue
_objc_storeStrong
_objc_retain
_objc_release
_objc_retainAutoreleasedReturnValue
```

컴파일 시에 ARC 는 명시적으로 특정 소스파일을 "-fno-objc-arc" 이라는 컴파일 Flag 를 이용함으로써 사용할 수 없게 할 수 있고, 이것은 iOS 어플리케이션 white box 평가의 일부분으로써 강조되어진다.

3.5. inspecting the Binary

바이너리를 암호 해제 시에 리버스 엔지니어에게 유용하게 쓰일 수 있는 __OBJC 세그먼트라는 것이 있다. __OBJC 세그먼트는 app 에 사용되었던 내부 클래스, 메서드와 변수 등을 제공해준다.

이러한 정보는 app 을 패치 하거나 app 의 실시간 hooking, app 의 기능들을 이해하고 조사하는데 특히 유용하다.

iOS Application (In)Security 번역

_OBJC 세그먼트를 분석에 사용하는 도구는 class-dmp-z[11]이다.

아래의 그림은 이전에 cryptid 와 함께 다뤘던 암호해제에 사용한 99Bottles App 을 class-dump-z 를 사용해 추출한 결과이다.

```
@interface BottleLayer : CALayer {
    @private
        BOOL flown;
    }
    @property(assign, nonatomic) BOOL flown;
    -(void)drawInContext:(CGContextRef)context;
    -(void)jiggle;
    -(void)flyAway;
    -(void)animationDidStop:(id)animation finished:(BOOL)finished;
    -(void)dealloc;
@end

__attribute__((visibility("hidden")))
@interface RootViewController : UIViewController <UISheetDelegate> {
    @private
        UILabel* numberDisplay;
        NSMutableArray* marr;
        Player* player;
        UIView* wall;
        BottleLayer* currentBottle;
        NSArray* names;
        NSArray* names10;
        int count;
        BOOL paused;
    }
```

그림 14. class-dump-z 사용

위의 예제에서 볼 수 있듯이 class-dump-z 를 통해서 "jiggle", "flyAway", "drawInContext"등을 포함해서 많은 숫자의 메서드들을 확인할 수 있다. 이것들을 실시간 hooking, 또는 수정 할 수 있다.

3.6. Manipulating the Runtime

Object-c 의 실시간 Hooking 은 어플리케이션의 내부 동작의 관찰이나 수정할 수 있는 강력한 방법이다. 실시간 Hooking 의 가장 일반적인 방법은 탈옥된(jailbreken)기기를 위한 hooking framework 인 MobileSubstrate 를 사용하는 것이다. 이것은 Mac OS X 에서의 Application Enhancer 와 유사하다.

iOS Application (In)Security 번역

MobileSubstrate 는 일반적으로 많은 iOS 의 탈옥과 Hooking 의 용이성을 위해서 Object-C, 뿐만 아니라 C, C++를 기본으로 제공하고 있다.

Cycript[13]은 JavaScript 를 Object-C 로 변환해주는 역할을 하는 command line tool 이다. 뿐만 아니라 Cycript 는 JavaScript 와 Object-C 를 혼용해서 MobileSubstrate 를 이용해서 실시간 Hooking 할수 있는 기능도 제공한다.

아마도 Cycript 의 가장 유용한 기능은 실행중인 프로세서에 연결(Attach)하는 기능, 실시간 조작(manipulate)기능이다. 예컨대 cycript 는 탈옥(jailbroken) 기기에서 실행중인 SpringBoard 프로세스에 코드삽입(inject)을 하거나, passcode 를 우회하는 방법을 사용하여 passcode 요구부분을 무효화 시킬 수 있다.

```
mdsec-iPhone:~/Documents/Cracked root# cycript -p SpringBoard
cy# SBAwayController.messages['isPasswordProtected'] = function() {return NO;}
{}
cy# [SBAwayController.sharedAwayController unlockWithSound:1]
cy#
```

그림 15. SpringBoard 프로세스에 코드 삽입 등 작업 가능

MobileSubstrate 확장기능을 사용하고자 하는 사람들을 위해서 iOSSOpenDev 는 XCode 템플릿을 이용해서 XCode 에 MobileSubstrate 를 통합 할 수 있는 수단을 제공한다.

iOSSOpenDev[14]는 MobileSubstrate tweak 을 단순화하기 위해서 CaptainHook Framework 를 사용한다.

Example:Bypassing Jailbreak Detection

예를 들어, 탈옥된(jailbroken) 기기에서 App 실행을 감지하고, 실행을 금지시키는 App 을 고려해보자. 이러한 기능을 위해서 할 수 있는 일반적인 방법은 알려진 jailbreak 와 관련된 파일시스템을 체크하는 방법이다.

[예제]

```
@implementation AppSecurity
-(BOOL)isJailBroken
{
    NSString *filePath = @"/Applications/Cydia.app";
    if ([[NSFileManager defaultManager] fileExistsAtPath:filePath])
    {
        return TRUE;
    }
    return FALSE;
}
```

그림 16. jailbreak 관련 파일시스템 체크

iOS Application (In)Security 번역

위에 보이는 메서드는 MobileSubstrate tweak 를 사용하여 다음과 같이 실시간으로 hook 또는 수정 될 수 있다.

```
#import <Foundation/Foundation.h>
#import <CaptainHook/CaptainHook.h>
#include <notify.h>

@interface hookDummy : NSObject
@end

@implementation hookDummy

-(id)init
{
    if ((self = [super init])){}
    return self;
}
@end

@class AppSecurity;
CHDeclareClass(AppSecurity);
CHOptimizedMethod(0, self, BOOL, AppSecurity, isJailBroken)
{
    NSLog(@"##### isJailBroken hooked");
    return false;
}

CHConstructor
{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    CHLoadLateClass(AppSecurity);
    CHHook(0, AppSecurity, isJailBroken); // register hook
    [pool drain];
}
```

그림 17. MobileSubstrate tweak 사용

일단, 컴파일 후에 라이브러리를 DynamicLibraries 폴더에 넣는다. 왜냐하면 기기에서 어플리케이션이 매번 실행될 때마다 로드 되도록(loaded) 하기 위해서다.

```
-rwxr-xr-x 1 root wheel 10912 Mar  8 10:15
/Library/MobileSubstrate/DynamicLibraries/hookDummy.dylib*
```

그림 18. DynamicLibraries 폴더에 저장

iOS Application (In)Security 번역

```
Mar  8 21:03:56 unknown DummyApp[1722] <Notice>: MS:Notice: Installing:
MDSec.DummyApp [DummyApp] (675.00)
Mar  8 21:03:56 unknown DummyApp[1722] <Notice>: MS:Notice: Loading:
/Library/MobileSubstrate/DynamicLibraries/Activator.dylib
Mar  8 21:03:56 unknown DummyApp[1722] <Notice>: MS:Notice: Loading:
/Library/MobileSubstrate/DynamicLibraries/hookDummy.dylib
Mar  8 21:03:56 unknown kernel[0] <Debug>: launchd[1722] Builtin profile:
container (sandbox)
Mar  8 21:03:56 unknown kernel[0] <Debug>: launchd[1722] Container:
/private/var/mobile/Applications/1F6A9800-DBD0-4831-A7C9-C4826C6F7EAD [69]
(sandbox)
Mar  8 21:03:57 unknown DummyApp[1722] <Warning>: ##### isJailBroken hooked
```

그림 19. 매번 실행 될 때마다 로드

라이브러리는 설정할 수 있다 application bundle identifier 를 포함하고 있는 plist 파일을 생성하므로 써 특정어플리케이션이 실행될 때만 로드 되도록 설정할 수 있다.

```
Filter = {
    Bundles = (MDSec.DummyApp);
};
```

그림 20. 특정 어플리케이션 실행 될 때 로드

실제로, CommBank 사의 Kaching 어플리케이션은 이와 유사한 방법으로 탈옥된(jailbroken) 기기를 식별해 낸다. class-dump 를 이용해서 이와 관련된 메서드를 확인할 수 있다.

```
@interface RootViewController : /private/tmp/KIA_IPHONE_SOURCE/
<UIWebViewDelegate, DILDisplayView, UIAlertViewDelegate>
{
    <snip>
    - (BOOL) isJailbrokenDevice;
```

그림 21. class-dump 을 이용하여 관련된 메서드 확인

iOS Application (In)Security 번역



그림 22. 경고 메시지 확인

탈옥된(jailbroken) 기기에서 App 을 실행시킬 때, App 은 위와 같은 UIAlertView 를 통해서 경고 문구를 보여주고 더 이상 진행되지 않는다.

다음의 MbileSubstrate tweak 를 이와 같은 보호기능을 다음과 같은 방법을 사용해서 우회하는 방법을 사용해서 정상적으로 App 을 사용할 수 있다.

```
@class RootViewController;
CHDeclareClass(RootViewController);
CHOptimizedMethod(0, self, BOOL, RootViewController, isJailbrokenDevice)
{
    NSLog(@"##### isJailbrokenDevice hooked");
    return false;
}
CHConstructor
{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    CHLoadLateClass(RootViewController);
    CHHook(0, RootViewController, isJailbrokenDevice);
    [pool drain];
}
```

그림 23. Jailbreaking 우회

App 을 다시 실행하면 정상적으로 실행된다. 왜냐하면 isJailBrokenDevice 메서드가 hook 되었고, 수정되어졌기 때문이다.

iOS Application (In)Security 번역

```
Mar  8 21:15:46 unknown KIA[1786] <Notice>: MS:Notice: Installing:
au.com.commbank.kaching [KIA] (675.00)

Mar  8 21:15:46 unknown KIA[1786] <Notice>: MS:Notice: Loading:
/Library/MobileSubstrate/DynamicLibraries/Activator.dylib

Mar  8 21:15:46 unknown kernel[0] <Debug>: launchd[1786] Builtin profile:
container (sandbox)

Mar  8 21:15:46 unknown kernel[0] <Debug>: launchd[1786] Container:
/private/var/mobile/Applications/63DC8037-5A2F-4C5C-ADDB-30AF3BF49449 [69]
(sandbox)

Mar  8 21:15:47 unknown KIA[1786] <Notice>: MS:Notice: Loading:
/Library/MobileSubstrate/DynamicLibraries/hookDummy.dylib

Mar  8 21:15:47 unknown securityd[1787] <Notice>: MS:Notice: Installing: (null)
[securityd] (675.00)

Mar  8 21:15:47 unknown securityd[1787] <Notice>: MS:Notice: Loading:
/Library/MobileSubstrate/DynamicLibraries/hookDummy.dylib

Mar  8 21:15:47 unknown KIA[1786] <Warning>: **** READ 60 LOG ENTRIES FROM DISK
****

Mar  8 21:15:47 unknown KIA[1786] <Warning>: ##### isJailbrokenDevice hooked
```

그림 24. isJailBrokenDevice Hook

3.7. Defending the Binary

개발자들은 실시간 공격들을 완화하거나 리버스엔지니어링의 복잡도를 증가시켜 공격자에 대한 방어책으로 몇몇 대응전략을 채택하고 있다. 하지만 일반적으로 탈옥된(jailbroken) 환경에서 실행한다면 완벽한 대비책이 없다는 것을 인정 해야 한다.

실시간 방어책으로 가장 일반적인 접근방법 중의 하나는 Object-C 가 실시간 Hook 또는 수정이 되었을 때 app 으로 하여금 예상되는 주소들(expected addresses), 체크섬(checksum)의 무결성을 체크하는 클래스를 사용하여 확인하는 방법이다. (One of the most common approaches for defending the runtime is to integrity check classes for expected addresses or checksums, allowing an app to determine if the Objective-C runtime has been hooked or modified.)

이러한 접근법은 전통적으로 클래스의 주소를 검색하고 검증함으로써 이루어진다. 런타임은 클래스 메서드가 호출되면 function pointer 를 리턴 해주는 class_getMethodImplementation 메서드를 제공한다.

[간단한 구현]

iOS Application (In)Security 번역

```
#import "security.h"
@implementation security
void * perform_sec_check()
{
    void * addr = verify_address("AppSecurity", "isJailBroken");
    fprintf(stderr, "\ncaddr = %p\n", addr);
    if(addr != 0x25a9) take_evasive_action();
}
void * verify_address(const char * cname, const char * method)
{
    id class = objc_lookUpClass(cname);
    SEL selector = sel_registerName(method);
    IMP imp = class_getMethodImplementation(class, selector);
    return imp;
}
void * take_evasive_action() {
    fprintf(stderr, "%s", "Tamper detected\n");
    exit(-1);
}
@end
```

그림 25. class_getMethodImplementation 메서드를 제공

위의 클래스는 간단한 런타임 조작 감지 기능을 구현한 것이다. verify_address 함수는 이전의 예제에서 다뤘던 AppSecurity: isJailBroken 메서드 function pointer 의 주소를 검색한다. 이 주소는 개발자에 의해 하드코드 된 알려진 safe address 와 비교하게 된다. 만약 주소가 다르면 변조가 된 것이고, 적절한 조치를 취하게 된다.

실시간 hooking 없이 실행한 어플리케이션은 jailbreak 감지기능이 정상적으로 실행될 것이다.

```
caddr = 0x25a9
2012-04-18 20:51:51.580 DummyApp[595:707] ##### Sorry, you are running on a
jailbroken device
```

그림 26. jailbreak 감지 기능 정상 실행

빨간색으로 표시된 주소는 AppSecurity: isJailBroken 메서드의 예상주소(expected address)이다. 위에 보이는 주소는 어플리케이션을 MobileSubstrate Library 로 다시 실행하면 AppSecurity: isJailBroken 메서드의 주소는 아래와 같이 바뀌게 된다.

```
caddr = 0x76ec5
Tamper detected
```

그림 27. isJailBroken 메소드 변경

iOS Application (In)Security 번역

위의 anti-tamper 감지(Detection)는 효과적이지만 이것 또한 감지부분의 Hooking, 또는 바이너리 패치 등의 방법으로 우회할 수 있다. 개선된 감지(Detection)를 위해, 컴파일러는 함수가 호출될 때마다 본문에 함수전체를 삽입하기 때문에 함수는 그때 그때 처리(inlined)하도록 할 수 있다.

따라서 공격자는 함수가 호출될 때마다 매번 호출되는 모든 함수를 매번 패치를 해야 하는 상황이 벌어진다.

이것은 다음과 같은 keyword inline 을 사용함으로써 얻을 수 있다.

```
inline void * perform_sec_check()
{
    void * addr = verify_address("AppSecurity", "isJailBroken");
    fprintf(stderr, "\ncaddr = %p\n", addr);
    if(addr != 0x25a9) take_evasive_action();
}
```

그림 28. keyword inline 사용

3.8. 결론

결과적으로, iOS 어플리케이션의 black box 평가 부분 중에 채택된 몇몇 기술 등에 대해서 검토해봤다. 실제로 어플리케이션의 내부적 작업, 심지어 이것들이 AppStore 의 암호화로 인해서 보호되어진 것들까지도 깊이 이해할 수 있는 가능성을 얻게 되었다.

실시간 hooking 은 평가들(asses)과 어플리케이션의 수정의 상호작용을 위한 강력한 수단을 제공한다. 특히, 검토하는 사람(an evaluator)에게 app 의 내부와 리버스엔지니어링이 없으면 기능을 확인할 수 없는 API 같은 내부의 기능을 평가를 할 수 있도록 하고 있다.

방어적 관점에서, 개발자들은 checksum 이나 클래스와 메서드의 런타임 주소를 확인하는 방법을 사용하여 app 들의 변조(tempering)로부터 보호하는 방법을 고려해야 한다. 이 같은 보호효과는 inline function 들을 사용하므로써 더욱 향상시킬 수 있다.

가능하면 개발자들은 리버스엔지니어링의 복잡도를 높이고, 클래스의 구조로 인해서 들어나는 많은 정보를 줄이는 방향으로 코딩을 해야 한다.

4. 안전하지 않은 API 사용 진단

iOS 어플리케이션들은 서버, 어플리케이션 내부자원활용, 다른 어플리케이션과의 상호작용을 위해서 보통 표준 API 를 활용한다. 많은 기본 보안 값을 구현하는 동안 MDSec 는 기본 옵션들을 사용하지 않거나, API 가 안전하게 작동하기 위해서 간단하게만 신뢰 되어진 많은 어플리케이션을 검사했다.

iOS 어플리케이션들의 소스코드를 검토할 때 다음의 핵심부분(key touch point)을 검토해야만 한다.

4.1. Evaluating Transport Security

대부분의 iOS 어플리케이션들은 모바일기기의 본래의 기능 때문에 네트워크 통신을 하는데 이러한 통신은 종종 호텔이나, Café 의 Wi-fi, 모바일기기의 hotspot 이나 휴대폰과 같은 신뢰받지 않은 혹은 보안에 취약한 통신을 하곤 한다. 따라서 이와 같은 통신의 보안이 꼭 필요하다.

iOS app 들은 보통 RPC 메커니즘에 기반을 둔 웹 통신을 하거나, 온라인 웹 어플리케이션과 상호작용을 한다. 이 같은 상호작용들은 NSURLConnection 클래스를 통해 이루어진다.

이 클래스는 NSURLRequest 라는 객체를 이용하고, NSURLRequest 객체를 사용해서 HTTP(S)프로토콜을 요청하게 된다.

API 는 보안접속을 하기 위해서 SSL(Secure Socket Layer) 암호를 기본설정으로 사용한다.

불행히 API 는 개발자로 하여금 절충적으로 사용 가능한 암호군으로부터의 암호의 선택의 폭이 충분치 않다. (unfortunately the API is not granular enough to allow the developer to select which ciphers from the suite to negotiate with.) SDK 에 버전 차이로 인한 절충 되어진 전송의 몇몇 차이점이 있다.

이러한 요약은 아래의 테이블과 같다.

SDK Version	Protocol	"Weak" Cipher Suites	Total Cipher Suites
4.3	TLS 1.0	5	29
5.0	TLS 1.2	0	37
5.1	TLS 1.2	0	37

표 1. SDK 버전별 전송 체계

iOS Application (In)Security 번역

테이블에서 강조된 부분은 각 버전의 갱신에 따른 암호 군(the cipher suites)의 개선된 절충(negotiated)되어진 부분이다. (The table highlights an improvement in the cipher suites negotiated over time with the release of the newer versions of the SDK.)

다음의 예제는 로컬호스트에 대한 간단한 HTTPS 접속을 구현 한 것이다.

```
@implementation insecuressl
int main(int argc, const char* argv[])
{
    NSString *myURL=@"https://localhost/test";
    NSURLRequest *theRequest = [NSURLRequest requestWithURL:[NSURL
    URLWithString:myURL]];
    NSURLResponse *resp = nil;
    NSError *err = nil;
    NSData *response = [NSURLConnection sendSynchronousRequest:
    theRequest returningResponse: &resp error: &err];
    NSString * theString = [[NSString alloc] initWithData:response
    encoding:NSUTF8StringEncoding];
    [resp release];
    [err release];
    return 0;
}
```

그림 29. HTTPS 접속 구현

어플리케이션을 4.4, 5.0, 5.1 버전의 SDK 를 이용해서 컴파일 한 후에 각각의 실행결과를 모니터링 한 결과 각기 다른 결과 값을 나타낸다.

4.3 버전의 SDK 를 이용해서 컴파일 한 어플리케이션은 그림 3, 4 와 같이 TLS 1.0 세션을 사용하고 29 개의 암호 군중의 하나를 사용한다.

iOS Application (In)Security 번역

```
Handshake Type: Client Hello (1)
Length: 135
Version: TLS 1.0 (0x0301)
▷ Random
Session ID Length: 0
Cipher Suites Length: 58
▷ Cipher Suites (29 suites)
Compression Methods Length: 1
▷ Compression Methods (1 method)
Extensions Length: 36
▷ Extension: server_name
▷ Extension: elliptic_curves
▷ Extension: ec point formats
```

그림 30. 4.3 버전 - TLS 1.0 버전 사용

```
▽ Cipher Suites (29 suites)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_RC4_128_SHA (0xc007)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc008)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
Cipher Suite: TLS_ECDHE_RSA_WITH_RC4_128_SHA (0xc011)
Cipher Suite: TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (0xc012)
Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA (0xc004)
Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA (0xc005)
Cipher Suite: TLS_ECDH_ECDSA_WITH_RC4_128_SHA (0xc002)
Cipher Suite: TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc003)
Cipher Suite: TLS_ECDH_RSA_WITH_AES_128_CBC_SHA (0xc00e)
Cipher Suite: TLS_ECDH_RSA_WITH_AES_256_CBC_SHA (0xc00f)
Cipher Suite: TLS_ECDH_RSA_WITH_RC4_128_SHA (0xc00c)
Cipher Suite: TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA (0xc00d)
Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
Cipher Suite: TLS_RSA_WITH_DES_CBC_SHA (0x0009)
Cipher Suite: TLS_RSA_EXPORT_WITH_RC4_40_MD5 (0x0003)
Cipher Suite: TLS_RSA_EXPORT_WITH_DES40_CBC_SHA (0x0008)
Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
Cipher Suite: TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (0x0016)
Cipher Suite: TLS_DHE_RSA_WITH_DES_CBC_SHA (0x0015)
Cipher Suite: TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA (0x0014)
Compression Methods Length: 1
```

그림 31. 29 개 암호군 중에 사용

5.0 혹은 5.1 버전의 SDK 를 이용해서 컴파일 한 어플리케이션은 그림 5, 6 와 같이 TLS 1.2 세션을 사용하고 37 개의 암호 군중의 하나를 사용한다.

iOS Application (In)Security 번역

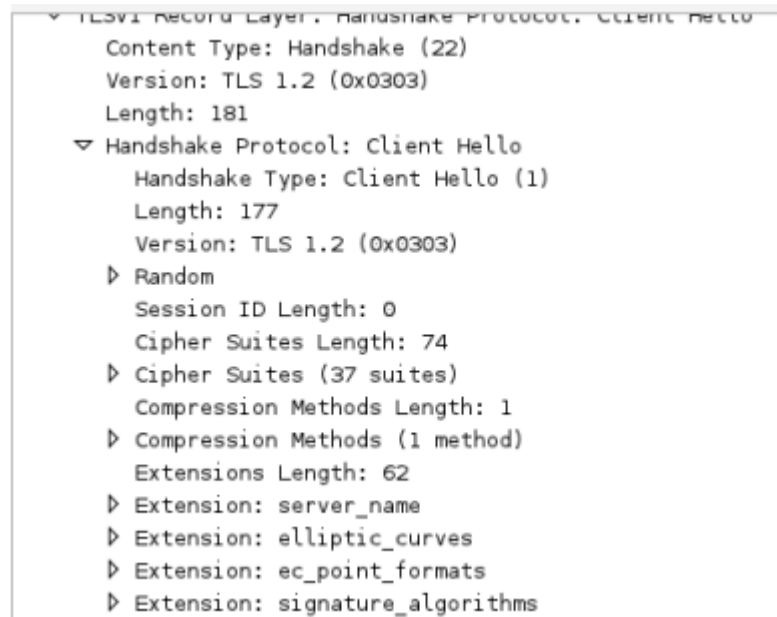


그림 32. SDK 5.0 or 5.1 – TLS 1.2 세션 사용

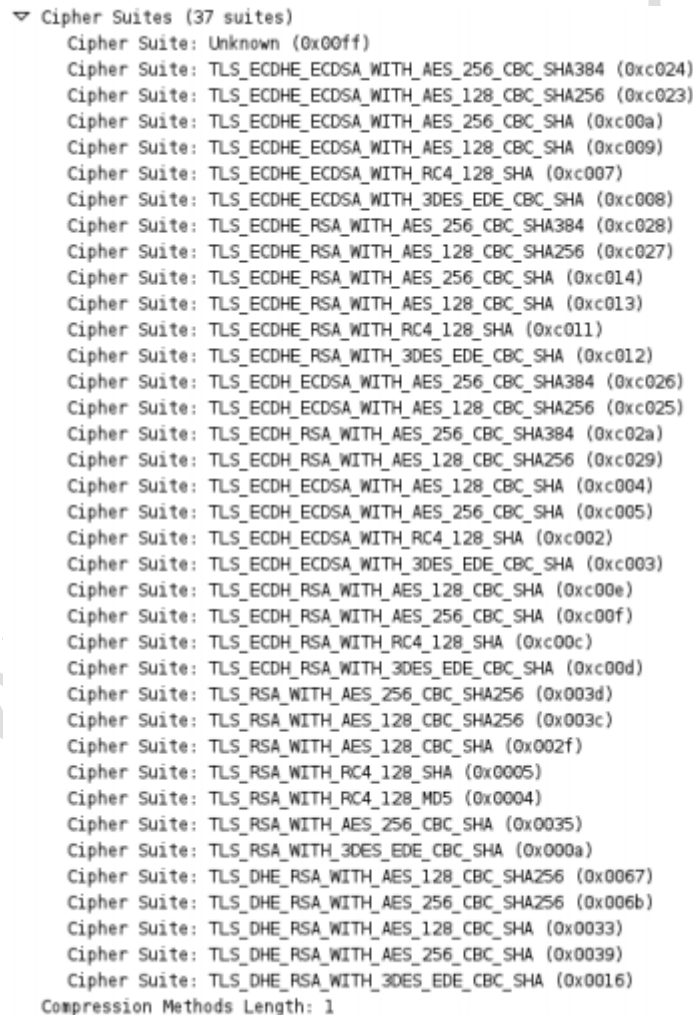


그림 33. 37 개 암호군 중 사용

4.3 SDK 절충(negotiation)에서, 다음의 암호군들(cipher suites)은 취약한 것으로 보인다:

iOS Application (In)Security 번역

- TLS_RSA_WITH_DES_CBC_SHA
- TLS_RSA_EXPORT_WITH_RC4_MD5
- TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
- TLS_DHE_RSA_WITH_DES_CBC_SHA
- TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA

iOS 어플리케이션에서 MiTM(Man-in-the-Middle)공격을 방지하기 위해서 자체 서명된 인증서(self-signed certificate)을 필수적으로 금지 시켜야 한다. 기본적으로 NSURLRequest 클래스를 위한 조치로 자체 서명된 인증서(self-signed certificate)을 사용하면 NSErrorDomain 의 예외처리가 된다.

하지만, 가끔씩 개발자들이 모든 종류의 인증서(certificate)를 수용하기 위해서 이와 같은 동작 (behaviour)을 덮어쓰는 경우는 보기 힘들지만, 사전 제작환경에서 배치되어진(deployed) 자체 서명된 인증서(self-signed certificate)를 사용하기 위해서 빈번하게 사용한다.

인증서 유효성검사는 allowsAnyHTTSPCertificateForHost 메서드를 사용함으로써 도메인 요청을 기능을 사용하지 못하도록 할 수 있다.

다음의 예제와 유사하게 사용:

```
#import "loadURL.h"

@interface NSURLRequest (DummyInterface)
+ (BOOL)allowsAnyHTTSPCertificateForHost:(NSString*)host;
+ (void)setAllowsAnyHTTSPCertificate:(BOOL)allow forHost:(NSString*)host;
@end

@implementation loadURL
-(void) run
{
    NSURL *myURL = [NSURL URLWithString:@"https://localhost/test"];
    NSMutableURLRequest *theRequest = [NSMutableURLRequest requestWithURL:myURL
    cachePolicy:NSURLRequestReloadIgnoringCacheData timeoutInterval:60.0];
    [NSURLRequest setAllowsAnyHTTSPCertificate:YES forHost:[myURL host]];
    [[NSURLConnection alloc] initWithRequest:theRequest delegate:self];
}
@end
```

그림 34. allowsAnyHTTSPCertificateForHost 메서드를 사용

allowsAnyHTTSPCertificateForHost 는 숨겨져 있는 메서드이기 때문에 개발자가 해당 메서드를 사용해서 AppStore 에 App 등록을 할 때 Apple 사로부터 등록이 거부 될 수 있다.

SSL 을 우회하기 위한 대안으로는 보통 사용하지 않는 continueWithoutCredentialForAuthenticationChallenge 셀렉터(Selector)를 사용하는 것이다.

iOS Application (In)Security 번역

NSURLConnection delegate 메서드 안에 didReceiveAuthenticationChallenge 메서드를 아래와 같이 구현하면 사용할 수 있다.

```
- (void)connection:(NSURLConnection *)connection
didReceiveAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge
{
    if ([challenge.protectionSpace.authenticationMethod
        isEqualToString:NSURLAuthenticationMethodServerTrust])
    {
        [challenge.sender useCredential:[NSURLCredential
            credentialForTrust:challenge.protectionSpace.serverTrust]forAuthenticationChallen
            ge:challenge];

        [challenge.sender
            continueWithoutCredentialForAuthenticationChallenge:challenge];
        return;
    }
}
```

그림 35. didReceiveAuthenticationChallenge 메서드 구현

CFNetwork 프레임워크는 SSL 구현을 위해서 대체 API 를 제공해준다. 실제로 프레임워크는 개발자에게 SSL 세션 구현을 보다 다양한 제어와 사용자화(customisation)를 제공한다. NSURLRequest 와 유사하게, 개발자들에서 취약한 SSL 구성을 제공해주지는 않는다.

하지만 CFNetwork 는 보다 세분화된 컨트롤, 어플리케이션의 만료된 인증서, root 인증, 인증서체인(certificate chain)의 유효성도 검사할 수 있도록 허용하고 있다. 다음의 onSocket delegate 메서드는 실제 어플리케이션으로부터 가져온 것이다.

```
- (void)onSocket:(AsyncSocket *)sock didConnectToHost:(NSString *)host
port:(UInt16)port {
    NSMutableDictionary *settings = [[NSMutableDictionary alloc]
        initWithCapacity: 3];

    [settings setObject:[NSNumber numberWithInt:YES]
        forKey:(NSString *)kCFStreamSSLAllowsExpiredCertificates];
    [settings setObject:[NSNumber numberWithInt:YES]
        forKey:(NSString *)kCFStreamSSLAllowsAnyRoot];
    [settings setObject:[NSNumber numberWithInt:NO]
        forKey:(NSString *)kCFStreamSSLValidatesCertificateChain];
    [sock startTLS:settings];
}
```

그림 36. onSocket delegate 메서드

불행히도, CFNetwork 프레임워크를 사용 시에 암호군 (cipher suite)의 수정할 방법은 명확하지 않고 또 다시 SDK 의 기본설정 암호들(ciphers)이 사용된다.

결론적으로, 모바일 어플리케이션들의 전송 메서드 구현 시에는 보안에 적절하게 최종 버전의 SDK 의 기본모드와 기본설정을 사용하는 것이 필수이다.

iOS Application (In)Security 번역

하지만 API 들은 보안취약부분에 보안적 전송을 허용하고 있고, 보통 이것이 개발자에 의해서 구현되는 것 흔치 않다. 개발자는 개발할 때 일시적으로 취약해진 전송보안에 주의해야 한다. 혹은 staging environments 에서는 제품에 이러한 코드를 계속 사용하지 않도록 주의해야만 한다.

(Developers looking to temporarily weaken transport security for development or staging environments should be cautious to ensure that this code does not persist in production.)

이것을 달성하기 위한 가장 간단한 방법은 development build 시에만 사용하는 전 처리 매크로 코드를 포함하는 것이다.

4.2. Abusing Protocol Handlers

iOS sandbox 에 도입된 제한들 때문에 Inter-Process Communication (IPC)는 일반적으로 금지되어 있다. 그러나 만약 어플리케이션에 등록된 임의의 프로토콜 처리를 하는 경우 API 는 간단한 IPC 형식을 지원한다.

개발자들이 IPC 를 지원하는 여러 가지 이유가 있다. 실제로 우리가 접해본 몇몇의 예제 app 들은 Safari 로부터 app 이 실행되는 것을 허용하거나 app 들과의 데이터 전송을 허용하고 있다. iOS 에는 통상적으로 protocol handler 들을 구현할 수 있고 그것들은 "application:openURL", "application:handleOpenURL" 2 개의 API 메서드가 있다. 후자는 현재는 사용되지 않는다.

"openURL" 메서드는 URL Request 인스턴스를 요청받은 소스어플리케이션의 유효성검사를 지원하는 이점이 있다. iOS 어플리케이션은 plist 파일에 URL type 을 추가함으로써 custom URL 스키마를 등록할 수 있다. 아래의 "vuln" 이라는 custom URL 스키마를 등록한 VulnerableiPhoneApp 프로젝트에 볼 수 있다.

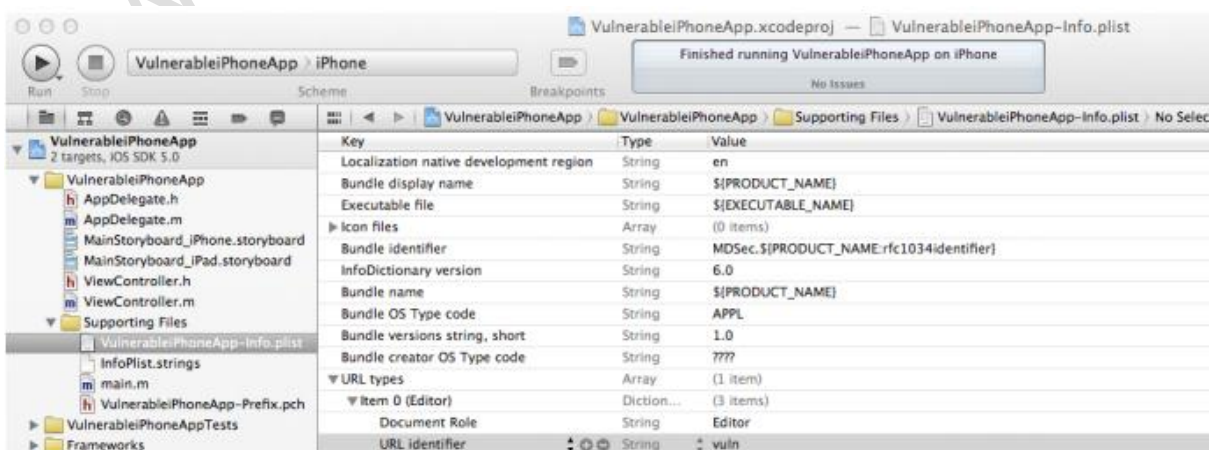


그림 37. custom URL 스키마를 등록한 VulnerableiPhoneApp 프로젝트

iOS Application (In)Security 번역

프로토콜의 코드처리는 아래의 그림에서 볼 수 있듯이 요청한 URL 텍스트를 보여주는 alertView 에서 구현했듯 application delegate 메서드인 handleOpenURL 이나 openURL 을 이용해서 구현할 수 있다.

```
- (BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url {
    UIAlertView *alertView;
    NSString *text = [[url host]
        stringByReplacingPercentEscapesUsingEncoding:NSUTF8StringEncoding];
    alertView = [[UIAlertView alloc] initWithTitle:@"Text" message:text
        delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil];
    [alertView show];
    return YES;
}
```

그림 38. handleOpenURL 이나 openURL 을 이용

실제 어플리케이션에서 설정을 변경함으로써 custom URL 구현해서 처리한 것을 발견할 수 있다: 이 기능은 처음에 개발자의 편의를 위해서 어플리케이션에 내장되었지만 점차 제품의 Release 통해 지속적으로 지원하게 되었다.

다음의 handleOpenURL 메서드의 구현을 보자.

```
- (BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url {
    if (!url) { return NO; }
    NSString *method = [[url host]
        stringByReplacingPercentEscapesUsingEncoding:NSUTF8StringEncoding];
    if([method isEqualToString:@"setHomeURL"])
    {
        Settings *s = [[Settings alloc] init];
        NSString *querystr = [[url query]
            stringByReplacingPercentEscapesUsingEncoding:NSUTF8StringEncoding];
        NSArray *param = [querystr componentsSeparatedByString:@"="];
        NSString *value = [param objectAtIndex:1];
        [s setHomeURL:value];
    }
    return YES;
}
```

그림 39. handleOpenURL 메서드의 구현

위의 예제에서 custom URL Handler 는 어플리케이션이 실행될 때 기본 URL 을 업데이트하는데 사용된다. 메서드는 NSURL 객체를 받아서 parse 되고, 만약 호스트가 setHomeURL 이라는 메서드를 전달해주면 메서드는 첫 번째 URL paramater 의 값과 함께 사용자가 설정한 객체와 더불어 setHomeURL 메서드를 호출하게 된다.

iOS Application (In)Security 번역

어플리케이션의 환경설정파일에 사용자가 설정한 setHomeURL 객체를 저장하는 코드구현은 다음과 같다.

```
@implementation Settings
- (void) setHomeURL:(NSString*)url
{
    NSUserDefaults *prefs = [NSUserDefaults standardUserDefaults];
    [prefs setObject:url forKey:@"homeURL"];
    [prefs synchronize];
}
```

그림 40. setHomeURL 객체를 저장

공격자는 다음과 유사한 악성 iframe 을 사용해서 어플리케이션의 설정파일을 재설정함으로써 기본 방문페이지를 변경할 수 있다.

```
<iframe src="vuln://setHomeURL?url=http://mdattacker.net"></iframe>
```

그림 41. iframe 을 사용해서 어플리케이션의 설정파일을 재설정

이 같은 문제를 해결하려면 갱신된 "openURL" API 를 사용하고 마찬가지로 어플리케이션에서 URL Request 에 의한 정보를 제공하는 것이다.

다음의 예제는 어플리케이션에 의해서 요청된 URL 을 확인할 수 있다.

```
- (BOOL)application:(UIApplication *)application openURL:(NSURL *)url
sourceApplication:(NSString *)sourceApplication annotation:(id)annotation {
    NSString* myBid = [[NSBundle mainBundle] bundleIdentifier];
    if ([sourceApplication isEqualToString:myBid])
    {
        return NO;
    }
    else if (!url) { return NO; }
    NSString *method = [[url host]
stringByReplacingPercentEscapesUsingEncoding:NSUTF8StringEncoding];
    if([method isEqualToString:@"setHomeURL"])
    {
        Settings *s = [[Settings alloc] init];
        NSString *querystr = [[url query]
stringByReplacingPercentEscapesUsingEncoding:NSUTF8StringEncoding];
        NSArray *param = [querystr componentsSeparatedByString:@"="];
        NSString *value = [param objectAtIndex:1];
        [s setHomeURL:value];
    }
    return YES;
}
```

그림 42. 요청된 URL 을 확인

iOS Application (In)Security 번역

대안적으로, 개발자가 다른 어플리케이션 (예컨대 Safari 같은 어플리케이션)에서 호출된 URL 을 검사하고 싶다면 다음과 같이 구현하면 된다,

```
- (BOOL)application:(UIApplication *)application openURL:(NSURL *)url
sourceApplication:(NSString *)sourceApplication annotation:(id)annotation {
    NSString *SafariPath = @"/Applications/MobileSafari.app";
    NSBundle *bundle = [NSBundle bundleWithPath:SafariPath];
    if ([sourceApplication isEqualToString:[bundle bundleIdentifier]])
    {
        return No;
    }
}
```

그림 43. 호출된 URL 을 검사

실제로 취약점이 있었던 사례는 iOS 어플리케이션인 채팅을 하면서 전화를 걸 수 있도록 "skype"라는 protocol handler 가 등록 되어져 사용되고 있는 Skype 에서 볼 수 있다.

공격자는 악의적인 iframe 을 이용해서 권한 없이 전화를 걸 수 있고, 이것은 Nitesh Dhanjani [15]에의해서 처음으로 문서화 되어졌다. 공격자는 아래에서 볼 수 있듯 다음의 코드를 MobileSafari 를 통해서 Skype app 를 실행시켜서 악용 할 수 있다.

Skype 는 이 문제를 사용자에게 전화를 걸 것인지 그렇지 않을 것인지 UIAlertView 을 보여줌으로써 해결했다.

```
<iframe src="skype://123456789?call"></iframe>
```

그림 44. UIAlertView 확인

iOS 어플리케이션에서 유효한 URL 를 확인하는 가장 간단한 방법은 해당 app 을 받은 후 app 을 암호해제하고 protocol 문자열을 확인하는 것이다. 아래의 그림의 페이스북 어플리케이션에서 protocol 을 찾아본 것이다.

```
bash-3.2# strings Facebook.app/Facebook | grep "://" | grep -v "http"
fb://upload/actions/newalbum
fb://root
fb://birthdays
fb://messaging
fb://notifications
fb://requests
fb://publish
fb://publish/profile/(gatePublishWithUID:)
fb://oldpublish
fb://oldpublish/profile/(initWithUID:)
fb://publish/post/(initWithPostId:)
fb://publish/photo/(initWithUID:)/(aid:)/(pid:)
fb://publish/mailbox/(initWithFolder:)/(tid:)
fb://publish/privacy
fb://place/create
fb://compose
fb://compose/profile/(initWithUID:)
```

그림 45. 페이스북 어플리케이션에서 확인한 protocol

결론적으로 protocol handler 는 개발자들에게 프로세스간의 통신을 편리하게 하도록 하지만 개발자들은 입력 받은 모든 소스 데이터와 유효성 검사를 실행해야 하고, 민감하거나 위험한 기능을 이용할 때 protocol handler 를 사용하지 않도록 주의해야 할 것이다.

4.3. Locating InSecure Data Storage

모바일기기에 저장 되어진 데이터의 보호는 모바일 어플리케이션 개발자들이 처리해야 하는 가장 중요한 이슈들 중의 하나일 것이다. client-side 에 저장되어진 데이터의 적절한 보안을 취하는 것은 필수적일 것이다.

앞에서도 언급했듯이, 모바일기기에서 민감한 콘텐츠의 보호를 하기 위해서 개발자들은 Data Protection API 를 사용해야만 한다. 불행히도, 다국적 기업의 어플리케이션조차 그들의 민감한 데이터를 관행적으로 일반 텍스트로 저장했다.

이러한 대표적인 사례는 2010 년에 논란이 되었던 AppStore 에서 다운로드 받은 Citigroup 온라인 뱅킹 어플리케이션이 있었고, 다음과 같은 글이 올라왔다.

iOS Application (In)Security 번역

“편지에서, US 의 거대은행인 Citi Monile App 은 사용자의 개인정보를 숨김 파일로 저장을 해서 공격자가 숨겨진 파일을 이용해서 온라인 계좌를 사용할 수도 있다. 개인정보는 계좌번호, 청구서지불과 개인보안코드를 포함하고 있다...[16]”

본 지면은 app 상의 데이터 저장에 초점을 맞추고, 어떻게 어플리케이션들이 Data Protection API 를 사용할 수 있는지, Jean-Baptiste Bedrune 과 ESEC 의 Jean Sigwal [17]는 iPhone 의 암호화에 대한 체계적인 발표가 이루어졌다.

Client-side 데이터는 다양한 형태로 저장될 수 있고, 아래의 것들을 포함하지만 이에 국한되지는 않는다.

- Custom created files
- Databases.
- System logs
- Cookie stores.
- Plists
- Data caches

이것들 모두는 중요한 데이터를 저장하고 있고 만약 모바일기기를 분실했거나, 도난 당했을 경우 보호 되어져야만 한다. 이 데이터들은 보통 어플리케이션의 sandbox container 에 저장된다.

어플리케이션들은 mobile 이라는 file-system 에 저장되고 /var/mobile/Applications 디렉터리에 고유한 GUID 는 app 데이터를 저장하기 위해서 sub direcotry container 로 사용된다.

어플리케이션의 디렉터리 구조는 다음과 같다.

Directory	Description
Application.app	Stores the static content of the application and compiled app. This content is signed and checked at runtime.
Documents	A persistent store for application data; this data will be synched and backed up to iTunes.
Library	This folder contains support data used by the app such as configurations, preferences, cache data and cookies.
tmp	This folder is used to store temporary files.

표 2. 어플리케이션 디렉터리 구조

iOS Application (In)Security 번역

공격자는 어플리케이션 데이터를 이와 같은 디렉터리 구조나 혹은 다른 구조들에 의해서 데이터를 추출할 가능성이 있다. 하지만 파일시스템을 탐색하기 위해서는 우선 필요한 작업이 기기를 탈옥(jailbroken)시키는 것이다.

실제로 AppStore 에 있는 1 백만 유저의 사용자로부터 4+의 평점과 6405 개의 추천수를 받은 Kik Messenger 라는 소셜네트워킹 어플리케이션을 살펴보자.

이 어플리케이션은 사용자간의 무료 메시지를 보낼 수 있게 해주는데, 이러한 일을 하기 위해선 첫번째로 Kik 계정에 등록 해야 한다. Kik 어플리케이션 디렉터리에 있는 환경설정파일인 plist 파일 즉 "Library/Preferences/com.kik.chat.plist"파일이 어플리케이션에 사용되고, 개인적인 정보 예컨대 "사용자이름", "비밀번호" "e-mail", "주소"와 어플리케이션의 설정정보를 저장하고 있다.

다음과 그림과 같이 저장되어 있다.

The screenshot shows the plist editor with the following data:

Key	Type	Value
WebKitLocalStorageDatabasePathPreferen	String	/var/mobile/Applications/68E38644-9203-488F-A707-A52E23879386/Library/WebKit/LocalStorage
username	String	mdsec
install_date	Date	19 Jan 2012 12:43:24
hasProfilePicture4	Boolean	NO
did_app_crash	Boolean	NO
matchingOptedIn	Boolean	YES
lastFullRosterRequestDate	Date	19 Jan 2012 12:43:24
xmppPassword	String	xxxxxxxxxxxxxx
recanTimestamp	Number	132688673.618672
doAddressBookMatching	Boolean	NO
has_asking_to_rate_app	Boolean	NO
xmppUsername	String	mdsec_g6x
wipeCoreData	Boolean	NO
email	String	xxxxxxxxxxxxxx
serverDeviceToken	String	90ad7f6c-12b5-436d-83ef-81ad78b27f908e6a17c5baff9bdfadb8a225d7e6
networkTimestamp	Number	0
firstName	String	MDSec
WebKitOfflineWebApplicationCacheEnable	Boolean	YES

그림 46. com.kik.chat.plist 파일

plist 파일은 Data Protection API 에 의해 보호되어 있지 않았고, 장치가 켜져 있는 동안 잠금 상태와 상관없이 파일시스템 내에 보호되지 않은 채로 저장되어 있다.

위의 예제는 인증서(credential)와 같은 중요한 정보는 keychain 에 저장되어야지 파일시스템의 plist 파일이 저장하는 전형적인 잘못된 데이터 저장의 예다.

게다가, Kik 메신저는 기기에 SMS Chat 기록이나 연락처 정보 등을 포함해서 다른 정보를 모바일기기에 저장한다. 이러한 데이터들은 sqlite database 에 저장되고, 경로는 "Documents/kik.sqlite"에 있고 역시 암호화 되어 있지 않았다.

Table: ZKIMMESSAGE											New Record	Delete Record
Z_PK	Z_ENT	Z_OPT	ZINTERNALID	ZSTATE	ZSYSTEMSTATE	ZTYPE	ZDRAFTM	ZI_ZLAS	ZB66	ZBODY	ZSTANZAO	
1	1	3	2)	0	16	0	1	1	173423	Welcome to Kik! To find your friends, go to Contacts and	08431543-57	
2	2	3	1)	0	0	0	4		176113	MdSec has been added to your contacts	0855744-79c	
3	3	3	1)	1	0	0	4	2	111983	MdSec has added you as a contact	77a1a207-ab	
4	4	3	2)	0	0	0	4		159175	Test has added you as a contact	a521ef4-d20	
5	5	3	3)	1	16	12	1		511313	Hi this is a test	231a909e-82	
6	6	3	2)	3	0	0	4		172215	Test has been added to your contacts	09d8251-c9	
7	7	3	1)	2	30	0	1		141124	Hello test	4aaa1982-c5	
8	8	3	2)	4	16	12	1	3	175032		3d6144c3-09	

그림 47. sqlite database 에 저장

Kik 어플리케이션이 직면해 있는 딜레마 중 하나는 잠겨있는 상태이건 아니건 실시간으로 수신되는 메시지를 처리해야 한다는 것이다. 만약 어플리케이션이 NSFileProtectionComplete 라는 API 를 적용시킨다면 iPhone 이 잠겨있는 상태일 때는 SQLite db 에 접근할 수 없다.

첫째로 NSFileProtectionCompleteUntilFirstUserAuthentication 라는 상수 설정을 함으로 iPhone 을 잠금 해제 할 때까지 데이터를 암호화함으로써 일부만 완화시킬 수 있는 방법이 있다. 재부팅 후에 데이터는 암호화 될 것이다. 그러나 이러한 기능은 iOS 5 에서만 이용 가능하다.

마찬가지로 Kik 어플리케이션은 사용자에서 사진 같은 첨부파일들을 전송할 수 있고 역시 암호화되어 있지 않고 "Documents/fileAttachments"에 저장되어 있다.

사진 첨부파일을 전송한 그림은 아래와 같다.

```
mbp:Documents $ file fileAttachments/057a8fc9-0daf-4750-b356-5b28755f4ec4
fileAttachments/057a8fc9-0daf-4750-b356-5b28755f4ec4: JPEG image data, JFIF
standard 1.01 mbp:Documents $
```

그림 48. 첨부파일 그림

iOS 는 기기에 저장되어져 있는 그림들에는 데이터보호를 적용하지 않기 때문에 논의할 필요가 있다. 하지만 이 부분은 어플리케이션이 잠재적으로 피해야할 위험이다.(however it is a risk that the application could potentially avoid)

Data Protection API 는 4 단계로 파일시스템을 보호하고 있는데 NSData 혹은 NSFileManager 클래스들에 확장성을 전달해줌으로써 설정 할 수 있다.

가능한 보호의 레벨은 다음과 같다:

iOS Application (In)Security 번역

Level	Description
No Protection	The file is not encrypted on the file-system.
Complete Protection	The file is encrypted on the file-system and inaccessible when the device is locked.
Complete Unless Open	The file is encrypted on the file-system and inaccessible while closed. When a device is unlocked an app can maintain an open handle to the file even after it is subsequently locked, however during this time the file will not be encrypted.
Complete Until First User Authentication	The file is encrypted on the file-system and inaccessible until the device is unlocked for the first time. This helps offer some protection against attacks that require a device reboot.

표 3. 보호 레벨

위의 보호 레벨들 중에서 하나를 적용하기 위해서는 확장된 속성을 해당 클래스에서 전달해줘야만 한다.

NSData	NSFileManager
NSDataWritingFileProtectionNone	NSFileProtectionNone
NSDataWritingFileProtectionComplete	NSFileProtectionComplete
NSDataWritingFileProtectionCompleteUnlessOpen	NSFileProtectionCompleteUnlessOpen
NSDataWritingFileProtectionCompleteUntilFirstUserAuthentication	NSFileProtectionCompleteUntilFirstUserAuthentication

표 4. 확장 속성

예를 들어서 어플리케이션이 필요한 몇몇 데이터를 파일시스템에 저장을 한다면 모바일기기가 잠겨있는 동안에는 파일에 접근(Access)할 필요가 없다, 예를 들어 어플리케이션에서 문서파일을 다운로드하고 나중에 다운로드한 문서를 보는 경우처럼 말이다.

모바일기기가 잠겨 있는 동안에는 어플리케이션은 파일에 접근(Access)하지 않기 때문에 NSDataWritingFileProtectionComplete 나 NSFileProtectionComplete 속성을 설정함으로써 완벽하게 데이터를 보호하는 이점이 있다.

iOS Application (In)Security 번역

```
-(BOOL) getFile
{
    NSString *fileURL = @"http://www.mdsec.co.uk/training/wahh-live.pdf";
    NSURL *url = [NSURL URLWithString:fileURL];
    NSData *urlData = [NSData dataWithContentsOfURL:url];
    if ( urlData )
    {
        NSArray *paths =
        [searchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES)];
        NSString *documentsDirectory = [paths objectAtIndex:0];
        NSString *filePath = [NSString stringWithFormat:@"%s/%s",
        documentsDirectory,@"wahh-live.pdf"];
        NSError *error = nil;
        [urlData writeToFile:filePath options:NSDataWritingFileProtectionComplete
        error:&error];
        return YES;
    }
    return NO;
}
```

그림 49. NSFileProtectionComplete 속성을 설정

이 시나리오에서 모바일기기가 잠금 해제된 상태 동안에만 문서에 접근할 수 있다. 잠긴 모바일기기와 이용 불가능한 파일 사이에 OS는 10초간 윈도우를 보여준다.

다음은 모바일기기가 잠겨 있는 동안 파일에 접근을 시도한 것이다:

```
mdsec-iPhone:/var/mobile/Applications/7F5ED565-781E-47FD-8787-4C76CD7A4DD5 root#
ls -al Documents/ total 372
drwxr-xr-x  2 mobile mobile   102 Jan 20 15:24 ./
drwxr-xr-x  6 mobile mobile  204 Jan 20 15:23 ../
-rw-r--r--  1 mobile mobile 379851 Jan 20 15:24 wahh-live.pdf
mdsec-iPhone:/var/mobile/Applications/7F5ED565-781E-47FD-8787-4C76CD7A4DD5 root#
strings Documents/wahh-live.pdf
strings: can't open file: Documents/wahh-live.pdf (Operation not permitted)
mdsec-iPhone:/var/mobile/Applications/7F5ED565-781E-47FD-8787-4C76CD7A4DD5 root#
```

그림 50. 잠긴 모바일기기에 파일 접근 시도

기기에 적당한 데이터 저장 보호레벨을 적용하고 싶은 개발자들은 파일접근을 위하여 각 개발자들의 요구사항에 적합한 속성들을 전달해주는 등의 방식으로 데이터 보호를 할 수 있다.

결론적으로 iOS의 데이터보호는 많은 부분 개발자들의 손에 달렸다, 또 iOS는 파일시스템에 데이터를 적용할 수 있는 보호레벨의 구성을 위해서 세분화된 컨트롤을 제공하고 있다.

(In conclusion, iOS leaves data protection very much in the developer's hands, providing granular controls to configure the level of protection that can be applied to data written to the filesystem.)

불행히도, 개발자들은 민감한 데이터를 위험에 그대로 방치해두고 있고 이 같은 데이터 보호의 장점을 이용하지 않고 있다.

4.4. Attacking the iOS Keychain

iOS 의 keychain 은 인증서(credentials)와 같은 중요한 데이터에 사용되고, keychain 의 접근 가능한 그룹의 멤버가 아닌 경우 오직 그들 개인의 keychain 아이템들에 접근하도록 app 들을 제한하는 암호화된 container 다.

파일시스템에서의 파일들과 유사하게 보호 레벨(Protection Level)은 Data Protection API 를 사용해서 적용시킬 수 있다.

다음의 테이블은 keychain 아이템들에 대한 사용가능한 보호레벨(Protection Level)들을 설명한 것이다.

Attribute	Description
kSecAttrAccessibleAlways	The keychain item is always accessible.
kSecAttrAccessibleWhenUnlocked	The keychain item is only accessible when the device is unlocked.
kSecAttrAccessibleAfterFirstUnlock	The keychain item is only accessible after the first unlock from boot. This helps offer some protection against attacks that require a device reboot.
kSecAttrAccessibleAlwaysThisDeviceOnly	The keychain item is always accessible but cannot be migrated to other devices.
kSecAttrAccessibleWhenUnlockedThisDeviceOnly	The keychain item is only accessible when the device is unlocked and may not be migrated to other devices.
kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly	The keychain item is accessible after the first unlock from boot and may not be migrated to other devices.

표 5. KeyChain 보호 레벨

iOS Application (In)Security 번역

Keychain 아이템들은 SecItemAdd 또는 적용할 보호레벨(Protection Level)을 정의하기 위해서 위의 나열된 속성들 중의 하나를 사용하는 업데이트된 SecItemUpdate 메서드를 사용함으로써 추가 할 수 있다.

기본적으로 모든 keychain 아이템들은 언제든지 접근 할 수 있고, 다른 기기들로 옮길 수 있는 kSecAttrAccessibleAlways 의 보호레벨(protection level)로 생성된다.

어플리케이션의 keychain 아이템들의 접근은 그들이 허용한 권한(entitlements)에 의해서 제한된다. keychain 은 app 의 provisioning profile 의 "keychain-access-group" 권한(entitlement)안에 저장되어 있는 어플리케이션의 identifier 를 사용한다.

다음의 샘플 provisioning profile 만이 app 의 keychain 에 접근할 수 있다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>application-identifier</key>
  <string>my.company.VulnerableiPhoneApp</string>
  <key>get-task-allow</key>
  <true/>
  <key>keychain-access-group</key>
  <array>
    <string>my.company.VulnerableiPhoneApp</string>
  </array>
</dict>
</plist>
```

그림 51. provisioning profile 만이 app 의 keychain 에 접근

이전에 언급했듯이, app 은 SecItemAdd 메서드를 사용해서 keychain 에 아이템을 추가할 수 있다. 저장하고 싶다면 다음의 예제를 어플리케이션을 보자:

keychain 에 있는 license key 는 모바일기기가 잠겨있지 않을 때에만 아이템에 접근 할 수 있다.

iOS Application (In)Security 번역

```
- (NSMutableDictionary *)getkeychainDict:(NSString *)service {
    return [NSMutableDictionary dictionaryWithObjectsAndKeys:
        (id)kSecClassGenericPassword, (id)kSecClass,
        service, (id)kSecAttrService, service, (id)kSecAttrAccount,
        (id)kSecAttrAccessibleWhenUnlocked, (id)kSecAttrAccessible, nil];
}

- (BOOL) saveLicense:(NSString*) licenseKey {
    static NSString *serviceName = @"my.company.VulnerableiPhoneApp";
    NSMutableDictionary *myDict = [self getkeychainDict:serviceName];
    SecItemDelete((CFDictionaryRef)myDict);
    NSData *licenseData = [licenseKey dataUsingEncoding:NSUTF8StringEncoding];
    [myDict setObject:[NSKeyedArchiver archivedDataWithRootObject:licenseData]
    forKey: (id)kSecValueData];
    OSStatus status = SecItemAdd((CFDictionaryRef)myDict, NULL);
    if (status == errSecSuccess) return YES;
    return NO;
}
```

그림 52. 모바일기기가 잠겨있지 않을 때에만 아이템에 접근

첫째로 어플리케이션은 keychain 에 대한 설정 속성으로 쌍의 key-value dictionary 를 생성했다. 위의 예제에서 어플리케이션은 잠금 해제된 모바일기기에서 잠금 해제된 때에 keychain 아이템에 접근하기 위해서 kSecAttrAccessibleWhenUnlocked 속성 값을 사용했다.

그 후 어플리케이션은 keychain 에 저장하고 싶은 data 값에 kSecValueData 속성을 설정했고, 이 예제에서는 license key data 이고 SecItemAdd 메서드를 사용해서 keychain 아이템에 추가를 했다.

아래의 그림은 간단한 SQLite database keychain 이고, 다른 DB 처럼 질의를 할 수 있다. 예를 들어, keychain 그룹의 목록을 알고 싶다면 다음과 같은 명령을 사용해서 실행시킬 수 있다.

```
mdsec-iPhone:/var/Keychains root# sqlite3 keychain-2.db "select agrp from genp"
apple
apple
apple
ichat
com.apple.apsd
apple
apple
T84QZS65DQ.platformFamily
T84QZS65DQ.platformFamily
apple
apple
my.company.VulnerableiPhoneApp
mdsec-iPhone:/var/Keychains root#
```

그림 53. SQLite database keychain

탈옥(jailbroken)한 모바일기기에서 Data Protection API 에 의해 처리되는 이전의 같은 caveats 환경에 있는 모든 어플리케이션의 keychain 아이템들을 dump 해 낼 수 있다.

관련된 keychain-access-group 들과 보호된 아이템들의 검색을 위한 keychain 서비스의 질의는 어플리케이션을 생성하므로써 달성할 수 있다.[18]

4.5. Conduction Cross-Site Scripting (XSS) through UIWebViews

UIWebView 는 텍스트를 보여주기 위한 iOS 의 렌더링 엔진이다. 또 다음의 포맷들을 포함하고 있고 여러 다양한 파일형태를 제공하고 있다.

- HTML
- PDF
- RTF
- Office Documents (doc, xls, ppt)
- iWork Documents (Pages, Numbers and Keynote)

Web View 는 WebKit 기반으로 제작되었고, Safari, MobileSafari 와 같은 핵심 프레임워크를 사용한다. 따라서 Web view 역시 웹브라우저 처럼 remote content 를 가져오는데 사용 되어질 수 있다.

웹 브라우저처럼 web view 들은 어플리케이션에 동적, client-side scripting 을 할 수 있도록 JavaScript 를 지원하지만, API 를 이용해서 이와 같은 설정을 사용할 수 없도록 하는 선택기능은 제공되지 않는다.

때문에, 보통의 웹 어플리케이션처럼 iOS 어플리케이션 역시 Cross-Site Scripting(XSS)의 영향을 받을 수 있다.

iOS Application (In)Security 번역

iOS 에서의 Cross-Site Scripting 은 보통의 XSS 공격보다도 더 까다롭다. Session 을 탈취할 때 개발자는 JavaScript 를 Object-C brige 로 구현을 함으로써 iOS 에서 작동하도록 한다.(as developers commonly expose native iOS functionality by implementing a JavaScript to Objective-C bridge;)

MDSec 는 JavaScript 로 사진을 찍거나, 위치정보에 접근하거나, SMS/E-mail 을 보내는 등의 몇몇 사례를 경험했다. Cross-Site Scripting 는 iOS 의 사용자 입력 값이 검사 없이(without sanitisation) 맹목적으로 채워져 UIWebView 에게 전달된다면 어떠한 상황에서도 발생할 수 있다.(Cross-Site Scripting can occur in an iOS in any scenario where user supplied input is blindly populated in to a UIWebView without sanitisation.)

때때로 web view 에서 개발자가 Object-C 변수 제어에 사용할 때 발생할 수 있다. iOS 어플리케이션인 Skype 도 수신전화에서 "전체이름"을 보여주는 과정에서 이러한 취약점의 양향을 받았다.

Skype 어플리케이션은 수신전화에 "전체이름"을 보여주는 과정에 어플리케이션 안에 저장되어 있는 HTML Template 파일을 검사 없이(without sanitising) UIWebView 를 사용 했다.

이 사례에서 공격자는 파일이 어플리케이션 안에서 로딩 되기 때문에 로컬 파일시스템에 접근할 수 있었다. 이와 같은 결함 개념의 증명으로써 모바일기기의 주소록을 검색하고, 업로드 하도록 개발되었다.[19] (a proof of concept exploit was developed to retrieve and upload the device's address book [19].)

다음의 예제는 object-c 의 변수인 사용자이름에 UIWebView 의 DOM 이 더해지는 코드다.

```
NSString *javascript = [[NSString alloc] initWithFormat:@"var myvar=\"%@\";",
username];
[mywebView stringByEvaluatingJavaScriptFromString:javascript];

[mywebView loadRequest:[NSURLRequest requestWithURL:[NSURL
fileURLWithPath:[NSBundle mainBundle] pathForResource:@"index"
ofType:@"html"]isDirectory:NO]];
```

그림 54. UIWebView DOM

첫째로, javascript 라는 객체에 사용자의 이름이 더해지고, stringByEvaluatingJavaScriptFromString 메서드를 사용함 web view 의 DOM 에 사용자 이름을 저장한 javascript 객체가 더해진다.

JavaScript 는 직접 UIWebView 에 의해서 평가 되기 때문에 바로 이 지점에서 Cross-Site Scripting 이 발생하고 변수 역시 번들 디렉토리에 저장되어 있는 HTML 파일이 채워진다.

(Whilst there is also Cross-Site Scripting at this point as the JavaScript is directly evaluated by the UIWebView, the variable is also populated into the local HTML file stored in the bundle directory)

```

<html>
  <p>
    Cross-Site Scripting in UIWebView:
  </p>
  <p>
    This is an example of XSS:
    <script>document.write(myvar) ;</script>
  </p>
</html>

```

그림 55. XSS 예제

4.6. Attacking XML Processors

XML 은 데이터 표현을 위해서 모바일 어플리케이션 개발에 널리 사용되고 iPhone SDK 는 XML 을 파싱하기 위해서 NSXMLParser, libxml2 두개의 옵션을 제공하고 있다. 하지만 XML 구현을 위한 다양한 유명한 third party XML 파서들이 존재한다.

종종 XML 파서와 관련된 일반적인 공격은 “billion laughs” [20] 공격으로써 파서는 확장 서비스 거부를 유발시킬 수 있는 중첩되는 많은 수의 엔티리를 제공한다.

iOS SDK 에 포함 되어져 있는 기본적인 파서는 취약하지 않다. 왜냐하면 NSXMLParser 는 중첩된 엔티리가 감지되면 NSXMLParserEntityRefLoopError exception 을 일으킨다. 또 libxml2 파서는 “엔티티 참조 루프 감지”를 알리는 오류를 발생시킨다.

XML 파서의 또 다른 공격 시나리오는 외부 XML 엔티티의 파싱이다. NSXMLParser 는 기본적으로 외부 XML 엔티티들의 파싱을 허용하지 않는다. 개발자가 대안으로 LibXML2 파서를 사용하면 외부 XML 엔티티들의 파싱을 기본으로 사용 할 수 있다.

NSXMLParser 를 이용해서 외부 엔티티들의 파싱을 사용 가능하게 하기 위해서 개발자들은 엔티티를 발견했을 때 foundExternalEntityDeclarationWithName 라는 delegate 메서드를 호출해주는 setShouldResolveExternalEntities 옵션 설정을 해줘야 한다.

간단히, 취약한 NSXMLParser 의 구현은 다음과 같다:

iOS Application (In)Security 번역

```
#import "XMLParser.h"
@implementation XMLParser
- (void)parseXMLStr:(NSString *)xmlStr {
    BOOL success;
    NSData *xmlData = [xmlStr dataUsingEncoding:NSUTF8StringEncoding];
    NSXMLParser *addressParser = [[NSXMLParser alloc] initWithData:xmlData];
    [addressParser setDelegate:self];
    [addressParser setShouldResolveExternalEntities:YES];
    success = [addressParser parse];
}

- (void)parser:(NSXMLParser *)parser didStartElement:(NSString *)elementName
namespaceURI:(NSString *)namespaceURI qualifiedName:(NSString *)qName
attributes:(NSDictionary *)attributeDict {}

- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string {}
- (void)parser:foundExternalEntityDeclarationWithName:publicID:systemID {}

- (void)parser:(NSXMLParser *)parser parseErrorOccurred:(NSError *)parseError{
    NSLog(@"Error %i, Description: %@", [parseError code],
        [[parser parseError] localizedDescription]); }
@end
```

그림 56. 취약한 NSXMLParser 의 구현

개발자는 setShouldResolveExternalEntities 파서 옵션의 설정을 함으로써 엔티티의 처리를 위해서 함수 구현이 호출 되었을 때 파서는 외부 엔티티들의 파싱을 사용 가능하게 할 수 있다.

```
NSString *xmlStr = @"<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>\n\n<!DOCTYPE foo [\n\n    <!ELEMENT foo ANY > \n\n    <!ENTITY xxe SYSTEM \"http://192.168.0.7/hello\"> \n\n]> \n\n<foo>&xxe;</foo>";

XMLParser *xp = [[XMLParser alloc] init];
[xp parseXMLStr:xmlStr];
```

그림 57. setShouldResolveExternalEntities 파서 옵션의 설정

파서가 엔티티의 처리를 할 때, 모바일기기에서 웹 서버로 강제로 HTTP 요청을 할 것이다.

iOS Application (In)Security 번역

```
bash-3.2# nc -lvp 80
listening on [any] 80 ...
192.168.0.2: inverse host lookup failed: Unknown host connect to [192.168.0.7]
from (UNKNOWN) [192.168.0.2] 49287 GET /hello HTTP/1.0

Host: 192.168.0.7
Accept-Encoding: gzip
```

그림 58. 웹 서버로 강제로 HTTP 요청

libxml2 파서를 이용한 동일한 공격 벡터의 구현은 다음과 같다:

```
#import <libxml/xmlmemory.h>
@implementation LibXml2
-(BOOL) parser:(NSString *)xml {
    xmlDocPtr doc = xmlParseMemory([xml UTF8String], [xml
lengthOfBytesUsingEncoding:NSUTF8StringEncoding]);

    xmlNodePtr root = xmlDocGetRootElement(doc);
}
@end
```

그림 59. 공격 벡터의 구현

이 예제에서, XML string 에 "xmlParseMemory"메서드가 호출되었을 때 외부 엔티티가 파싱 되고 모바일 기기로부터 HTTP 외부 접속의 결과를 얻을 수 있다.

다른 환경에서, 개발자들은 sandbox 의 제약 하에 존재하는 "file://" 이라는 protocol handler 를 사용해서 로컬 파일을 열수 있다는 것을 알아야만 한다.

4.7. SQL Injection

iOS 어플리케이션들은 보통 일부 어플리케이션의 데이터를 client-side 에 저장해야 한다. 데이터 저장에 가장 간단한 방법은 SQLite data store 를 사용하는 것이다. SQL 을 웹 어플리케이션에 사용할 때, 만약 질의 형식이 정확하지 않을 경우에 SQL Injection 을 초래 할 수 있다.

대부분의 환경에서 client-side 에 데이터 저장하는 것의 영향은 적지만, 서버에서 악의적인 사용자에게 의해서 신뢰할 수 없는 데이터가 검색 될 경우 악용될 수 있는 환경을 제공한다.

client-side SQLite database 에서 데이터 접속을 하기 위해서 iOS 에서는 내장된 SQLite 데이터 라이브러리를 제공한다.

SQLite 를 사용할 때, 어플리케이션은 "libsqlite3.dylib" 라이브러리에 연결된다. 전통적인 웹 어플리케이션과 유사하게, iOS 어플리케이션 SQL Injection 은 사용자의 검증 받지 않은 정보 (unsanitised)의 입력으로 dynamic SQL statement 를 구성할 때 발생한다.

iOS Application (In)Security 번역

SQL statement 를 컴파일 하기 위해서는, statement 는 첫째로 고정된 문자배열로 정의되어야 하고, SQLite 에 준비되어 있는 메서드 중 하나에 전달되어야 한다.

다음의 소셜네트워킹 어플리케이션 예제에서 여러 사용자들의 상태 메시지를 읽고, SQLite 데이터베이스에서 오프라인 보기의 결과를 저장한 결과를 보자. 어플리케이션은 여러 사용자들의 feed 들과 사용자의 개인 프로필 링크를 렌더링하고 그들의 이름을 보여준다.

다음의 예제코드는 사용자의 메시지 feed 를 읽었을 때 동적으로 생성되어진 SQLite statement 를 보여준다.

```
sqlite3 *database;
sqlite3_stmt *statement;
if(sqlite3_open([databasePath UTF8String], &database) == SQLITE_OK)
{
    NSString *sql = [NSString stringWithFormat:@"INSERT INTO messages VALUES('1', '%@','%@','%@')", msg, user, displayname];
    const char *insert_stmt = [sql UTF8String];
    sqlite3_prepare_v2(database, insert_stmt, -1, &statement, NULL);
    if (sqlite3_step(statement) == SQLITE_DONE)
```

그림 60. SQLite statement

위의 코드에서 개발자는 먼저 databasePath 변수에 저장되어 있는 SQLite DB 를 연다. 데이터베이스가 성공적으로 열렸다면, NSString 객체는 "msg", "user", "displayname"등의 변수를 공격자-제어, 검증 받지 않은 정보(unsanitised)를 사용해서 dynamic SQL statement 를 생성한다. SQL 질의는 고정적인 문자 배열로 변환되고, "sqlite3_prepare_v2" 메서드를 사용해서 SQL statement 로 컴파일 된다.

마지막으로, SQL statement 는 "sqlite3_step" 메서드를 사용해서 실행된다. 사용자로부터 만들어진 statement 에서 사용된 파라미터에서 결과 statement 는 사용자가 제어할 수 있다.

예를 들어서, 다음과 같은 악의적인 사용자가 상태 메시지를 고려해보자:

```
Check out my cool site http://mdattacker.net', 'Goodguy', 'Good guy');/*
```

그림 61. 상태 메시지 고려

이것의 결과는 다음의 SQL 쿼리가 실행한다.

```
INSERT INTO messages VALUES('1', 'Check out my cool site http://mdattacker.net', 'Goodguy', 'Good guy');/*','originaluser','Original User');
```

그림 62. SQL 쿼리 실행 예제

iOS Application (In)Security 번역

따라서 공격자는 쿼리에서 필드를 그 다음의 필드를 컨트롤 할 수 있고, 다른 사용자로부터의 메시지를 보여줄 수 있다.

해결책은 전통적인 SQL Injection 방지와 유사하다. 쿼리구조는 바인드 변수와 파라미터 쿼리가 정의 되어야 한다. SQLite 는 준비된 statement 들의 텍스트 값의 바인딩을 위해서 sqlite3_bind_text 기능을 제공한다.

이전의 예제는 다음과 같이 해결 할 수 있다:

```
const char *insert_stmt = "INSERT INTO messages VALUES('1', ?, ?, ?)";
sqlite3_prepare_v2(database, insert_stmt, -1, &statement, NULL);
sqlite3_bind_text(&insert_stmt, 1, [msg UTF8String], -1, SQLITE_TRANSIENT);
sqlite3_bind_text(&insert_stmt, 2, [user UTF8String], -1, SQLITE_TRANSIENT);
sqlite3_bind_text(&insert_stmt, 3, [displayname UTF8String], -1, SQLITE_TRANSIENT);
if (sqlite3_step(statement) == SQLITE_DONE)
```

그림 63. sqlite3 에서 SQL Injection 해결 방안

파라미터 쿼리를 사용하면, 컴파일 된 statement 에 msg 변수는 바인드 된 변수 값에 바인드 될 수 있고, escaped 될 수 없다.

4.8. Filesystem Interacion

iOS 에서 파일시스템의 상호작용은 NSFileManager 나 NSFileHandle 클래스들을 이용해서 달성할 수 있다. NSFileManager 클래스는 명시적으로 파일시스템에 사용되고, NSFileHandle 클래스 역시 소켓이나, 파이프, 모바일기기에 접근을 허용하고 있다.

NSFileManager 클래스는 파일운용을 위해서 많은 강력한 파일시스템 상호작용 인스턴스 메서드를 포함하는데 그것들은 다음의 것들을 포함한다.

Instance Methods	Description
<code>fileExistsAtPath</code>	Determines if a file exists.
<code>contentsEqualAtPath</code>	Compares the contents of two files.
<code>isReadableFileAtPath</code> , <code>isWritableFileAtPath</code> , <code>isExecutableFileAtPath</code> , <code>isDeletableFileAtPath</code>	Determines if a file is readable, writeable, executable or deletable.
<code>moveItemAtPath</code>	Renames the specified file.
<code>copyItemAtPath</code>	Copies a file to the specified destination.
<code>removeItemAtPath</code>	Deletes the specified file.
<code>createSymbolicLinkAtPath</code>	Creates a symbolic link to the specified file.

표 6. 파일시스템 상호작용 인스턴스 메서드를 포함

NSFileHandle 클래스는 file descriptor 와의 상호작용으로 더욱 향상된 방법 등을 제공한다. 이 클래스는 전통적인 C file operation 에 더 가깝고, 파일 내에서 offset 을 탐색하는 방법을 제공하고 개발자들이 handle 을 닫아야 하는 등의 문제를 해결했다. NSFileManager, NSFileHandle 클래스 둘 다 공격자가 파일이름의 일부를 제어할 수 있는 디렉터리 탐색 문제들의 영향을 받을 수 있다.

파일의 내용을 읽을 수 있는 두 개의 클래스의 구현을 살펴보면:

```
- (NSData*) readContents:(NSString*) location
{
    NSFileManager *filemgr;
    NSData *buffer;
    filemgr = [NSFileManager defaultManager];
    buffer = [filemgr contentsAtPath:location];
    return buffer;
}
```

그림 64. 파일 매니저 구현

```
- (NSData*) readContentsFH:(NSString*)location
{
    NSFileHandle *file;
    NSData *buffer;

    file = [NSFileHandle fileHandleForReadingAtPath:location];
    buffer = [file readDataToEndOfFile];
    [file closeFile];
    return buffer;
}
```

그림 65. 파일 매니저 구현

위에 메서드들에서 개발자 파일을 열기 전에 예를 들어서 "../.."와 같은 전통적인 경로문자의 취약점을 검사하도록 만들지 않았다.

```
NSString *fname = @"../Documents/secret.txt";
NSString *sourcePath = [[NSString alloc] initWithFormat:@"%s/%s", [[NSBundle mainBundle] resourcePath], fname];
NSLog(@"##### PATH = %s", sourcePath);
NSString *contents = [[NSString alloc] initWithData:[fm readContentsFH:sourcePath] encoding:NSUTF8StringEncoding];
NSLog(@"##### File contents: %s", contents);
```

그림 66. 파일 시스템 접근 취약점 검사

위의 예제에서 fname 변수 값은 사용자 제어 문자로써 resource bundle 디렉터리의 밖의 Documents 디렉터리의 공격이 가능하다. 개발자들은 Object-C 와 C 를 혼합해서 작업할 때 특히 Object-C 를 이용해서 file 작업을 할 때 위험하다는 것을 인식 해야 한다. Object-C 에서 NSString 객체에서 string 객체의 끝에 NULL Byte 를 사용하지 않는다.

만약 개발자가 NSString 객체를 사용자 제어 파일경로를 사용하고, 나중에 C 에서 file operation 작업을 하게 되면 공격자는 다음의 예제처럼 string 객체를 일찍 단축(terminate)시킬 수 있다.

```
NSString *fname = @"../Documents/secret.txt\0";
NSString *sourcePath = [[NSString alloc] initWithFormat:@"%s/%s.jpg", [[NSBundle mainBundle] resourcePath], fname];
char line[1024];
FILE *fp = fopen([sourcePath UTF8String], "r");
fread(line, sizeof(line), 1024, fp);
NSString *contents = [[NSString alloc] initWithCString:line];
fclose(fp);
```

그림 67. string 객체를 일찍 단축(terminate)

위의 예제에서 개발자는 JPG 파일의 위치를 문자로 정의하고, sourcePath 변수에 확장자는 jpg 를 직접 정의했다. 하지만 fname 변수에 정의 되어 있는 NULL Byte 때문에 C String 으로 변환 시에 string 이 일찍 단축(terminate)됨으로써 공격자는 어떠한 파일 타입도 열수 있게 되었다.

4.9. GEO-Location

Apple 은 Core Location Framework 를 이용해서 모바일기기의 geo-location 기능에 접근할 수 있도록 지원한다. 모바일기기의 좌표는 GPS, cell tower 의 측정 혹은 WiFi 네트워크의 근접도를 이용해서 결정된다.

geo-location 데이터를 이용할 때, 개발자가 고려해야만 하는 중요한 두 가지가 있는데 첫째는 어떻게, 그리고 어디에 데이터가 기록되는지 그리고 요청되어진 좌표의 정확성이다.

Core Location 은 구동 이벤트로써 어플리케이션은 수신된 정보를 찾고, 갱신된 정보를 수신하기 위해서 등록을 해야만 한다.

(Core Location is event driven and an app looking to receive location information must register to receive event updates.)

어플리케이션에서 사용할 수 있도록 Event update 는 위도와 경도좌표를 제공해준다.

어플리케이션을 평가할 때 어떻게 좌표 데이터가 저장되는지가 가장 중요하게 고려 되어야 한다.

만약 어플리케이션이 client-side 에 좌표 데이터를 저장한다면, 개발자는 이 데이터를 이전에 언급했던 세부적인 방법 중의 하나의 방법을 사용해서 보호해야 한다.

하지만 사용자의 움직임을 추적하는 행위를 피하기 위해서 보통 모바일기기에 위치 정보를 저장하지 말 것을 추천한다.

게다가 client-side 에 기록하기 위해서, 만약 어플리케이션이 서버에 좌표 정보를 전달해주는 경우, 개발자는 정보가 기록되었는지, 그리고 익명으로 되었는지 반드시 확인해야만 한다.

Event update 시에 개발자도 고려해야 할 또 다른 사항은 그들이 요청한 정보의 정확성이다.

예를 들어서, 어플리케이션이 위성 네비게이션 정보를 사용했다면, 매우 정확한 위치 정보를 요구하게 될 것이다. 반면에 가장 가까운 식당 등의 정보를 제공하는 어플리케이션이라면 다소 부정확해도 상관은 없다.

위치 기록과 마찬가지로, 개발자에 의해서 iOS 어플리케이션을 개발할 때 좌표의 정확성은 개인정보침해의 우려가 있기 때문에 고려되어야만 하는 사항이다.

CLLocationManager 를 이용할 때, 어플리케이션은 다음의 상수들을 제공하는 CLLocationAccuracy 클래스를 사용해서 정확한 정보를 요청 할 수 있다.

- kCLLocationAccuracyBestForNavigation

- kCLLocationAccuracyBest
- kCLLocationAccuracyNearestTenMeters
- kCLLocationAccuracyHundredMeters
- kCLLocationAccuracyKilometer
- kCLLocationAccuracyThreeKilometers

4.10. Logging

Logging 은 개발과정 동안 디버깅을 위한 귀중한 리소스를 제공한다. 하지만 몇몇 사례에서 모바일기기가 다음번 부팅 시까지 저장되는 민감하거나, 중요한 정보들이 검출되었다. Object-C 에서의 Logging 은 메시지가 Apple System Log 에 보내지는 NSLog 메서드를 이용해서 이루어진다.

이러한 Console log 들은 XCode Organiser 어플리케이션뿐만 아니라 모바일기기에 저장되어 있는 어떠한 어플리케이션에서도 ASL Library 를 이용해서 접근할 수 있다.

몇몇 사례에서 탈옥된 모바일기기들이 NSLog 의 출력이 syslog 로 돌려지는(redirected)경우가 있었다. 이 사례에서 민감한 정보는 파일시스템의 syslog 파일에 저장이 된다. 따라서 개발자들은 민감하거나 정보나 등록 정보등을 사용할 때 NSLog 의 사용을 피해야 할 것이다.

개발자들이 제품 개발 시에 NSLog 사용하지 않고 컴파일 하는 가장 간단한 방법은 #define NSLog(...)와 같은 dummy 전 처리 매크로를 사용해서 재정의 하는 것이다.

4.11. Backgrounding

만약에 어플리케이션이 실행되었으면, 사용자가 Home 버튼을 눌렀거나 전화가 오는 등 상태변화로 인해서 실행중인 어플리케이션을 Background 로 보낼 수 있다.

어플리케이션이 background 에서 정지해 있는 동안 iOS 는 "snapshot"을 찍을 것이고 이것은 어플리케이션 내부의 cache directory 에 저장된다.

어플리케이션이 다시 열리는 모바일기기는 인터페이스를 구성하기 위해서 Screenshot 을 사용하게 되고 어플리케이션은 즉시 로딩이 되게 되는데 이것은 실제 어플리케이션을 다시 로딩 하는 시간보다는 적은 시간이 걸리기 때문에 유용하게 사용될 수 있다.

만약에 민감한 정보가 열린 채로 background 로 보내어지면, 파일시스템에 snapshot 이 명확하게 찍혀 저장된다 하지만 UIApplication delegate 메서드인

iOS Application (In)Security 번역

applicationDidEnterBackground 는 어플리케이션이 background 로 보내지거나, 디스플레이가 수정되는지 적절하게 감지해 낼 수 있다.

예를 들어서 민감한 정보가 저장되어 있는 Field 가 있다고 하면, 어플리케이션은 "Hidden"속성을 사용함으로써 이와 같은 정보를 감출 수 있다.

```
- (void)applicationDidEnterBackground:(UIApplication *)application {  
    viewController.creditcardNumber.hidden = YES;  
}
```

그림 68. "Hidden"속성을 사용

반대로 어플리케이션이 재 시작할 때, 아래와 같이 applicationDidBecomeActive delegate 를 사용해서 이것들이 숨겨져 있는 것을 다시 보이게 할 수 있다.

```
- (void)applicationDidBecomeActive:(UIApplication *)application {  
    viewController.creditcardNumber.hidden = NO;  
}
```

그림 69. applicationDidBecomeActive delegate 를 사용

5. 대응 방안

iOS 어플리케이션은 일반적으로 buffer overflow 와 같은 메모리 손상 문제들에 대해서는 탄력적이다. 개발자가 Object-C 메모리 할당하는 작업을 할 때 개발자는 버퍼의 사이즈를 고정적으로 설정할 수 없다.

하지만 이전에도 언급 했듯이 C 와 Object-C 를 혼용해서 사용해서 iOS 어플리케이션을 구현하는 경우는 흔하진 않지만 외부의 라이브러리나 암호화 등의 독립적인 코드의 사용에서는 버퍼 사이즈를 설정하는 경우를 볼 수 있다. 이와 같은 시나리오에서는 전통적인 메모리 취약점을 야기할 수 있다.

하지만 Object-C 에서 적게나마 메모리 손상의 문제와 관련된 일들이 있었는데 다음과 같다.

5.1. Format String

Format string 의 취약점들은 format specifier 를 허용하는 Object-C 메서드의 부적절한 사용으로 야기하는 메모리 취약점 버그의 클래스이다.

취약한 Object-C 메서드들은 다음과 같다.

NSLog

[NSString stringWithFormat]

[NSString stringByAppendingFormat]

[NSString initWithFormat]

[NSMutableString appendFormat]

[UIAlertView alertWithMessageText]

[UIAlertView informativeTextWithFormat]

[NSException format]

[NSMutableString appendFormat]

[NSPredicate predicateWithFormat]

Format string 취약점들은 공격자가 format specifier 의 일부분이나 관련된 전체 메서드를 제공할 수 있을 때 발생한다

예를 들면 다음의 상황을 고려해보자:

iOS Application (In)Security 번역

```
NSString *myURL=@"http://10.0.2.1/test";
NSURLRequest *theRequest = [NSURLRequest requestWithURL:[NSURL
URLWithString:myURL]];
NSURLResponse *resp = nil;
NSError *err = nil;
NSData *response = [NSURLConnection sendSynchronousRequest: theRequest
returningResponse:&resp error: &err];
NSString * theString = [[NSString alloc] initWithData:response
encoding:NSUTF8StringEncoding];
NSLog (theString);
```

그림 70. 취약한 예

예제에서 request 는 10.0.2.1 에서 실행중인 웹서버에 요청하고, 응답은 NSData 객체에 저장되고 NSString 객체로 변환된 후에 NSString 은 NSLog 에 사용되어 로그가 기록된다. 문서화 되어진 NSLog 함수의 사용법에서 NSLog 는 NSLogv 에 쌓여 있고 args 는 인수의 변수 번호이다.(NSLog is a wrapper for NSLogv and args is a variable number of arguments is :)

```
void NSLogv (
    NSString *format,
    va_list args
);
```

그림 71. NSLogv

하지만 이 인스턴스에서, 개발자는 오직 1 개의 인자를 대입하고, 공격자는 특정한 파라미터의 타입을 허용하고 그리고 기록된다. 디버거에서 위의 예제를 실행시켜보면, HTTP 웹서버의 응답을 이용해서 format string 의 취약점이 적용되는 있는지 볼 수 있다.

```
bash-3.2# nc -lvp 80
listening on [any] 80 ...
10.0.2.2: inverse host lookup failed: Unknown host
connect to [10.0.2.1] from (UNKNOWN) [10.0.2.2] 52141
GET /test HTTP/1.1
Host: 10.0.2.1
User-Agent: fmtstrtest (unknown version) CFNetwork/548.0.4 Darwin/11.0.0
Accept: /*/*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Connection: keep-alive

HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 16

aaaa%x%x%x%x%x%x%x
```

그림 72. format string 의 취약점이 적용 여부 확인

iOS Application (In)Security 번역

HTTP 수신 본문은 NSLog 에 기록되고 format string 의 취약점에 노출되기 때문에 stack memory 가 console log 에 dump 된다. 아래의 그림과 같이:

```
(gdb) r
Starting program: /private/var/root/fmtstrtest
objc[8008]: Object 0x11f0b0 of class NSURL autoreleased with no pool in place -
just leaking - break on objc_autoreleaseNoPool() to debug
objc[8008]: Object 0x11e310 of class NSURLRequest autoreleased with no pool in
place - just leaking - break on objc_autoreleaseNoPool() to debug
objc[8008]: Object 0x11f540 of class NSThread autoreleased with no pool in place
- just leaking - break on objc_autoreleaseNoPool() to debug
2012-02-29 17:02:36.304 fmtstrtest[8008:303] aaaa124a600782fe5b84411f0b00
Program exited normally.
(gdb)
```

그림 73. console log 에 dump

전형적인 format string 취약점의 Exploitation 은 공격자가 stack 의 임의의 메모리 주소에서 읽은 "%n" format specifier 를 사용할 수 있도록 함으로써 완성할 수 있다.

그러나, Object-C 에서 Format specifier 는 이용할 수 없다. 대신에 iOS format string 취약점은 Object-C 객체를 정의 하는 "%@" specifier 를 이용하므로써 exploite 할 수 있다.

결과적으로 이것은 임의의 함수포인터가 호출 되도록 할 수 있는 것이다.

다음의 예를 살펴보자([인용[7]] argv[1]의 값을 단순히 NSLog 로 전달 해준 것이다.)

```
int main(int argc, const char* argv[])
{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    NSString *n = [[NSString alloc] initWithCString:argv[1]];
    NSLog(n);
    [pool drain];
    return 0;
}
```

그림 74. 예제

사용자 제어부분의 stack memory 에 접근하기 위해서 충분한 데이터를 popping 하면 우리의 포인터를 역참조(dereferencing)할 때 어떻게 "%@" specifier 가 crash 를 발생시키는지 볼 수 있다.

```
(gdb) r  
bbbbbbbbbbbbbbbbbbxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx@  
Starting program: /private/var/root/fmtstrtest  
bbbbbbbbbbbbbbbbbbxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx  
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx@  
Program received signal EXC_BAD_ACCESS, Could not access memory.  
Reason: KERN_INVALID_ADDRESS at address: 0x62626262  
0x320f8fb6 in ?? ()  
(gdb)
```

그림 75. crash 를 발생 여부 확인

그러나 대부분의 상황에서 Object-C에서는 객체를 저장할 때 Heap을 사용한다. 따라서 실제로 exploitation을 사용하지 않을 수 있다.

더 자세한 format string 취약점 exploitation 은 [7]에서 찾을 수 있다.

5.2. Object Use-after-Free

Object Use-after-Free 취약점은 객체가 해제 된 후에 여전히 존재하는 객체를 참조할 때 발생한다. 만약에 사용 해제된 메모리를 다시 사용하면 공격자는 재사용한 메모리에 대해서 영향을 줄 수 있다. 일부 환경에서 임의의 코드를 실행시킬 수 있는 가능성이 있다.

Object-C 에서의 use-after-free 취약점의 Exploitation 에 대한 심도 깊은 내용은 [21]에 있다.

다음의 예제를 고려해보자:

```
MDSec *mdsec = [[MDSec alloc] init];
[mdsec release];

[mdsec echo: @"MDSec!"];
```

그림 76. use-after-free 취약점 예제

위의 예제에서, MDSec 클래스의 인스턴스를 생성한 뒤에 release 를 사용해서 메모리 해제를 했다. 하지만 객체가 메모리 해제된 후에 이전에 메모리 해제된 포인터에서 echo 메시드가 호출되었다. 이와 같은 인스턴스에서 crash 는 일어나지 않을 것이다.

메모리는 재할당(reallocation)혹은 deconstruction(해체)를 통해서 corrupte 를 일의 키지는 않을 것이다. 하지만 다음과 같이 heap 이 사용자 제어 데이터와 함께 뿌려진 데이터를 고려해보자:

iOS Application (In)Security 번역

```
MDSec *mdsec = [[MDSec alloc] init];  
[mdsec release];  
for(int i=0; i<=50000; i++) {  
    char *buf = strdup(argv[1]);  
}  
[mdsec echo: @"MDSec!"];
```

그림 77. heap 이 사용자 제어 데이터와 함께 뿌려진 데이터

위의 예제를 실행시켜보면 echo 메시지가 호출될 때 이전에 사용 해제된 객체 인스턴스에 의해서 heap memory 의 재사용 때문에 액세스 위반 에러가 발생한다.

```
(gdb) r AAAA  
Starting program: /private/var/root/objuse AAAA  
  
Program received signal EXC_BAD_ACCESS, Could not access memory.  
Reason: KERN_INVALID_ADDRESS at address: 0x41414149  
0x320f8fbc in ?? ()  
(gdb)
```

그림 78. 액세스 위반 에러가 발생

iOS 5.0 릴리즈에서 Automatic Reference Counting (ARC)를 도입했다. (섹션 3.4 참조)

이는 개발자에서 컴파일러로 메모리 관리의 책임을 전가한 것이다.

따라서 ARC 를 사용한 어플리케이션은 use-after-free 이슈들에 대한 많은 수에 대해서 상당히 감소되었을 것이고, 개발자는 더 이상 객체의 retain, 혹은 release 에 대한 책임을 지지 않는다.

6. 결론

결론적으로 모바일 컴퓨팅의 증가 덕분에 모바일 보안이 어느 때 보다 중요해졌다. 개인뿐만 아니라 기업시장에서 모바일 어플리케이션의 사용증가로, 모바일 어플리케이션보안은 점점 주목 받을 것이다.

아마도 모바일 어플리케이션 보안에서 가장 중요한 두 가지 문제는 개인과 기업에 가장 큰 위험을 줄 수 있는 것으로 첫째 어떻게 데이터를 전송 할 것인가, 둘째 어떻게 저장할 것인가이다.

많은 API Specific injection 스타일 공격과 Memory Corruption 결함 등의 공격은 때때로 공격자가 모바일기기의 통신을 조작할 수 있는 서버나 보안에 취약한 전송메커니즘에 기인한다.

보안기능은 플랫폼에서 제공하는 수많은 공격자가 모바일기기에서 성공적으로 Memory corruption 결함 exploit 을 할 수 없도록 장애물들을 추가적으로 제공 해야 한다.

실제로 third party 어플리케이션에서의 memory corruption 결함은 플랫폼에 iOS 에서의 취약점과 결합해서 사용할 수 있는 작은 위험을 야기한다.

앞으로, 모바일 어플리케이션의 취약점들은 가까운 미래에 사라질 것이다. OWASP[22]와 같은 모바일 보안프로젝트는 모바일 보안문제들의 문제점들을 제기하고 모바일 개발 분야에서 희망적으로 도움이 될 것이다.

안전하게 모바일 어플리케이션을 구현하려고 하는 기관들(Organisations)은 개발과정 전반에 보안평가들(security assessments)을 통합해야만 하고 개발자들 그리고 QA 팀이 충분한 보안교육을 받을 수 있도록 해야만 한다.

7. iOS App Compliance Checklist

iOS 어플리케이션의 평가 기간 동안, 때때로 보안평가자 뿐만 아니라 개발자들에게 어플리케이션이 부합해야 한다면지, 가이드 라인을 제시하는 등의 준수사항의 기준이 있으면 유용하다.

아래는 MDSec 가 제공하는 iOS 평가를 위한 평가가이드이다.

검사항목(Issues)	준수사항(Compliance)
Compiler Protection	
Application is compiled with PIE	PASS/FAIL
Application is compiled with stack cookies	PASS/FAIL
Application uses Automatic Reference Counting	PASS/FAIL
Transport Security	
Application rejects self-signed certificates: allowsAnyHTTPSCertificateForHost / continueWithoutCredentialForAuthenticationChallenge	PASS/FAIL
Application rejects expired certificates: kCFStreamSSLAllowsExpiredCertificates	PASS/FAIL
Application validates root certificates: kCFStreamSSLAllowsAnyRoot	PASS/FAIL
Application validates certificate chain: kCFStreamSSLValidatesCertificateChain	PASS/FAIL
Inter Process Communication	
Application validates the source bundle : handleOpenURL	PASS/FAIL
Application validates content of IPC parameters	PASS/FAIL
Data Storage	
Application encrypts data written with NSData	PASS/FAIL
Application encrypts data written with NSFileManager	PASS/FAIL
Keychain	
Keychain items are protected using the Data Protection API : SecItemAdd / SecItemUpdate	PASS/FAIL
UIWebViews	
Application does not load UIWebView from a local resource	PASS/FAIL
Application validates user controlled content populated	PASS/FAIL

iOS Application (In)Security 번역

in to a UIWebView: stringByEvaluatingJavaScriptFromString	
XML Processing	
Application disables external entities with NSXMLParser: setShouldResolveExternalEntities	PASS/FAIL
Application disables external entities with LibXML2	PASS/FAIL
Application builds XML with user controllable strings	PASS/FAIL
SQL	
Application uses parameterized queries for data access : sqlite3_prepare_v2	PASS/FAIL
File System	
Application sanitises path for traversal characters	PASS/FAIL
Application validates NSString paths for null bytes	PASS/FAIL
GeoLocation	
Application uses suitable level of accuracy: CLLocationAccuracy	PASS/FAIL
Application does not log location data client-side	PASS/FAIL
Logging	
NSLog is disabled in production builds	PASS/FAIL
Custom logs contain no sensitive data	PASS/FAIL
Backgrounding	
Application removes sensitive data from view when backgrounded: applicationDidEnterBackground	PASS/FAIL
Memory Corruption	
Application uses correct format specifiers for vulnerable functions	PASS/FAIL
Application does not reference freed objects	PASS/FAIL

표 7. 평가 가이드

8. About MDSec

회사는 2011 년에 설립되었고 전세계의 유명한 기업과 자사의 고객을 기반으로 전분야 걸쳐 폭발 적인 성장을 했다. MDSec 는 당신 조직의 향상된 보안을 제공해줄 수 있습니다. sales@mdsec.co.uk 로 언제든지 연락 주시면 친절한 상담을 해드립니다..

www.boanproject.com

9. Acknowledgements

필자는 이 백서의 개발 및 검토기간 동안에 많은 조언과 권고사항을 제공해준 MDSec 의 Marcus Pinto, Ollie Whitehouse 의 Recx, 그리고 Hubert Seiwert 등 감사드립니다. (이하 생략)

www.boanproject.com

10. References

- [1] Mobile/Tablet Top Operating System Share Trend - NetMarketShare
<http://www.netmarketshare.com/operating-system-marketshare.aspx?qprid=9&qpcustomb=1>
- [2] Apple iOS 4 Security Evaluation – Dino Dai Zovi
https://media.blackhat.com/bh-us-11/DaiZovi/BH_US_11_DaiZovi_iOS_Security_WP.pdf
- [3] iOS Standard Agreement
https://developer.apple.com/programs/terms/ios/standard/ios_standard_agreement_20100909.pdf
- [4] Address Space Layout Randomization
http://en.wikipedia.org/wiki/Address_space_layout_randomization
- [5] Steffan Esser's ASLR Research
http://antid0te.com/CSW2012_StefanEsser_iOS5_An_Exploitation_Nightmare_FINAL.pdf
<http://www.suspekt.org/>
- [6] Apple Sandbox - Dionysus Blazakis
<http://securityevaluators.com/files/papers/apple-sandbox.pdf>
- [7] Auditing iPhone and iPad Applications – Ilja van Sprundel
<http://cansecwest.com/csw11/iPhone%20and%20iPad%20Hacking%20-%20van%20Sprundel.ppt>
- [8] Secure Development on iOS – David Thiel
http://www.isecpartners.com/storage/docs/presentations/iOS_Secure_Development_SOURCE_Boston_2011.pdf
- [9] Crackulous
<http://hackulo.us/wiki/Crackulous>
- [10] Mach-O File Format Reference
<https://developer.apple.com/library/mac/#documentation/developertools/conceptual/MachORuntime/Reference/reference.html>
- [11] Class-Dump-Z
http://code.google.com/p/networkpx/wiki/class_dump_z
- [12] MobileSubstrate
<http://iphonedevwiki.net/index.php/MobileSubstrate>
- [13] Cycrypt: Objective-Javascript
<http://www.cycrypt.org/>
- [14] iSOPenDev
<http://www.iosopendev.com/>
- [15] Insecure Handling of URL Schemes in Apple's iOS – Nitesh Dhanjani
<http://software-security.sans.org/blog/2010/11/08/insecure-handling-urlschemes-apples-ios/>
- [16] Citigroup iPhone Data Storage Issues

http://www.theregister.co.uk/2010/07/27/citi_iphone_app_weakness/

[17] iPhone data protection in depth

<http://esec-lab.sogeti.com/dotclear/public/publications/11-hitbamsterdamiphonedataprotection.pdf>

[18] Keychain Dumper

<https://github.com/ptoomey3/Keychain-Dumper>

[19] Skype iOS XSS

<https://superevr.com/blog/2011/skype-xss-explained/>

[20] Billion Laughs

<http://en.wikipedia.org/wiki/BillionLaughs>

[21] Abusing the Objective-C Runtime

<http://www.phrack.org/issues.html?issue=66&id=4>

[22] OWASP Mobile Security Project

https://www.owasp.org/index.php/OWASP_Mobile_Security_Project

보안 프로젝트(www.boanproject.com)에서는 모바일 서비스, 디지털 포렌식 분석, 최신 이슈 등 기술적 진단에 관한 문서를 중점적으로 번역 진행 중입니다. 열정적이고 관심있는 분들은 언제나 환영합니다.