ViaForensics 42+ Best Practice

= Secure Mobile development for iOS and Android=

(번역 문서)

2012-06-11

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

- 보안프로젝트 번역팀(www.boanproject.com) -

목 차

그림 목차

1. 개요

1.1. 시작하며

viaForensic 에서 우리는 모바일 앱 공격, 해킹, 암호 해제, 결점 발견, 모의침투진단(Pentesting) 그리고 불안전하게 저장된 민감한 데이터를 찾는데 많은 시간을 사용한다. 우리는 회사들이 그들의 앱을 더욱 안전하게 만들 수 있도록 도와주기 위해 이와 같은 일을 한다.

우리의 연구 개발 및 프로젝트 작업에서, 우리는 많은 보안 허점을 발견했다. 어떤 문제들은 커다란가 하면, 다른 것들은 악의적 공격자들을 도와주는 데이터 누설이나 잠재적 벡터를 제공하는 등의 문제도 있다. 우리는 이런 업무를 통해 언제나 보안 문제점을 개선하는데 효과적인 조언들을 제공한다. 본 문서는 우리가 고객 및 파트너와 공유하는 현명한 대처방법의 일부분이다. 우리는 모바일 보안의 상태를 개선하고, 이 정보를 무료로 제공함으로서, 지역 사회에 가치 있는 공헌을 할 수 있으리라 희망한다.

이 보고서에서 공격과 보안 충고에 관한 설명은 철저하고 완벽하지 않으며, 우리 또한 필요에 따라 계속 진화하고 수정해 나갈 것이다. 우리는 업데이트된 추천서를 개방하기 위해 권한은 접어둘 것이다. 만약 질문이나 피드백이 있다면 부디 알려주기 바란다.

1.2. viaForensics 에 관하여

viaForensics 는 혁신적인 디지털 웅변술이자 보안 회사로서, 기업, 로펌 그리고 사법/정부 기관에 서비스를 제공한다. 우리가 초점을 두는 분야는 컴퓨터, 모바일 법의학, 모바일 앱 보안 그리고 선진 기업 보안이다.

1.3. 작가에 관하여

Andrew Hoog is the co-founder of viaForensics and a leading mobile security and forensics researcher. He recently published the book Android Forensics and Mobile Security, and along with Katie Strzempka co-authored iPhone and iOS Forensics.

Jonathan Zdziarski is Sr. Research Scientist, and a pioneer in the field of Apple iOS forensics and security. He has authored numerous books on iPhone development and forensics for O'Reilly Media, most recently Hacking and Securing iOS Applications.

Thomas Cannon is Director of Research and Development for viaForensics. He is a noted Android security and malware expert with extensive experience in risk mitigation, security assessment, digital investigation and secure development.

Jared Carlson is a Senior Engineer at viaForensics focused on mobile devices, developing tools for file system analysis, reverse engineering and other security topics. His experience includes algorithm development for fluid dynamics and defense systems.

Ted Eull is VP of Technology Services for viaForensics, managing delivery of mobile application security assessments, mobile device security audits and other services. He is an experienced IT developer, project manager, and security analyst.

작가에 관한 내용은 번역에서 제외합니다.

2. 모바일 보안 분석

2.1. 모바일 보안의 입문

모바일 보안은 버퍼 매니지먼트, 로컬 암호화 그리고 악성 소프트웨어와 같은 전통적 넓은 고객 어플리케이션와 공통적인 위험와 결합된 넓은 범위의 군중, 빠른 개발, 지속적 네트워크 연결성 등의 웹 보안 등의 많은 문제를 수반한다. 모바일 환경의 독특한 특징 중 하나는 신뢰할 수 없는 출처로 고려되어야 할 미지의 개발자로부터 온 설치된 어플리케이션의 보급이다.

모바일 공격 분석

아래에 설명된 바와 같이, 모바일 공격은 장치 레이어, 네트워크 레이어, 데이터 센터 혹은 이러한 것들의 조합을 포함할 수 있다. 선천적 플랫폼 취약점과 소셜 엔지니어링은 사이버 도둑에게 지속적으로 주요한 기회를 제기하고, 그리하여 사용자 데이터를 보호하려는 사람에게는 중대한 문제가 된다.

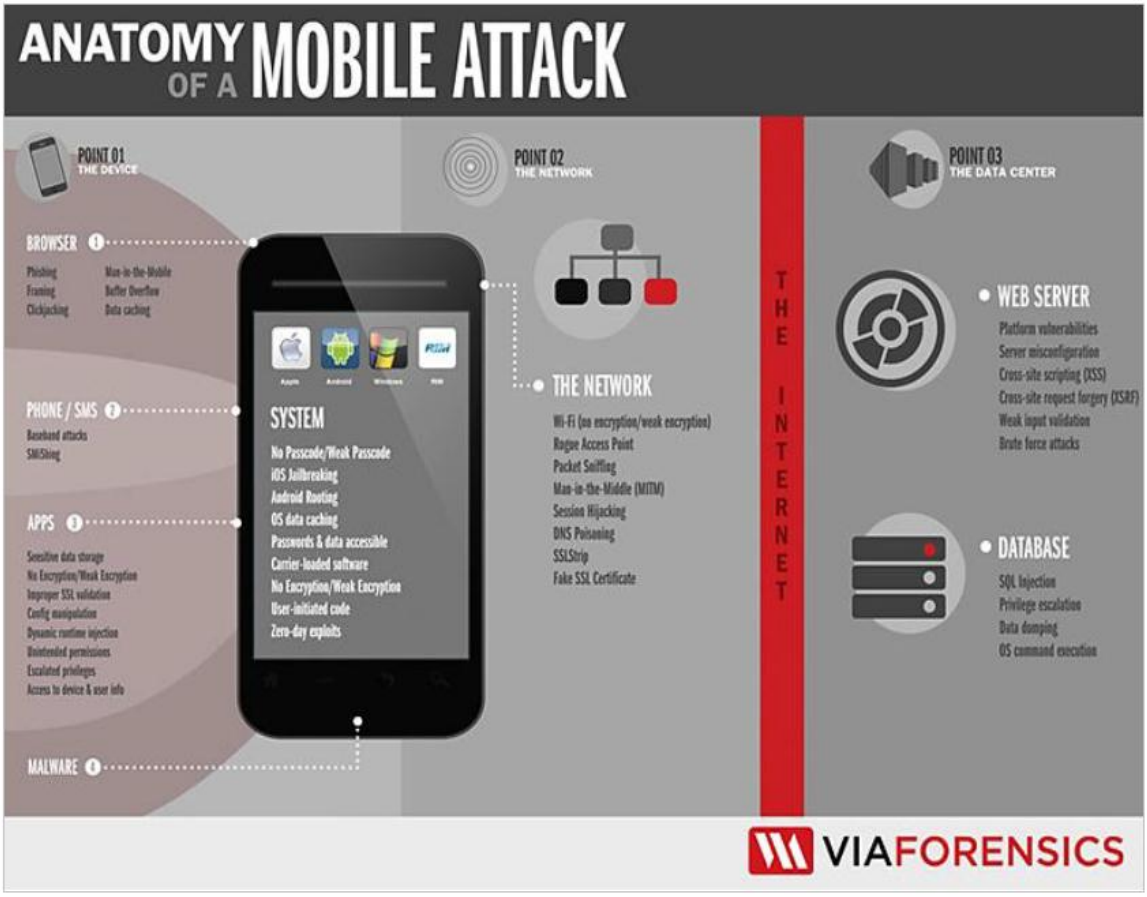

그림 1.. 모바일 공격 분석

공격 벡터

모바일 기술 체인에는 악의적 집단이 악성 공격을 가하기 위해 취약점을 이용할 수 있는 3 가지 부분이 있다.

- 장치
- 네트워크
- 데이터 센터

2.2. 장치

모바일 장치는 민감한 기업 정보(SCI)에 현격한 위험을 제기한다. 주요 위험에는 데이터 손실과 보안성 훼손이 있다. 아이폰, 안드로이드 혹은 다른 스마트폰 중 어느 것이든, 장치 자체를 목표로 하는 공격자들은 다양한 진입 경로를 사용할 수 있다.

- 브라우저, 메일 혹은 다른 미리 실행된 어플리케이션
- 폰/SMS
- 제 3 자 어플리케이션 (앱)
- 운영 시스템
- Baseband, 블루투스 그리고 다른 대역의 무선신호(RF)

브라우저 기반 공격

공격의 브라우저 기반 포인트는 아래를 포함한다:

피싱 - 피싱은 이메일 스푸핑을 통해 신뢰할 수 있는 실체로 가장하여 사용자 이름, 패스워드 그리고 신용 카드 세부 사항등의 개인 정보를 얻는 것을 말한다. 연구에 따르면, 모바일 사용자는 데스크톱 사용자에 비해 3 배나 더 높은 확률로 피싱 웹사이트에 개인 정보를 유출한다고 한다. 이는 부분적으로, 모바일 브라우저가 실행되는 규모가 작아진 환경이 제한된 스크린 공간, 제한된 경고 다이얼로그, 규모가 줄어든 보안 잠금 아이콘으로 인해 작은 일부분의 URL 만을 보여주기 때문인 것이며, 대형 STOP 아이콘, 강조된 주소 바 그리고 다른 시각적 지표 등의 많은 사용자 인터페이스 표시를 축약했기 때문이다.

framing - framing 은 위장 사이트를 활성화시켜 클릭재킹 공격을 유도한뒤 iFrame 을 이용해 의도하지 않은 Web/WAP 으로 인도한다.

클릭재킹 - UI 수정을 통한 일종의 속임수로서, 클릭재킹은 사용자가 보기에 아무렇지 않은 링크나 버튼을 클릭했을 때 사용자를 속여 기밀 정보를 유출하거나 그들의 장치를 통제하는 행위를 말한다. 이 공격은 사용자가 미지 상태에서 실행시키는 임베디드 코드나 스크립트의

형태를 띈다. 클릭재킹은 페이스북을 포함해 사이트 상에서 이용되어 정보를 훔치거나 사용자가 사이트를 공격하게끔 유도한다.

Drive-by Downloading - 특히 안드로이드는 이 공격에 취약한데, 이는 웹사이트에 방문하면 사용자가 모른채 다운로드가 실행되거나 기만적 프롬프트로 사용자를 속인다. 다운로드는 악성 앱일 수도 있고, 그러면 사용자에게 앱을 설치하도록 장치가 자동적으로 재촉할 것이다. 안드로이드 장치가 "출처 미상"에서의 앱을 허락하도록 설정되어 있을 때, 설치가 허용된다.

Man-in-the-Mobile(MitMo) - 악의적 사용자가 신원 확인을 위해 사용자 핸드폰에 SMS 텍스트 메세지를 통해 코드를 전송하는 암호 인증 시스템을 우회하도록 모바일 장치에 있는 악성 소프트웨어를 사용하게 해준다.

폰/SMS 기반 공격

베이스밴드 공격 - 셀 타워와 라디오 신호를 주고 받는 하드웨어인 폰의 GSM/3GPP 베이스밴드 프로세서에서 발견된 취약점을 이용하는 공격이다.

SMiShing - 피싱과 유사하지만, 사용자가 불법 웹사이트를 방문하고 사용자 이름, 패스워드 그리고 신용 카드 번호 등의 민감한 정보에 접근하도록 하기 위해 이메일 메세지 대신에 핸드폰 문자 메세지를 사용한다.

RF 공격 - 블루재킹, NFC 공격 그리고 다른 RF exploit 는 전형적은 근접 장치 대 장치 소통에서 사용되는 다양한 주변 소통 경로 상의 취약점을 찾는다.

어플리케이션 기반 공격

민감한 데이터 저장 - 2011 년 viaForensics 연구는 안전하지 않게 스토어 데이터를 샘플로 한 인기있는 앱들의 83%를 찾았다.

미암호화 / 취약한 암호 - 암호화 되어 있지 않거나 약하게 암호화 되어 있는 데이터 전송을 허용하는 앱은 공격에 취약하다.

부적합한 SSL 유효화 - 앱의 보안 소켓 레이어 (SSL) 유효화 프로세스안의 버그 데이터 보안 위반을 일으킬 수 있다.

Config manipulation(구성 조작) - 관리 인터페이스, 구성 스토어 그리고 클리어 텍스트 구성 데이터 회수에 비인가 접근을 끌어들이는 것을 포함한다.

Dynamic runtime injection - 공격자가 어플리케이션의 런타임을 조작하고 악용하여 보안 잠금, 로직 검사를 피해가고, 어플리케이션의 특권 부분에 접근하며, 심지어는 저장된 메모리에서 데이터를 훔치도록 해준다.

의도하지 않은 허가 - 구성이 잘못 설정된 앱은 의도하지 않은 허가를 부여함으로서 공격자들에게 가끔 문을 열어 줄 수도 있다.

향상된 특권 - 보통 어플리케이션이나 사용자에 의해 보호된 리소스에 접근 권한을 얻기 위해 버그, 디자인 결함 혹은 구성 설정상 실수를 이용한다.

운영 시스템 기반 공격

운영체제에서 기인하는 공격포인트들은 다음과 같다

패스코드 미설정 - 많은 사용자들이 패스코드를 설정하지 않거나 강도가 약한 PIN, 패스코드 또는 패터락을 사용한다.

iOS jailbreaking(탈옥) - "Jailbreaking"은 제작자와 캐리어에 의해 장치에서 비인가 코드가 실행되는 것을 방지하기 위해 마련된 보안 메커니즘을 제거하는 것을 말한다. 한번 이러한 제약이 제거되면 장치는 악성 소프트웨어와 다른 공격들의 관문이 될 수 있다.

안드로이드 루팅 - "Jailbreaking"과 유사하게, 루팅은 안드로이드 사용자들이 시스템 어플리케이션과 세팅을 변경하거나 대체하도록 해주고, 관리자 레벨의 허가가 필요한 특수 앱을 실행하게 해준다. jailbreaking 과 마찬가지로, 민감한 데이터의 노출을 초래할 수 있다.

패스워드와 접근 가능한 데이터 - 애플 라인의 iOS 장치와 같은 장치는 암호화된 패스워드와 데이터 저장을 위해 그들 자신의 암호 메커니즘 안의 취약점을 알아왔다. 이러한 취약점에 관한 지식을 가진 공격자들은 장치의 키체인 암호를 해제하여, 사용자 패스워드, 암호화 키 그리고 다른 사적인 데이터를 노출시킬 수 있다.

Carrier-loaded software - 장치에 미리 설치된 소프트웨어는 보안 결점을 갖고 있을 수 있다. 최근 안드로이드 핸드셋에서 일부 미리 실행된 앱들이 핸드셋을 지우고 데이터를 훔치며, 심지어 통화내용 도청하는데 사용될 수 있는 보안 취약점을 포함하고 있는 것으로 발견됐다.

Zero-day 공격 - 공격은 때로 취약점이 처음으로 이용당했을 때와 소프트웨어 개발자들이 해당 취약점에 대한 이슈를 다루는 발간물/패치가 나오는 사이에 수행되기도 한다

2.3. 네트워크

Wi-Fi (취약한 암호화 설정/ 미암호화) - Wi-Fi 네트워크에서 사용될 때 암호화 실행에 실패한 어플리케이션은 무선 연결에서 도청하는 악의적 공격자에 의해 데이터가 가로채질 수도 있는 위험을 안고있다. 많은 어플리케이션이 SSL/TLS 를 이용하는데, 이는 어느 정도 수위의 보호를 제공한다. 그러나 SSL/TLS 에 대항하는 일부 공격 또한 공격자에게 중요한 사용자 데이터를 노출하는 것으로 증명됐다.

비인가 AP 접속 - 안전한 네트워크에 접근 권한을 집단에게 부여하는 비인가 무선 접근 지점을 물리적으로 설치하는 것을 말한다.

패킷 스니핑 - 악의적 침입자가 전형적으로 클리어 텍스트로 전송된 사용자 이름, 패스워드 정보를 포함하는 네트워크 트래픽을 캡처하고 분석하도록 해준다.

Man-in-the-Middle (MITM) - 존재하는 네트워크 연결에서 도청하여, 연결에 침입, 메세지 가로채기 그리고 지정 데이터 수정을 한다.

SSLStrip - 웹사이트에서 SSL/TLS 실행의 약점을 이용하는 man-in-the-middle 공격의 한 형태로, 사용자가 HTTPS 연결이 현재 존재하고 있음을 증명하는 것에 의존할 수 있다. 공격은 안 보이게 암호화 없이 HTTP 에 대한 연결을 저하시키고, 사용자가 모바일 브라우저에서 탐지하는 것 또한 어렵다.

세션 하이재킹 - 세션 키를 이용하여 사용자와 네트워크 정보에 비인가 접근 권한을 얻는 것을 말한다.

DNS 포이즈닝 - 네트워크 DNS 를 이용하는 것은 웹사이트의 사용자를 공격자가 선택한 다른 사이트로 인도하는데 사용될 수 있다. 어떤 경우에는 공격이 또한 앱을 통해 컨텐트를 주입할 수 있다.

가짜 SSL 증명서 - 또 다른 man-in-the-middle 공격으로 악의적 사용자가 안전해야 할 HTTPS 연결 상의 트래픽을 훔칠 수 있게 해주는 가짜 SSL 증명서를 발행하는 것을 말한다.

2.4. 데이터 센터

데이터 센터를 목표로 하는 공격자들은 2 개의 주요 진입 경로를 사용한다.

- 웹 서버
- 데이터베이스

웹 서버 기반 공격

플랫폼 취약점 - 웹 서버 상에서 실행되는 운영 시스템, 서버 소프트웨어 혹은 어플리케이션 모듈 안의 취약점들은 공격자에 의해 이용될 수 있다. 취약점은 때로 프로토콜 혹은 접근 제어 상의 약점을 찾기 위해 모바일 장치와 웹 서버 사이의 소통을 관찰하는 방법으로 찾아낼 수 있다.

잘못된 서버 구성 설정 - 잘못 설정된 웹 서버는 일반적으로 보호되어야 할 리소스에 비인가 접근을 허용할 수 있다.

교차 사이트 스크립팅 (XSS) - 교차 사이트 스크립팅(Cross-site Scripting) 공격은 웹사이트에 악성 자바 스크립트 코드를 주입하는 것을 말한다. 이러한 공격에 취약한 페이지들은 적절한 처리를 거치지 않은 채 브라우저에 사용자 입력을 되돌려보낸다. 이 공격은 종종 사용자가 어떤 페이지를 방문할 때 자동적으로 코드를 실행하여 사용자의 브라우저를 통제한다. 브라우저의 통제권이 설립되면, 공격자는 이러한 통제를 이용하여 다양한 공격을 가할 수 있는데, 여기에는 컨텐트 주입 혹은 악성 소프트웨어 선전 등이 있다.

교차 사이트 요청 위조 (CSRF) - Cross-site request forgery 는 어떻게 특정 어플리케이션이 작동하는지에 대한 지식에 기반해 HTTP (Web) 요청을 만들고 사용자나 브라우저를 속여 이러한 요청을 제출하도록 하는 것을 말한다. 만약 웹 앱이 취약하다면, 공격은 마치 사용자로부터 온 것처럼 보이는 작업 혹은 제출을 실행에 옮긴다. CSRF 는 일반적으로 XSS, 소셜 엔지니어링 또는 다른 방법들을 통해 공격이 이미 사용자 세션 통제권을 얻은 뒤에 사용된다.

취약한 입력값 검증 - 많은 웹 서비스들은 과도하게 모바일 어플리케이션으로부터 오는 입력을 신뢰하여, 어플리케이션이 end user 에 의해 제공된 데이터를 유효화 하는데 의존한다. 그러나 공격자는 웹 서버로 가는 그들만의 소통을 위조하거나 어플리케이션의 로직 검사를 완전히 피해가서, 서버 상에 유효화 로직이 없다는 점을 이용하여 비인가 행동을 취할 수 있도록 해준다.

Brute-force 공격 - brute-force 공격은 간단히 필드에 유효한 입력을 맞추도록 시도해 보는데, 때론 높은 비율의 시도와 가능한 값의 사전을 사용한다. brute-force 공격은 인증 절차에서 가장 흔히 이뤄지지만, 이는 또한 웹 앱 상에서 다른 유효한 값을 발견하는데 사용될 수도 있다.

데이터 베이스 공격

SQL injection - 사용자 입력을 적절하게 유효화하지 않는 인터페이스는 SQL 이 원래 같았으면 아무런 문제 없었을 어플리케이션 query 에 주입되는 결과를 초래하여, 데이터베이스가 노출되거나 일반적으로 사용자 또는 어플리케이션으로부터 제한되어야 할 데이터가 조작되는 일이 발생할 수 있다.

OS 명령 실행 - SQL 주입과 유사하게, 특정 데이터베이스 시스템은 OS 레벨 명령을 실행하는 방법을 제공한다. 공격자는 이러한 query 에 주입할 수 있는데, 이로서 데이터베이스가 이러한 명령들을 서버 상에서 실행시켜, 공격자에게 추가적인 특권을 제공하는데, 이 특권은 루트 레벨 시스템 권한까지도 포함할 수 있다.

특권 상승 - 더 높은 단계의 접근 권한을 얻기 위해ㅓ 공격이 일부 exploit 을 leverage 할 때 발생한다. 데이터베이스 상에서 이것은 민감한 데이터의 도난으로 이어질 수 있다.

데이터 덤핑 - 공격자는 데이터베이스가 자신 안의 일부 또는 전체 데이터를 없애버려, 민감한 기록을 노출한다.

2.5. 모바일 앱의 종류

모바일 어플리케이션은 전형적으로 아래의 3 가지 작동 카테고리로 나뉘어진다.

웹 - 일반 목적의 웹 브라우저를 통해 작동하는 앱이다. 때로는 WAP 또는 Mobile Sites 라고 가리키기도 하는데, 이들은 몇 십년 전에 온라인 뱅킹과 쇼핑 등 많은 기능을 제공하여 인기를 끌었던 기능적 웹 어플리케이션과 동등한 모바일이다. 비록 정규 웹 사이트는 모바일 웹 브라우저에서 사용될 수 있지만, 많은 회사는 이제 별도의 모바일 웹 앱을 만들어 모바일 속성에 최적화된, 더 작은 스크린 크기, 터치 기반 네비게이션과 GPS 탑재 등의 기능을 더했다.

Native – 특정 모바일 플랫폼에 맞춰서 제작되고 모바일 기기에 맞춰 설치 및 운영될 수 있도록 컴파일된 형태로서, 이것들은 통상 다운로드(예외적으로 임의설치도 가능) 되고, 앱 마켓을 통해 설치된다.

Wrapper – 네이티브앱과 웹앱의 효과와 기능을 혼재한 형태로서 때론 "shell apps" 또는"hybrid apps"라고 불리운다. 엔드 유저에게 네이티브 앱으로 나타나는 동안, 웹 기반 기능은 완전히 네이티브 코드된 앱에서 발견된 것과는 다른 종류의 취약점을 결과로 낳을 수 있다.

*부연설명 : 설치는 네이티브형식인데 취약점은 웹앱 형태의 취약점이 나타날 수 있다.

2.6. NAND 플래시 메모리에 대한 이해

RAM 과 달리 NAND 플래시는 안정적이어서 장치에 전원이 공급되지 않거나 재부팅되어도 데이터가 보존된다. NAND 플래시는 시스템 파일 뿐만 아니라, 디스크에 쓰인 모든 사용자 데이터가 저장된다.

NAND 플래시 메모리는 현대 하드 드라이브의 전자 미디어와는 매우 다른 특징들을 갖고 있다. 이러한 특성들은 보안 개발(그리고 forensic 분석가를 위한 기회을 위한 수많은 어려움이 있는 동시에도 NAND 를 모바일 장치에게 최적의 저장고로 만들어준다.

한 가지 눈에 띄는 문제점은 각 블럭이 한정된 쓰기/삭제 사이클 수명을 갖고 있다는 것이다. NAND 플래시 컨트롤러는 일반적으로 wear leveling 이라는 프로세스 안에서 자주 동일한 블럭을 덮어쓰지 않고, 다수의 경우 강제적인 것이 아니라면 블럭이 너무 빨리 소모되는 것을 막기 위해 블럭 삭제를 피하도록 설계되었다. 그러므로 다수의 경우 이런 장치 상의 데이터는 전통적 하드 디스크보다 더 오랫동안 삭제된 데이터를 유지할 수 있다.

2.7. 암호화에 대한 이해

많은 어플리케이션이 민감한 데이터를 보호하기 위해 암호화를 이용하지만, 여러 가지 기술적 결함으로 인해 공격자가 키를 회수, 추측 또는 운영 시스템에서 키를 추출해낸다. 마스터가 있으면, 민감한 데이터는 공격자에 의해 암호 해제될 수 있고, 혹은 어플리케이션이 공격자가 데이터 암호 해제를 하도록 조작될 수 있어, 공격을 가속화한다.

암호화를 이용하는 어플리케이션은 자주 아래의 결함 중 한 가지 이상을 갖는다:

마스터키 저장하기 - 디스크에서 마스터키를 저장하는 어플리케이션은 만약 자신이 사용자가 제공한 패스프레이즈로 암호화에 실패하면 그들 자신을 공격자에게 노출한다. 만약 키 자체가 가로채졌다면, 사용자가 제공한 패스프레이즈 없이도 모든 민감한 데이터의 암호 해제를 하는데 사용되어, 효과적으로 패스프레이즈를 UI 잠금 전용으로 바꾸게 된다. 이러한 결점은 마스터기가 안전하게 장치에 저장되어 있다는 개발자의 신뢰에 기반한 것이지만, iOS 와 같은 운영 시스템의 많은 보안 메커니즘은 키체인에 저장돼 있는 데이터의 암호를 해제하는 능력과 같이 손상 위험에 대해 알고 있었다.

키 추출 기능 사용의 실패 - 사용자 패스프레이즈로 마스터키를 암호화하는 어플리케이션은 만약 키 도출 기능 사용을 실패하거나 강제될 수 있는 약한 실행을 사용하면, 스스로를 공격에 노출시킨다. 많은 어플리케이션에게 가장 흔한 실천으로는 간단히 MDS 해쉬나 한 가지 종류의 암호화 단일 패스를 사용하여 마스터키를 암호화하는 방법으로, 이는 크래킹 툴이나 레인보우 테이블을 사용하여 마스터키 암호화를 강요하기 쉽게 된다. 키 도출 기능은 대량의 조작 자원이 패스프레이즈로부터 키를 도출해내어 강제 공격을 가하기 어렵게 만들도록 설계되었다.

런타임 공격 - 데이터를 내부적으로 암호화하는, 다시 말해 사용자 패스프레이즈가 암호화에 통합되지 않은 어플리케이션은 잠재적으로 런타임 공격에 노출되어 있는데, 여기서 어플리케이션의 내부 로직은 우회될 수 있어 어플리케이션 스스로가 자동적으로 마스터키를 실행하고 사용자 데이터의 암호를 해제하도록 유도하는데, 심지어 때로는 공격자에게 GUI 에 대한 접근 권한마저 제공한다. 비록 가끔 민감한 사용자 데이터가 이러한 방법대로 저장되었더라도, 서버 증명서나 다른 내부적으로 암호화된 데이터와 같이 내부적으로 사용되는 어플리케이션의 일부에 흔한 경우이다.

2.8. 최적의 실천 방법: 안드로이드와 iOS 앱 보호하기

앱 보안을 달성하기 위해서는, 확실한 묘책이란 없다. 그보다는 데이터와 다양한 공격 방법으로부터 오는 시스템을 보호하도록 설계된 여러 가지의 방어 조치를 포함한 접근법을 선택해야만 한다. 이런 목표를 달성하는 것은 당신의 앱이 시장에 출시되기 전에 미리 조치를 취해야 함을 의미한다.

중요 사항: 본 문서 상의 모든 추천 사항이 모든 앱에 적합한 것은 아니다. 일반적으로 데이터와 보안에 대한 필요가 더 민감한 수록, 이러한 실천 방법들이 더 많은 주의를 필요로 한다.
더 필수적인 조언들이 먼저 게재되었고, 첫 번째 10 개 조언 사항은 모든 개발자들이 준수해야할 매우 기본적인 보안 실천 사항들을 대표한다.

어플리케이션 레이어 안전성 강화하기

어플리케이션 레이어 안전성을 강화하는 것은 여러 가지 실천 방법들을 포함하는데, 크게는 다음과 같다:

- 민감한 데이터를 안전하게 저장하거나, 아예 하지 않는다.
- 인증 절차와 세션을 올바르게 관리한다.
- 보안과 암호화 메커니즘을 올바르게 사용하도록 앱을 설계하고 코딩한다.

- 의도치 않은 정보 누출을 피한다.
- 런타임 조작에 저항한다.
- 코드 난독화와 복잡한 리버스 엔지니어링 방지를 위한 팩킹을 이용한다.

아래 최적의 실천 방법들은 이러한 분야를 더욱 세세하게 설명하고, 일부 경우 코드 예시의 레벨까지 다룬다. 모바일 앱들은 보안 조치를 취할 기회를 제공하지만, 동시에 취약점을 피하기 위해 적절한 언지니어링과 코드를 요구한다. 자세한 내용은 아래를 참조한다.

1. 장치에 민감한 데이터 저장을 피한다.

가능하다면 장치에 민감한 데이터를 저장하지 않는다. 설령 암호화됐더라도 가까이에 저장된 모든 데이터들은 훼손될 수 있기에, 꼭 필요한 때에만 데이터를 저장한다. 많은 어플리케이션의 경우, 간단히 민감한 데이터를 저장하지 않는 것이 기능적으로 그리고 기술적으로 가능하다.

만약 민감한 데이터가 반드시 그 위치에 저장되어야 한다면, 올바르게 실행된 암호화 작업을 거친다. 100% 안전하지 않은 상황에서, 공격에 현저한 복잡성을 추가시킬 수 있다. 마스터키는 더욱 빠른 플랫폼에서조차 강제적 시도들이 매우 시간 소모적이게끔 CPU 소비량을 높일 수 있는 아주 높은 수의 반복과 함께 PBKDF2 와 같은 키 도출 기능을 사용하여 반드시 사용자 패스프레이즈로 암호화되어야 한다.

사용자 정보의 저장을 줄일 수 있는 방법은 아래와 같다:
(1) 전송과 전시는 괜찮지만, 메모리까지 지속하지 않는다. 이 또한 특별한 주의를 요하는데, 이는 데이터 스크린샷이 디스크에 기록된 곳에서 아날로그 누출이 스스로를 드러내지 않도록 보장하기 위해서이다.
(2) RAM 에만 저장한다.
(3) 어플리케이션이 시작할 때 요구되는 패스프레이즈로 암호화된 마스터키와 결합된 강도 높은 암호화를 사용하여 데이터를 암호화한다.

로컬 파일을 사용하고나서 어플리케이션 종료 전 이를 삭제하는 등의 다른 방법들은 데이터 복구를 방지하기에는 충분하지 않다.

2. 장치 상에 앱 데이터를 캐시에 은닉하는 것을 피한다.

데이터는 대체로 의도하지 않게 다양한 인공물에서 캡처될 수 있다. 개발자들은 로그/디버그 파일, 쿠키, 웹 히스토리, 웹 캐시, 특성 목록, 파일과 SQLite 데이터베이스를 포함한 데이터 저장 방법은 간과한다.

HTTP 캐싱을 방지한다. 개발자들은 iOS 가 데이터를 캐시에 은닉하지 않도록 설정할 수 있는데, 특히 HTTPS 트래픽이 그러하다. NSURLConnection 대표를 실행하는 것과 newCachedResponse 를 비활성화하는 것을 지켜본다.

추가로, 우리는 URL 히스토리와 레지스트레이션과 같은 웹 프로세스의 페이지 데이터가 캐시에 은닉되는 것을피하기 위해 이 절차대로 이뤄지길 추천한다. HTTP 캐싱은 헤더는 여기서 중요하고 웹 서버에서 구성 설정 된다. HTTP 프로토콜은 스스로가 끌어내온 응답이나 요청 모두의 일부분조차도 저장하지 않아야 한다는 브라우저를 지시하는 응답 헤더 안의 "no-store" 지시를 지원한다.

웹 어플리케이션에 관해선, HTML 형식 입력은 브라우저가 값들을 캐시에 은닉하지 않도록 지시하기 위해 autocomplete=off setting 을 사용할 수 있다. 캐싱 회피는 앱 활용 이후 장치 데이터의 forensic 검사를 통해 유효화 될 수 있다.

3. 민감한 데이터를 위해 query string 의 사용을 피한다.

심각한 은행 침범 사태는 간단한 query string 수정인 "공격"으로 실행되었다. query string 파라미터는 더욱 잘 보이고 때로 예상치 못하게 캐시에 은닉될 수 있다(웹 히스토리, 웹서버 혹은 프록시 로그 등). 중요한 데이터를 위한 암호화되지 않은 query string 의 사용은 피해야 한다.

대신, 안전한 POST 를 사용하여 XSRF 토큰 프로텍션과 함께 사용자 데이터를 전송한다. POST 데이터는 query string 데이터가 있는 곳에서 초기 설정 상태에서 기록되지 않는다.

만약 증명서가 query string 파라미터로서 전송되었다면, POST 요청의 몸통 부분에서와는 달리, 이것들은 다양한 장소에 로그인되어야 한다. 예를 들면, 사용자의 브라우저 히스토리 안에서, 웹 서버 로그 안에서, 그리고 호스팅 기반 시설 안에 사용된 모든 리버스 프록시의 로그 안에서 등이다. 만약 공격자가 이 중 어떠한 자원이라도 훼손하는데 성공한다면, 그는 그곳에 저장돼 있는 사용자 증명서를 캡처하는 방법으로 특권을 상승시킬 수도 있다.

POST 나 GET 이나, 임시 세션 쿠키가 사용되어야 한다. 데이터 암호화는 non-zero 초기화 벡터와 임시 세션 키 또한 연속적 공격을 방지하는데 도움이 된다. 만약 필요하다면, query string 데이터는 Diffie-Hellman 과 같은 보안 알고리즘을 사용하는 호스트들 사이에 협상된 임시 세션 키를 사용하여 암호화 될 수 있다.

4. 충돌 로그를 피한다.

만약 앱이 충돌하면, 결과적인 로그는 공격자에게 가치있는 정보를 제공할 수 있다. 앱들이 경고 없이 만들어지지 않고, 철저히 테스트 되어 충돌을 피하도록 확실히 한다. 이는 두말 없이 항시의 목표이며 충돌 로그로 인해 언급할 가치가 있다.

iOS 의 NSAssert 비활성화를 생각해보자. 이 설정은 만약 권리 행사가 실패하면 앱이 즉각적으로 충돌하도록 유발 할 것이다. 충돌하여 충돌 로그를 생성하는 것보다는 실패한 권리 행사를 처리하는 것이 더 낫다.

5. SSL/TLS 적용.

많은 앱들은 SSL 증명서를 적절하게 유효화하지 않고, man-in-the-middle 공격에 여전히 취약하게 놔둔다. 증명서는 고객에 의해 완전히 유효화 되어야 하고 신뢰된 루트 CA 에 의해 서명되어야 한다. 낮은 레벨의 암호화(export-grade 40 bit 암호화 같이)는 반드시 서버에서 비활성화 되어야 한다.

SSL 세션이 유효한지 확실히 하여 이를 보호하는 것에 추가로, 접근 당하는 URL 이 바이너리 자체에 쉽게 변경이 가능한 형식으로 저장하지 않도록 주의한다. 바이너리에서 간단히 URL 을 변경하는 것으로, 공격자는 URL 안의 도메인을 그들의 것으로 바꾸는데, 이는 유효한 SSL 증명서를 소유하고 있을 수도 있기 때문이다.

WAP 사이트에서는 HTTPS 와 호스트 주소를 체크한다. 비록 본래의 코드보다 변경하기 쉽기는 하지만, 고객 측의 자바 스크립트는 https, 적합한 호스트 URL 그리고 수용 가능한 호스트 IP 주소의 존재 여부를 체크하는데 사용될 수 있다. 개선된 점으로, 브라우저는 Strict Transport Security HTTP Header 를 실행하기 시작했는데, 이는 한 번 헤더가 수신되면 브라우저가 HTTPS 에 사이트를 요청하도록 한다.

사용자 인터페이스 관점에서 봤을 때, WAP 사이트는 사용자가 더욱 안전한 본토 모바일 앱을 설치할 수 있또록 격려한다. 개발자들은 잠금 아이콘이나 아무런 실제 보안 메커니즘에 의존하지 않는 "보안"이란 장황함을 보여주는 유혹을 피해야 하는데, 이는 이것이 신뢰의 잘못된 감지를 장려할 수 있기 때문이다.

다수의 경우, 공공 증명 당국의 증명서를 사용할 필요가 없는 것은, 이것들은 고객이 시스템이 연결하는 곳이 신뢰되지 않은 곳이란 것을 알지 못하는 경우가 발생하는 시스템을 위해 설계되었다. 어플리케이션은 정확히 스스로가 어디에 연결하는지 알고 있기 때문에, 그리고 선천적으로 스스로가 연결하는 곳의 기반 시설을 신뢰하기 때문에, 공공 증명서 당국과는 별개로 "사적" 공공 키 기반 시설을 사용하는 것이 가능하다(때로는 더욱 안전하다). 이러한

방법으로 공격자는 MITM 공격이나 그와 비슷한 다른 공격들을 수행하기 위해 서버 측에서 개인 키를 요청할 것이다.

6. 로컬 세션 타임아웃 활성화.

모바일 장치는 자주 분실 또는 도난 당하고, 민감한 데이터에 접근, 거래 집행 또는 장치 소유자의 계정을 연구하는데 사용될 수 있는 앱은 짧은 시간의 휴식 이후 로그인을 요구해야만 한다. 어느 때던지 앱이 5 분 이상 활용되지 않는 때, 활성 세션은 종료시키고, 사용자는 증명서를 다시 입력하도록 요구되어야 한다.

어플리케이션 세션의 시간이 종료되면, 어플리케이션은 사용자 데이터와 데이터 암호 해제를 위해 사용된 모든 마스터키와 연관된 모든 메모리를 버리거나 삭제해야 한다. 도난 당한 장치에서 작업하는 숙력된 공격자는 만약 삭제되지 않으면 잠재적으로 이 메모리로부터 데이터를 훔칠 수도 있다.

7. 디버그 로그를 비활성화 시킨다.

디버그 로그는 민감한 데이터를 저장할 수 있다. 디버그 로그는 생산 빌드에서 활성화되지 않도록 한다.

iOS 에서 NSLog 내역서 비활성화는 도난 당할 위험이 있는 잠재적으로 민감한 데이터를 제거하고, 앱의 수행 능력을 약간 향상시킬 것이다.

전형적으로 앱에 의해 디버그 메세지를 출력하는데 사용되는 안드로이드 시스템 로그는 메모리에 있는 몇 킬로바이트 크기의 순환 로그이다. 장치 재부팅에서 이는 삭제되지만, 그때까지 readlog 허가가 있는 안드로이드 앱은 로그의 정보를 얻을 수 있다.

8. 계정 숫자를 숨기고 토큰을 사용한다.

많은 앱 스토어는 여러 가지 스크린에서 계정 숫자를 완성시킨다. 공공 장소에서 모바일 앱의 보편적 사용이 이뤄진 상황에서, 일부 숫자만 보여주는 (예: ***9881) 것은 이 정보의 최대한의 프라이버시을 보장할 수 있다. 장치에 전체 숫자를 저장해야하는 필요가 있지 않는 한, 일부분 숨겨진 숫자를 저장한다.

때로 계정 숫자는 서버 측 계정 데이터를 참조하기 위해 사용된다; 이 데이터는 메모리로부터 쉽게 도난 당하거나 일부의 경우, 사용자가 접근할 수 있는 권한이 없어야 하는 계정으로 작업하기 위해 조작될 수 있다. 그래서 계정 숫자 대신, 토큰이 각각의 계정에 배치되고 고객에 제공되는 것을 추천한다. 사용자가 추측할 수 없는 이러한 토큰들은 실제 계정에 서버측 맵핑(mapping)을 갖고 있다. 만약 어플리케이션 데이터가 도난 당하게 되면, 사용자의 계정 숫자는 노출되지 않을 것이고, 공격자 또한 직접적으로 계정 숫자를 참고할 수는 없겠지만, 반드시 먼저 계정으로 다시 돌아갈 수 있는 토큰을 결정해야 한다.

9. 쿠키에 보안 설정을 적용한다.

Set-Cookie 헤더는 반드시 안전한 설정을 사용해야 한다. 만약 쿠키가 안전하다고 표시되면, 호스트가 있는 세션이 안전할 경우에 한해서만 전송될 것이며, HTTPS 연결을 통해서만 전송될 것이다. 이 설정은 HTTPS 를 이용하는 앱의 모든 쿠키에 대해 적용되어야 한다.

10. 사용자 이름의 캐싱(cache 에 은닉하기)을 제한한다.

사용자가 "Save this User ID" 기능을 활성화시키면, 사용자 이름은 iOS 상의 CredentialsManager object 안 캐시에 은닉된다. 런타임에는 사용자 이름이 어떤 종류의 인증 절차도 발생하기 전에 메모리로 실행되어, 악성 프로세스가 사용자 이름을 가로채도록 잠재적 위험을 허용한다.

불안전한 저장 방법이나 런타임 때의 잠재적 가로채기를 통해 이러한 정보 유출을 피하는 동안 저장된 사용자 이름의 편리함을 사용자에게 제공하는 것은 어렵다. 비록 패스워드만큼 민감한 것은 아니지만, 사용자 이름은 보호되어야 할 사적인 데이터이다.

더 높은 안전성으로 캐시에 은닉된 사용자 이름의 선택을 제공하는 한 가지 잠재적 방법은 실제 사용자 이름 대신 감춰진 사용자 이름을 사용하고, 인증 속의 사용자 이름 값을 hash 값으로 대체하는 것이다. 해쉬 값은 레지스트레이션에 저장된 고유의 장치 토큰을 포함해 만들어질 수 있다.

해쉬와 장치 토큰을 사용하는 프로세스의 이점은 실제 사용자 이름이 본체에 저장되지 않거나 보호되지 않은 메모리로 실행되지 않으며, 다른 장치로 복사되거나 웹에서 사용된 값은 적합하지 않을 것이다. 그러면 악의적 사용자는 인증하기 위한 사용자 이름을 성공적으로 훔치기 위해 더 많은 정보를 캐내야 할 것이다.

11. 코드 난독화 및 팩킹 적용.

당신의 코드 그리고 총체적 기술 레벨과 공격하기 위해 필요한 시간을 증가시키는 것의 리버스 엔지니어링을 복잡하게 만든다. 리버스 엔지니어링 앱은 어떻게 앱이 작동하는지 가치있는 지식을 제공할 수 있다. 이것들을 내부적으로 더욱 복잡하게 만듬으로서, 공격자는 명확한 앱의 작업 경로를 보는데 불이익을 받게 되고, 이로서 공격 벡터는 줄어든다.

안드로이드 앱 (.apk file)의 리버스 엔지니어링은 비교적 쉽게 얻어지고, 그리고 나서 어플리케이션의 내부 작업 성과가 검사를 거친다. 악의적 사용자들에게 검사의 난이도를 높이기 위해 본 안드로이드 개발자 참고서에서 설명한 바와 같이 코드를 불명료하게 만드는 것을 추천한다.

http://developer.android.com/guide/publishing/licensing.html#app-obfuscation

iOS 어플리케이션은 애초에 설계된 방법 상의 문제때문에 리버스 엔지니어링 공격에 취약하다. 앱의 클래스와 프로토콜은 목표 파일 안에 저장되는데, 이는 공격자가 어플리케이션의 설계를 완전히 파악하게 해준다. Objective-C 스스로가 반사적 언어이고, 스스로의 상태를 인식하고 변경하는 것이 가능하다; 적절한 도구를 가진 공격자는 런타임이 어플리케이션을 관리하는 것과 동일한 방법으로 어플리케이션의 상태를 인식하고 변경할 수 있다. Objective-C 는 매우 쉽게 추적이 가능하고 가로채기 쉽게 조작이 가능하며, 심지어는 어플리케이션의 런타임에도 간섭하는 간단한 메세징 프레임워크를 통합한다. 상대적으로 간단한 공격은 Objective-C 런타임을 조작하여 이것이 인증 절차와 정책 검사, 내부 어플리케이션 위생 검사, 혹은 어플리케이션의 정책을 준수하도록 감시하는 로직 체크의 일종을 우회하도록 사용될 수 있다.

만약 어플리케이션이 매우 민감한 데이터를 다룬다면, 안티 디버그 기술 실행을 고려해 본다. 당신의 코드를 리버스 엔지니어링 하는 것의 복잡성을 증가시킬 수 있는 다양한 기술이 존재한다. 한 가지는 C/C++을 사용하여 공격자에 의한 모든 런타임 조작을 제한하는 것이다. 매우 성숙하고 Objective-C 와 결합하기 매우 쉬운 ample C 와 C++라이브러리가 있다. class-dump, class-dump-z 그리고 Cycript 와 같은 Objective-C 런타임과 Objective-C 리버스 엔지니어링 도구에 의한 노출과 조작을 피하기 위해 낮은 레벨 C 에서 코드의 중요 부분을 써 넣는 것을 고려해 본다.

디버거 억제하기 - 시스템 콜을 통함으로서, 어플리케이션은 운영 시스템은 디버거가 프로세스에 붙도록 허가하지 않아야함을 명시할 수 있다. 디버거가 붙는 것을 방지함으로서, 낮은 레벨의 런타임에 간섭하는 공격자의 능력이 제한된다. 공격자는 낮은 레벨에서 어플리케이션을 공격하기 위해서 반드시 먼저 디버깅 제한을 우회해야 한다. 이는 공격에 더 높은 복잡성을 더해준다. 안드로이드 어플리케이션은 공격자 또는 악성 소프트웨어에 의해 쉽게 런타임 조작을 방지하기 위해 android:debuggable="false" 를 어플리케이션 매니페스트에 설정해 놓아야 한다.

추적 검사 - 어플리케이션은 디버거나 다른 디버깅 툴에 의해 추적당하고 있는지 아닌지를 파악할 수 있다. 만약 추적당하고 있다면, 어플리케이션은 사용자 데이터를 보호하기 위해 암호화 키를 버리거나 서버 관리자에게 통보하거나 혹은 스스로를 방어하기 위한 시도로서 다른 종류의 반응과 같이 모든 가능한 공격 대응 행동을 펼칠 수 있다.

최적화 - 진보된 수학적 계산과 다른 종류의 복잡한 로직을 숨기려면, 컴파일러 최적화 활용이 대상 코드를 불명료하게 하도록 도와 공격자에 의해 쉽게 분해되지 않고, 이로써 공격자가 특정 코드에 대한 이해를 하기가 더욱 어려워진다. 안드로이드에서 이는 본토에서 NDK 로 컴파일된 라이브러리를 활용하여 더욱 쉽게 이뤄낼 수 있다.

바이너리 벗기기 - 본래 바이너리를 벗기는 것은 공격자가 어플리케이션의 낮은 레벨 기능의 구성을 비춰보기 위해 필요한 시간과 능력 레벨의 총량을 증가시키는 효과적인 방법이다. 바이너리를 벗김으로써, 바이너리의 심볼 테이블이 벗겨지고, 이로 인해 공격자는 어플리케이션을 쉽게 디버그 하거나 리버스 엔지니어링 할 수 없게 된다. 바이너리 벗기기는 Objective-C 클래와 iOS 의 대상 맵핑 데이터를 버리지 않는다.

앱 스토에 분포돼 있는 어플리케이션 안의 iOS 바이너리는 암호화되어 복잡성에 또 하나의 층을 추가하는 것이다. 디지털 권리 관리(Digital Rights Management) 암호화를 벗기기 위해 도구가 존재하는 한편, DRM 의 레이어가 시간과 바이너리를 공격하기 위해 필요한 능숙 정도의 총량을 증가시킨다. 앱 스토어 어플리케이션에서 사용된 암호화는 그러나 능숙한 공격자가 어플리케이션이 실행 중에 직접적으로 장치의 메모리에서 실행된 메모리를 버리는 것으로써 벗겨질 수 있다.

12. 주소 공간 레이아웃 임의화를 사용한다.

iOS 4.3 에서 소개된 새로운 무료 보안 기능인 ASLR 은 어떻게 앱이 실행되고 메모리에 유지되는지를 임의로 순서를 정한다. ASLR 은 어플리케이션에서 사용되는 주소 공간의 순서를 임의로 정해, 어플리케이션이 먼저 충돌하지 않고서는 악성 코드를 실행하기 어렵게 만들었다. 이는 또한 어플리케이션의 항당된 메모리를 없애는 과정을 복잡하게 만든다.

iOS 가 이제는 ASLR 을 지원하기 때문에, 어플리케이션은 이 점을 이용해야 한다. ASLR 의 완전한 보호를 위해선, 어플리케이션은 반드시 PIE(Position-independent-executable)에 대한 지원과 함께 컴파일 되어야 한다.

PIE 는 fPIE 옵션으로 커맨드 라인에 의해 컴파일 할 때 활성화 시킬 수 있다.

경고 - 활성화된 fPIE 로 컴파일된 어플리케이션은 iOS 4.3 이상의 버전에서만 돌아갈 것이다. 이는 어플리케이션의 버전들을 유지하는 데 필요한 논점이다.

공격자는 새로운 PIE 기능르로 다시 컴파일되지 않은 어플리케이션을 목표로 정할 수 있다. 이 문제점은 널리 공인화되어 PIE 가 아닌 어플리케이션이 목표물이 될 확률을 높였다. 이 문제점은 CNNMoneyTech 이라는 문서의 아래의 내용처럼 처리되었다.

애플은 아이폰의 최신 버전에서 중요한 보안 기능을 소개했지만, 아직 제 3 의 어플리케이션에서는 매우 드물게 사용되고 있어 사용자들을 목표 지향 공격에 취약하게 남겨뒀다. 주소 공간 레이아웃 임의화라고도 알려진 ASLR 은 아이폰 안의 키 조각들의 순서를 임의로 정해, 공격자가 이것들이 어디에 저장되어 있는지 찾기가 힘들도록 만들어졌다. ASLR 의 한 구성 요소인 PIE 는 해커들이 공격을 가하기 위해 사용해야 하는 실행 가능한

코드를 숨긴다. 한 번 활성화되면, 그 도구들은 아이폰이 해커에 의해 원격적으로 이용당하는 것으로부터 보호할 수 있다.

13. 간단한 로직은 피한다.

코드 안의 간단한 로직 테스트는 공격에 더욱 취약하다.

예) *if sessionIsTrusted = 1*

이는 간단한 로직 테스트이고 만약 공격자가 저 한 값을 바꿀 수 있다면, 그들은 보안 통제를 피해갈 수 있다. 애플 iOS 는 이러한 종류의 약점을 이용해 공격 당해왔고, 안드로이드 앱들은 다양한 보호 메커니즘을 피해가기 위해 Dalvik 바이너리들을 개선하였다.

이러한 로직 테스트들은 많은 레벨에서 피해가기가 쉽다. 조립 단계에서는, 공격자는 알맞는 CBZ(compare-and-branch-on-zero) 혹은 CNBZ(compare-and-branch-on-nonzero)설명서를 찾아 이를 뒤집기 위해 디버거 하나만을 이용하여 iOS 어플리케이션을 공격할 수 있다. 이것은 간단히 대상의 메모리 주소를 가로지르고 어플리케이션이 실행되면서 instance 변수를 바꾸는 것만으로 런타임에도 수행될 수 있다. 안드로이드에서 어플리케이션은 SMALI 와 재컴파일링 전에 개선된 브랜치 컨디션으로 디컴파일 될 수 있다.

더 좋은 프로그래밍 패러다임을 생각해보자. 세션을 신뢰할 수 없을 때 서버에 의해 혹은 특정 데이터의 암호 해제 방지에 의해 특권이 실행되거나, 그것도 아니면 어플리케이션이 도전/응답, OTP 혹은 다른 형식의 인증 절차를 통해 세션을 신뢰할 수 있다고 결정할 때까지 사용 가능하다.

추가로 모든 위생 검사 기능 고정 인라인을 선언하는 것을 추천한다. 이러한 접근 방법으로 이들은 인라인으로 컴파일 되어 패치아웃 하기가 더욱 어려워진다. (예로, 공격자도 간단히 한 기능을 중단하거나 패치할 수 없다). 이 기술은 공격자가 어플리케이션으로부터의 모든 검사의 instance 를 찾고 패치하도록 하여, 공격의 필요한 복잡함을 증가시킨다.

매우 민감한 앱들에 대해서는, 보안 코딩 원칙에서 설릭된 더욱 정교한 접근이 앞으로 계속 조사하는데 가치가 있을 것이다. 암호화, 시간제 콜백 그리고 플로우 기반 프로그래밍 등의 통합 기술은 공격자에게 복잡함을 추가시킬 수 있다.

14. 간단한 로직 변수를 피한다.

같은 맥락에서, 대상에 저장된 간단한 로직 변수는 공격자에 의해 쉽게 조작될 수 있다. 예:

session.trusted = TRUE

이러한 값들은 현재 어플리케이션에 의해 사용 중인 클래스의 instance 안에서 공격자에 의해 읽고 쓰여질 수 있다. iOS 에서는 Objective-C 런타임을 조작함으로서, 이러한 변수들은 조작되고, 그리하여 다음에 어플리케이션에 의해 참고되면, 모든 조작된 값들이 대신 읽혀질 것이다.

15. 키보드 캐시(cache)에 주의한다.

iOS 는 맞춤형 자동 교정과 형식 완성기와 같은 기능을 제공하기 위해 사용자가 입력하는 것을 기록하지만, 민감한 데이터 또한 저장될 수 있다. 거의 모든 非 숫자는 이것이 입력된 순서대로 키보드 캐시에 은닉된다. 캐시의 내용은 어플리케이션의 관리 특권을 넘어서며, 그래서 데이터는 어플리케이션에 의해 캐시에서 제거될 수 없다.

단지 비밀 번호 분야만이 아니라 민감한 정보를 위해 자동 교정 기능을 비활성화시킨다. 민감한 정보를 키보드 캐시에 은닉하기 때문에, 이는 복원될 수도 있다. UITextField 의 경우, autocorrection Type 특성 설정을 UITextAutocorrectionNo 로 바꿔서 캐시에 은닉하는 설정을 비활성화 시킨다.

이러한 설정은 SDK 가 업데이트 되면서 바뀔 수도 있기 때문에, 이것이 완전히 조사되었는지를 확인한다. 정기적으로 키보드 딕션어리를 지우기 위해 기엄 정책을 추가한다. 이는 엔드 유저가 간단히 Settings 어플리케이션으로 가서, General > Reset > Reset Keyboard Dictionary 처럼 설정하면 된다.

안드로이드는 사용자가 입력한 단어가 미래의 자동 교정을 위해 저장될 수 있는 사용자 사전을 갖고 있다. 이 사용자 사전은 특별 허가를 갖고 있지 않은 모든 앱에서 사용 가능하다.

더 높은 보안을 위해선, 캐싱과 악성 소프트웨어에 대한 추가적 보호 제공을 중지시킬 수 있는 커스텀 키보드(잠재적으로 PIN 도) 실행을 고려해 본다.

16. 복사/붙여넣기를 주의한다.

iOS 는 이제 복사/붙여넣기를 지원한다. 데이터가 처음에 암호화 됐는지와 무관하게 감한 데이터가 저장될 수 있고, 클리어 텍스트의 클립보드에서 복원될 수 있다. 복사/붙여넣기 API 는 계속 발전 중이며 민감한 데이터를 유출할 수도 있다.

안드로이드는 또한 초기 설정 상태에서 복사/붙여넣기 기능을 지원하고, 클립보드는 모든 어플리케이션이 접근할 수 있다.

적합하다면, 민감한 데이터를 다루는 곳에서는 복사/붙여넣기 기능을 끈다. 복사의 선택권을 제거하는게 데이터 노출을 피하는데 도움이 될 수 있다.

17. 제 3 자 라이브러리를 테스트한다.

개발자는 제 3 자 라이브러리에 매우 깊게 의존한다. 이것을 당신의 코드를 테스트하면서 철저하게 조사하고 테스트하는 것이 중요한다.

제 3 자 라이브러리는 취약점과 약점을 포함할 수 있다. 많은 개발자들이 제 3 자 라이브러리는 잘 개발되고 테스트 돼 있는 줄 알지만, 이들의 코드에도 문제점이 존재할 수 있다. 보안 감사는 반드시 제 3 자 라이브러리와 기능 또한 철저히 테스트해야 한다. 이것은 핵심 iOS 와 안드로이드 코드/라이브러리를 포함해야 한다.

새로운 버전의 제 3 자 라이브러리(또는 OS 버전)로 업그레이드하는 것은 당신 앱의 버전처럼 다뤄져야 한다. 업데이트 된 제 3 자 라이브러리(혹은 새로운 OS 버전)은 새로운 취약점을 갖고 있거나 당신의 코드에서 문제점을 노출할 수도 있다. 이것들은 당신의 앱을 위한 코드를 테스트할 때처럼 똑같이 테스트 해야 한다.

iOS 에서, 통계적으로 LD_PRELOAD 공격을 피하기 위해 제 3 자 라이브러리를 컴파일한다. 이러한 공격에서 라이브러리와 이의 기능들은 악성 코드로 대체되어 공격자의 라이브러리와 맞교환 될 수도 있다. 안드로이드는 LD_PRELOAD 공격에 취약하다.

18. 지리위치를 조심스럽게 사용한다.

안드로이드와 iOS 는 위치를 정확하게 파악하기 위해 GPS 를 사용한다. GPS 데이터를 사용하고 저장하지 않는 것의 의미를 생각해 보자. 더 나은 프라이버시를 위해선 가능한 한 조잡한 위치 서비스를 사용한다.

요구되지 않은 한, GPS 정보를 기록하거나 저장하지 않는다. 특정 어플리케이션에 GPS 를 사용하는 것이 유용할 수 있지만, 데이터를 기록하고 저장하는 것은 매우 드물게 필요하다. 이를 피하는 것은 많은 프라이버시와 보안 문제점을 제기한다. GPS 위치 추적 정보는 종종 iOS 와 안드로이드 상의 지정된 캐시에 한동안 은닉된다.

일부 어플리케이션은 GPS 를 자동적으로 사용한다. 한 예로 이미지를 종종 geo-tag 하는 카메라이다. 만약 이게 문제라면, XIF 데이터를 이미지로부터 벗겨내도록 주의한다.

안전한 위치에서 작업할 때는, GPS 데이터가 정확도를 높이기 위해 애플과 구글 서버에 다시 보고될 수도 있단 점을 기억한다. 애플은 장치가 연결돼 있는지의 여부와 관계없이 일정 범위 내의 가까운 액세스 포인트에 관한 정보를 캡처하는 것으로 알려져있다. 자신의 좌표 또는 무선 네트워크 위상 기하학이 판매 회사로 보고되지 않아야 하는 안전한 위치에서 또는 근처에서 실행될 어플리케이션 안의 GPS 를 활성화시키지 않는다. 이것에 추가적으로, 단일 액세스 포인트의 하드웨어 주소에 대한 지식은 공격자가 안전한 무선 환경을 모방하고 애플이나 구글로부터 GPS 환경 좌표를 돌려보내는데 사용될 수 있다.

19. iOS: 캐시에 은닉된 어플리케이션 스냅샷을 피한다.

인터페이스에서 시각적 변이를 보여주기 위해, iOS 는 장치 NAND 플래시의 파일 시스템 일부 안에 이미지로서 스냅샷을 캡처하고 저장하는게 증명되었다. 이는 장치가 정지하거나(종료가 아닌), 홈버튼이 눌렸을 때, 또는 전화 통화나 어플리케이션을 임시로 정지시키는 다른 이벤트가 있을 때 발생한다. 이러한 이미지들은 사용자와 어플리케이션 데이터를 종종 포함하고 있을 수 있고, 하나의 공식 발표된 케이스에서는 사용자의 이름, DOB, 주소, 고용자 그리고 신용 점수 등이 포함됐다.

민감한 데이터를 보호하기 위해서는, API 구성 설정이나 코드를 이용하여 어플리케이션 스냅샷의 캐싱을 막는다.

willEnterBackground API 사용 - iOS 앱이 백그라운드에 전송되려 할 때, 앱이 아무런 민감한 정보도 보여주고 있지 않은지 확인한다. 앱용 스플래쉬 스크린을 만들고 백그라운드로 움직이면서 이를 보여준다.

20. 안드로이드: 파일 허가를 조심스럽게 실행한다.

모든 앱이 설령 앱의 개인 데이터 디렉토리에 저장되어있다 하더라도 파일을 읽고 쓸 수 있기에 필요치 않는 한 MODE_WORLD_READABLE 혹은 MODE_WORLD_WRITABLE 의 허가로 파일을 생성하지 않는다.

21. 안드로이드: 인텐트를 조심스럽게 실행한다.

인텐트는 구성 요소 간 신호를 주고 받는데 사용될 수 있다:

- Activity 를 시작하려면, 앱용 유저 인터페이스를 여는데
- 시스템과 앱에 변경 내용을 알리려 방송하면서
- 백그라운드 서비스를 시작, 종료 그리고 소통하는데
- ContentProviders 를 통해 데이터에 접근하는데
- 이벤트를 다루기 위해 콜백하는데

인텐트를 통해 접근한 구성 요소는 공적이나거나 사적일 수 있다. 초기의 설정은 인텐트 필터에 의존하고, 실수로 구성 요소가 공공화 되도록 하기가 쉽다. 이를 막기 위해선 앱의 Manifest 에서 구성 요소를 android:exported=false 로 설정하는 것이 가능하다.

Manifest 에서 선언된 공공 구성 요소는 초기 설정에서 열려 있기 때문에, 모든 어플리케이션이 여기에 접근이 가능하다. 만약 구성 요소가 다른 모든 앱에 의해 접근될 필요가 없다면, Manifest 에서 선언된 구성 요소에서 허가를 설정하는 것을 고려해본다.

공적 구성 요소에게 수신한 데이터는 신뢰할 수 없고 반드시 정밀 분석되어야 한다.

22. 안드로이드: 검사 활동(Check Activity)

전형적으로 안드로이드 어플리케이션에서 Activity 는 앱에서 "Screen"이다. Activity 는 만약 수출되면 모든 어플리케이션에 의해 작동시킬 수 있다.
- 만약 Activity 를 위해 인텐트 필터를 정의 내린다면, 이것은 스스로를 자동적으로 공적으로 만들어 다른 어플리케이션에 의해 접근이 가능하게 할 것이다.
- 설령 Activity 가 공적이지 않더라도, 루트 특권에 의해 여전히 작동시킬 수 있다. 이는 데이터에 접근하기 위해 비밀번호 잠금 화면을 뛰어넘는 것과 같이 개발자가 의도하지 않은 방법으로 공격자가 어플리케이션을 넘나들 수 있게 해줄 수 있다. 이를 방지하기 위해 Activity 는 앱이 "잠금 해제"상태에 있는지, 아니라면 잠금 화면으로 다시 돌아가는지를 확인해 볼 수 있다.

- 인텐트 필터가 무엇으로 정의 내려진지와 상관없이, 공적으로 접근 가능한 Activity 는 나쁜 데이터와 직접적으로 작동될 수 있기 때문에, 입력 유효화가 중요하다.

- Activity 를 시작하는 인텐트에 민감한 데이터를 두지 않는다. 악성 프로그램은 더 높은 우선권을 인텐트 필터에 삽입하여 데이터를 채갈 수 있다.

23. 안드로이드: 브로드캐스트를 조심스럽게 실행한다.
브로드캐스트 인텐트를 전송할 때, 특정 앱만이 인텐트를 수신할 수 있도록 개발자는 브로드캐스트 상에 허가를 설정해 놓을 수 있다. 만약 아무런 허가가 설정되지 않았다면, 특권이 없는 모든 앱이 노골적인 목적지가 있지 않는 한 인텐트를 수신할 수 있다.

또한 만약 브로드캐스트 인텐트를 통해 제 3 자 구성 요소에 민감한 정보를 전송한다면, 구성 요소는 악성 설치에 의해 대체되었을 수도 있다는 것을 생각한다. 합법적 구성 요소로서 완전히 똑같은 구성 요소 이름으로 앱을 생성하는 악성 개발자에게는 억제 요소가 없고, 이름 쓰는 공간이 이미 사용되고 있지 않는 한 악성 앱은 설치될 것이다.

24. 안드로이드: PendingIntents 펜딩인텐트를 조심스럽게 실행한다.
펜딩인텐트는 앱들이 마치 앱을 만들어내는 것처럼 명시된 인텐트를 콜할 수 있는 또 다른 앱에 실행 권한을 넘기도록 해준다. 이는 외부 앱들이 사적 구성 요소로 다시 콜하도록 해준다.

외부 앱은, 만약 악성이라면, 목적지나 데이터/보전에 영향을 주려 시도할 수 있다. 그리하여 사적 브로드캐스트 수신기/Activity 로의 지연된 콜백으로 펜딩인텐트를 사용하고, 노골적으로 베이스 인텐트 안의 구성 요소 중 하나의 이름으로 명시하는 것이 가장 좋다.

25. 안드로이드: 서비스를 유효화한다.

서비스는 보통 백그라운드 프로세싱을 위해 사용된다. Activity 와 브로드캐스트 수신기와 비슷하게, 서비스는 외부 어플리케이션에 의해 작동될 수 있고, 적절하게 허가와 수출 플래그에 의해 보호 받아야 한다.

서비스는 외부 caller 에서 작동시킬 수 있는 방법이 두 개 이상일 수 있다. 각각의 방법에 임의의 허가를 정의하고 만약 콜링 패키지가 check-Permission()과 상응하는 허가를 갖고 있는지 체크하는 것이 가능하다. 다른 방법으로는, Manifest 에서 별도의 서비스와 허가 있을 때만 가능한 안전한 접근을 정의할 수 있다.

민감한 데이터와 함께 서비스를 콜할 때는, 악성 서비스가 아니라 올바른 서비스를 유효화시켜야 할 수도 있다. 만약 연결하려는 구성 요소의 정확한 이름을 알고 있다면, 연결할 때 사용한 인텐트 안에서 이를 명시할 수 있다. 그렇지 않으면 checkPermission()을 다시 사용하여 콜링 패키지가 당신의 인텐트를 수신하기 위해 필요한 특정 허가를 갖고 있는지 확인할 수 있다. 앱에게 허가를 부여할지는 설치하는 때에 사용자의 선택에 달려있다.

26. 안드로이드: 인텐트 스니핑을 피한다.

브로드캐스트 인텐트를 사용하는 또 다른 어플리케이션에 의해 액티비티가 시작되면, 인텐트에 넘어간 데이터는 악성 앱에 의해 읽힐 수 있다. 악성 앱은 또한 어플리케이션에 관한 최근 인텐트의 목록을 읽을 수 있다. 예를 들어, 안드로이드 웹 브라우저가 자신에게 URL 을 보내는 앱에 의해 작동되면, 그 URL 은 스니핑 될 수 있다.

민감한 데이터는 앱들 사이에서 브로드캐스트 인텐트를 사용하여 주고 받아선 안된다.

27. 안드로이드: 컨텐트 제공자를 조심스럽게 실행한다.

컨텐트 제공자(Content Provider)는 앱이 URI 주소 체계와 상관적 DB 모델을 사용하여 데이터를 공유하는 기본 방법이다. 이들은 URI 를 거쳐 "files"에 접근하는 방법으로 사용될 수도 있다.

컨텐트 제공자는 허가를 선언할 수 있고, 읽기와 쓰기 접근 권한을 분리할 수 있다. 필요하지 않다면 쓰기 접근 권한을 활성화시키지 않는다. 필요치 않다면, 모든 특권이 없는 앱이 컨텐트 제공자를 읽는 것을 방지하도록 허가를 활성화시킨다.

컨텐트 제공자는 전체 데이터베이스를 공유하는 일 없이 특정 기록(혹은"file")을 공유하기 위해 기록 레벨의 위임을 허락한다. 작업을 위해 필요한 최소한의 접근 권한을 제공하기 위해 이 기능을 사용한다. 예를 들면, 외부 앱이 모든 메세지에 대한 접근을 배제하면서, 연락처의 누군가에게 이 메일을 보내는 외부 어플리케이션으로 단일 "Instant Message"를 공유할 수 있다. 외부 앱이 본래의 생성하는 앱으로 돌아오면, 위임이 종료된다.

28. 안드로이드: 가장 좋은 웹뷰 실천 방법

웹뷰는 수많은 보안 문제점을 소개하고 조심스럽게 실행되어야 한다.

- 만약 필요치 않다면, 자바 스크립트와 플러그인 지원을 끈다. 초기 설정에서 이들은 이미 꺼져있지만, 노골적으로 이를 설정하는 것은 좋은 실천 방법이다.
- 로컬 파일 접근을 끈다. 이는 앱의 자원과 자산 디렉토리에 대한 접근을 제한하고, 그 곳 자체에서 접근 가능한 다른 파일에 대한 접근 권한을 찾는 웹 페이지로부터 오는 공격을 완화시킨다.
- 제 3 자 호스트로부터의 컨텐트 실행을 방지한다. 앱 안에서 완전히 이를 방지하기란 쉽지 않지만, 개발자는 웹뷰 안에서 시작된 대부분의 요청을 가로채고, 조사하며 유효화하기 위해 shouldOverrideUrlLoading 과 shouldInterceptRequest 를 중단시킬 수 있다.
- 화이트리스트 체계 또한 URI 클래스가 URI 의 구성 요소를 검사하고, 승인된 자원의 화이트리스트와 매치하는지 확인하도록 사용함으로서 실행될 수 있다.

29. 안드로이드: 캐시에 은닉된 카메라 이미지를 피한다.

많은 원격 디포짓 캡처 앱은 NAND 메모리에 체크 이미지 데이터를 남겨놓는데, 이는 심지어 삭제된 후에도 그렇다. 그래서 장치에 어떠한 체크 이미지 인공물도 남기지 않는 다른 접근법을 사용할 것을 추천한다. 한 가지 가능한 접근법은:

- SurfaceView 를 만들고 Camera Preview 에서 보여주거나, 카메라가 비추는 영상의 라이브 미리보기를 보여주도록 한다.
- 버튼을 삽입하고, 이것을 눌렀을 때, Camera Preview 를 픽셀 배열로 바뀌도록 프로그래밍한다. 그러면 이것은 비트맵으로 바뀌고, JPG, 코딩된 base64 로 압축되며, 모두 RAM 에 있는 원격 사이트에 제출된다.

30. 안드로이드: GUI 물체 캐싱을 피한다.

안드로이드에서 어플리케이션이 스크린은 메모리에 유지되고, 모든 어플리케이션은 멀티태스킹 때문에 메모리에 유지된다. 장치를 찾거나 훔치는 공격자는 여전히 유지되고 있는 이전에 사용된 스크린으로 바로 이동해서 GUI 에 나타난 모든 데이터를 본다. 이것의 한 예는 사용자가 그들의 거래 내역을 보고 어플리케이션에서 "로그 아웃"하는 은행 업무 어플리케이션이다. 직접적으로 거래 화면 활동을 시작함으로서, 공격자는 이전에 나타났었던 거래를 볼 수 있다. 이를 막기 위해선, 개발자가 3 가지 흔히 사용되는 옵션이 있다.

1) 사용자가 로그 아웃하면 앱을 완전히 종료시킨다. 당신의 앱을 종료시키는 것은 안드로이드 설계 원칙에 반하지만, GUI 가 파괴/재활용되기 때문에 이렇게 하는 것이 더욱 안전하다.
2) 사용자가 로그인 상태에 있는지 체크하기 위해 각 Activity/스크린의 시작에서 테스트를 실시하고, 로그인 상태가 아니라면, 로그인 스크린으로 전환한다.
3) 스크린을 떠나거나 로그 아웃할 때 GUI 스크린에 있는 데이터를 무력화시킨다.

31. iOS: 키체인을 조심스럽게 사용한다.
iOS 는 안전한 데이터 저장을 위해 키체인을 제공하지만, 몇 가지 경우에 있어서, 케체인은 훼손되고 이후에 암호가 해제될 수 있다.

iOS 5 를 포함한 그 아래의 모든 버전에서, 애플은 키체인 파일이 iTunes 백업에 암호화되어 있지만 깨뜨릴 수 있는 방식으로 저장된 상황을 만들었다. 이러한 경우, 공격자는 키체인을 손상시키는데 성공적일 확률이 높다.

iOS forensic 과 보안 도구에 대한 최근의 진보는 이제 키체인을 깨뜨릴 수 있게 해준다. 공격자가 iOS 장치와 특화된 소프트웨어에 대한 물리적 접근을 가져야하는 동안에, 키체인은 이제 노출되었다. iOS 4 와 iOS 5 의 암호를 해제할 오픈소스와 다운로드 가능한 도구들이 있다.

매우 민감한 데이터에 대해서는, 키체인 사용을 자제하는 방향을 검토해본다. 대신에 어플리케이션이 시작할 때 또는 사용자 인증 후 서버에서 보낸 암호화 키와 함께 사용자가 어플리케이션 패스프레이즈에 들어가는데 의지한다.

32. 보안 데이터 저장을 실행한다.
모바일 장치에 데이터를 안전하게 저장하는 것은 적절한 기술을 필요로 한다. 언제든지 가능하면, 그저 간단히 데이터를 저장/캐시에 은닉하지 말아라. 이것은 장치에서 데이터 손상을 완전히 피할 수 있는 유일한 방법이다.

커스텀 암호화 - 민감한 데이터를 저장할 때는, 당신의 혹은 제 3 자의 암호 기능을 포함한다. 당신의 코드베이스를 사용함으로써, 실행에 관해 강한 제어 능력을 갖게되고, 메인 iOS 데이터 보호 클래스에 초점이 맞춰진 공격은 당신의 어플리케이션을 직접적으로 목표로 지정하지 않을 것이다. 그러나 접근법은 더욱 복잡하고, 잘못할 경우, 보안 능력을 감소시킬 수 있다는 단점이 있다. 애플과 안드로이드의 공통 암호 라이브러리는, 적절할게 실행할 경우, 매우 안전한 암호 실행을 제공할 수 있는 여러 가지 기본 암호 기능을 제공한다.

언제든 사용자 데이터를 암호화 할 때는, 데이터가 접근 당하면 사용자가 제공한 패스프레이즈를 이용해 암호화된 마스터키를 이용해 암호화하도록 주의한다. 이는 만약 공격자가 장치에서 마스터키를 추출해내면 데이터가 게 복원되는 것을 방지해준다. 애플의 데이터 보호 API 와 키체인, 그리고 대부분의 안드로인 핸드셋의 장치 암호화 부족으로 인해, 마스터키나 패스프레이즈가 장치의 그 어떤 곳에도 저장하는 것을 추천하지 않는다.

iOS - 복잡한 패스프레이즈와 함께 iOS 안에 내장된 데이터 보호 API 는 데이터 보호를 위한 추가적 레이어를 제공할 수 있지만, 커스텀 암호를 실행하는 것만큼 안전하지는 않다. 이를

이용하려면, 파일은 반드시 구체적으로 보호가 필요하다고 표시되어야 한다. 애플의 데이터 보호 API 를 이용해 구체적으로 암호화되지 않은 모든 데이터는 암호화되지 않은 채로 저장되어 있다.

안드로이드 - 안드로이드에서는 SD 카드와 같은 외장 메모리는 확실한 허가가 없고, 초기 설정에서 모든 앱은 메모리에 대한 읽기 권한이 있고 모든 파일을 읽을 수 있다는 점을 기억한다.

안드로이드는 AES 와 같이 데이터를 지킬 수 있는 기본 암호 라이브러리를 실행한다. 이 방법으로 암호화된 데이터는 키와 키 관리를 도출해내는데 사용되는 패스워드만큼만 안전하다는 점을 기억한다. 패스워드 정책, 길이 그리고 복잡성을 사용자 편의에 대조하여 생각해보고, 어떻게 암호화 키가 메모리에 저장되는지도 생각해 본다. 루트 접근 권한을 갖고서는, 실행 중인 프로세스의 메모리를 제거하고 암호화 키를 위해 이를 찾는 것이 쉽다.

기본 암호 제공자 "AES"를 사용하는 것은 덜 안전한 AES-ECB 로 되돌아갈 것임 또한 주의한다. 그래서 256 비트 키와 SecureRandom 에 의해 생성된 임의의 IV 로 AES-CBC 를 명시하는 것을 추천한다. 그리고 또 잘 검증된 PBKDF2(Password-Based Key Derivation Function)를 사용하여 패스프레이즈로부터 키를 도출해 낼 것을 추천한다.

33. UUID 의 한계

대부분의 모바일 장치는 신원 확인의 목적으로 만들어진 고유의 ID(UUID)를 갖고 있다. 고유하게 장치를 식별하는 능력은 종종 데이터를 조달, 관리 그리고 보호하는데 매우 중요하다. 개발자들은 장치 식별을 위해 UUID 를 신속하게 도입하였고, 이로 인해 많은 시스템들에게 보안의 기초를 다졌다.

그러나 불행하게나 이 접근법에는 몇 가지 프라이버시와 보안 문제점이 있다. 첫째, 많은 온라인 시스템들은 장치의 UUID 를 심지어 사용자가 앱에 로그인이 되어있지 않을 때에도 어플리케이션을 지나 추적이 가능케 할 수 있는 개인 사용자에게 연결시켰다. 사용자를 추적할 수 있는 이러한 진보된 능력은 주요한 프라이버시 문제로 떠올랐다.

하지만 이를 넘어서, UUID 를 통해 사람을 식별하는 앱들은 이전의 장치 소유자를 새로운 사용자에게 노출시키는 위험을 만들었다. 한 실험의 예로, 아이폰을 재설정한 뒤, 설령 모든 사용자 데이터가 삭제된 뒤에도 온라인 음악 서비스에 대한 접근 권한을 얻을 수 있었다. 공격자가 UUID 를 위조하는 것은 쉬운 일이기 때문에, 이것은 보안 위협 뿐만 아니라 프라이버시 문제까지 될 수 있다.

애플은 UUID 의 프라이버시와 보안 위험성 모두를 인식하고, 개발자 접근을 제거했다. 그러나 개발자는 무선 네트워크 인터페이스의 MAC 주소와 같은 개체를 고유하게 식별하는 다른

방법들을 찾아냈다. MAC 주소 또한 속이기 어렵지 않고, 그러므로 더욱 믿을만한 접근법을 제시하지 못한다.

개발자는 특히 장치 인증 실행에 필수적이라면, 장치가 제공하는 아무런 식별기로도 장치를 식별하는데는 사용하지 않을 것을 권고한다. 대신 레지스트레이션, 설치 과정 혹은 앱이 처음으로 실행되는 때에 앱 특유의 "device factor"의 생성을 추천한다. 이 고유 앱 장치 요소는 세션을 생성하려는 사용자 인증과 함께 필요할 것이다. 장치 요소는 암호화 요소의 일부분으로 사용될 수 있다.

이것은 예측할 수 있는, 장치가 제공하는 데이터에 의존하는 것이 아니기 때문에, 이용이 더욱 어려울 수 있다. 도전-응답 접근법을 이용함으로서, 서버와 장치는 사용자 인증 전에 서로를 인증할 수 있다. 시스템 접근 권한을 얻기 위해선 공격자가 이 두 가지 요소 모두를 이용해야 할 것이다. 개발자는 또한 장치 요소가 고객 혹은 서버 측에서 재설정되는 기능을 실행할 수 있어, 사용자와 장치에 대한 더욱 엄격한 재 인증을 강요토록 한다.

34. 간섭 여부를 검사한다.

보안 앱은 항 간섭 기술을 갖고 있는데, 이는 간섭 받았을 경우 실행되는 것을 막아준다. 이는 고도로 안전한 어플리케이션에서 종종 발견되는데, 군대에서 사용되는 것이 그러하다. 그러나 이런 개념은 상업적 앱에도 결합이 가능하다.

검사 합계를 사용하면, 어플리케이션에서 사용되는 파일 상의 디지털 서명과 다른 유효화 메커니즘은 데이터 파일이 어플리케이션을 조작하려는 의도록 간섭 받았는지, 또한 모든 데이터 파일이 진짜인지를 어플리케이션에 임베드된 서명을 실행함으로서 탐지하도록 도와준다. 이러한 검사들은 공격자가 피해갈 수 있다는 점을 기억해두나, 이런 검사 과정은 어플리케이션을 깨는데 필요한 시간을 더욱 복잡하게 만들 뿐 아니라, 어플리케이션이 소리소문 없이 자신의 사용자 데이터 키나 다른 중요한 데이터를 간섭이 탐지됐을 때마다 없애버린다면, 공격자에게도 좌절감을 심어줄 수 있다. 유사하게, 간섭을 탐지한 어플리케이션은 관리자에게 통보할 수 있다.

안드로이드에서 앱에 서명하는데 사용되는 공공의 키는 앱의 증명서에서 읽을 수 있고, 어플리케이션을 조회하는데 사용되는 공공의 키는 개발자의 사적 키로 서명되었다. 이는 만약 앱이 공격자에 의해 백도어로 침입 당하고 문제가 생기면 탐지하는데 특히 유용하다. 이러한 공격들은 악성 코드를 자동적으로 삽입하고 제 2 의 시장에 출시하는 유명한 앱들이나 금융 앱들 사이에서 점점 더 성행하고 있다.

35. 강화된/두 가지 요소 인증을 실행한다.

패스워드는 간단해서는 안 된다. 알파벳과 숫자를 조합한 최소 6 자리 이상의 복잡한 패스워드를 지원하고 요구한다. 사용자가 만들지 않는 단어/아이콘 선택을 추가하는 것은 그들의 패스워드가 다른 서비스에서 훼손 당했을 때 사용자를 보호하도록 도와준다.

일부 경우에서는 사용자 이름과 패스워드는 모바일 앱에서 충분한 보안을 제공하지 않는다. 민감한 데이터나 거래가 포함된 경우, 두 가지 요소 인증을 실행한다. 매번 로그인 때마다는 사용 못할 수도 있지만 사이사이나 선택된 기능에 접근할 때 사용될 수 있다. 스텝업 인증은 비거래 분야에 정상적인 접근을 허용하고, 민감한 기능에 대해서는 두 번째 계층의 인증 절차를 요구한다.

강화된 인증 절차에 대한 옵션은 아래와 같다:

1. 추가적인 비밀 단어
2. SMS 나 이메일이 제공한 추가적인 코드 -- 하지만 공격자가 훔친 장치에서는 두 개 모두에게 접근할 가능성이 크다.
3. 패스워드에 추가적인 사용자가 아는 값을 더한다.
4. 사용자에 의해 사전에 설정된 보안 질문/대답 (등록하는 중에)

최대한 보안을 위해서는, 원타임 패스워드(OTP)의 사용은 사용자에게 어플리케이션이 올바른 증명서를 갖고 있는 것 뿐만 아니라, 원타임 패스워드를 갖고 있는 물질적 토큰을 소유하도록 요구한다.

36. 어플리케이션 세팅을 보호한다.

iOS 개발자들은 종종 어플리케이션 세팅을 일부 경우에서는 훼손될 우려가 있는 plist 파일에 저장한다.

이와 유사하게, 안드로이드 개발자들은 설정 사항을 초기 설정에서 암호화되지 않았으며 읽히거나 심지어는 루트 허가로 변경까지 가능한 공유된 환경 설정 XML 파일이나 SQLite DB 에 저장한다.

가능하다면 코드 안에 세팅을 컴파일하도록 한다. iOS 에서는 변경 사항이 묶음화되어야 하고 어쨌거나 새로운 앱으로서 배치되어야하기 때문에, 앱을 plist 파일을 통해 구성 설정하는 것에는 큰 이점이 없다. 대신에 공격자들이 조작하려면 더 많은 시간과 기술을 필요로 하는 앱 코드 안에 구성 설정을 포함하도록 한다.

먼저 암호화되지 않은 한, 딕션어리나 다른 파일에는 아무런 중요한 세팅도 저장하지 않는다. 이상적으로는, 사용자가 제공한 패스프레이즈로 암호화된 마스터키나 사용자가 시스템에 로그인 할 때 원격적으로 제공된 키를 사용하여 모든 구성 설정 파일을 암호화 하는 것이다.

37. 민감한 데이터를 RAM 에 안전하게 저장한다.

암호 해제 키와 같은 민감한 데이터를 사용할 때는 필요 이상으로 RAM 에 오랜 시간 놔두지 않는다. 키를 지탱해주는 변수들이 사용 후에 무력화될 수 있다.

안드로이드에서는 어플리케이션이 메모리가 재사용 되어야 할 때까지 사용 후에도 계속 남아있기 때문에, 암호화 키가 백그라운드에 계속 있을 수 있다. 이런 장치를 찾거나 훔치는 공격자는 디버거를 붙이고 어플리케이션으로부터 메모리를 없애버리거나 RAM 의 모든 내용물을 없애버리기 위해 kernel 모듈을 실행할 수도 있다.

38. 데이터의 안전한 삭제 이해하기

안드로이드에서 file.delete()을 콜하는 것은 파일을 안전하게 삭제하지 않을 것이며, 이것이 덮어 쓰여진 것이 아닌 이상, 장치의 물질적 이미지에서부터 쓰여질 수 있다. 파일을 덮어 쓰는 것에 대한 전통적인 접근법은 일반적으로 NAND 플래시 메모리의 공격적인 관리 때문에 모바일 장치에서는 작업을 하지 않는다. 개발자는 장치에 쓰여진 모든 데이터는 복원될 수 있다고 생각해야 한다. 일례로서, 암호화는 보호막을 한 층 추가해 줄 수 있다.

추가로, 대부분의 어플리케이션에는 추천하지 않지만, 파일을 삭제하고 모든 사용 가능한 공간을 덮어 쓰는 것인데, 이는 NAND 플래시로 하여금 할당되지 않은 모든 공간을 삭제하도록 강요할 것이다. 이 기술의 단점은 NAND 플래시를 닳게 만들어, 앱과 장치 전체가 느려지고, 현저하게 높은 전력 소모 등을 초래한다.

앞서 언급한 바와 같이, 최적의 접근법은 장치에 민감한 데이터 저장을 피하는 것이다.

39. 사용자 식별기로서 MEID 의 사용을 피한다.

비록 멀티플 세션에서 "동일한 장치"를 식별하는데 편리하지만, MEID 에는 두 가지 중요한 문제점이 있다.

첫째, MEID 는 개인 정보라기에는 너무 많은 사람들을 거쳐가며, 사용자의 동의 없이 회사의 개인의 활동(GPS 위치, 구매 등등)을 추적하는데 사용될 수 있는 키 식별기로 인해, 프라이버시 문제점이 있다. 둘째, MEID 는 많으 앱들이 사용 가능하고, 장치에서 스푸핑 될 위험이 있기 때문에 훼손된 것으로 생각한다.

이러한 문제점에 대한 고려 때문에, 차선의 방법은 특정 장치를 위해 앱의 설치/활성화에 고유의 장치 토큰을 생성하는 것이다. 이러한 토큰은 아무런 사용자 정보도 저장하지 않을 것이지만, 세션들 사이에서 거래 중에 "활성화된" 장치를 식별하는데 사용될 수 있다.

네트워크 레이어 보호하기

스마트폰 같은 모바일 장치는 이것저것 잡다한 것을 다룬다. 이들은 보안에 대한 별다른 생각 없이 다양한 Wi-Fi 네트워크에 무분별하게 연결한다. 이러한 네트워크는 감시되거나, 호스트 장치가 되거나, 공격을 취하는 사용자일 수도 있다. SSL 이 웹 소통을 보호하는 기준으로 역할을 하는 동안, 여러 가지 방식으로 취약하다는 것이 증명되었다. 개발자는 이러한 취약점을 목표로 하는 공격에 대항하여 그들의 앱을 강화시킬 수 있다.

40. SSLStrip 으로부터 보호한다.

MITM 공격은 모바일 웹 앱에서는 방지하기가 어렵다. 이것은 네트워크에서 투명하게 HTTP 트래픽을 장악하고, HTTPS 요구를 관찰하며, 그리고 나서 SSL 을 제거함으로서 고객을 안전하지 않은 연결에 남겨둔다.

SSLStrip 은 웹 앱에서는 막기 어려운 공격이지만, 이러한 위험을 완화시켜줄 수 있는 몇 가지 절차들이 있다. 첫째, 가능다하면 어플리케이션을 안전하게 보호하기 위해 모든 HTTP 트래픽을 제거한다. 이는 위험 자체를 제거하지는 않지만, 적절한 기반을 다지는데는 도움이 된다.

둘째, 고객이 SSL 이 활성화되도록 이를 유효화 시킨다. 이는 완전히 본래의 앱에서는 매우 직선적이다. 모바일 웹 앱에서는, 이는 자바스크립트를 통해 이뤄낼 수 있고, 만약 HTTPS 연결이 탐지되지 않았다면, 고객이 HTTPS 로 다시 방향을 설정할 수 있다. HTTPS 의 기본적인 검사에 대한 예제는 다음과 같다:

```
<script type="text/javascript">
if (document.location.protocol != "https:")
{
document.location.href = "https://subdomain.yourdomain.com" +
document.location.pathname;
};
</script>
```

SSL 을 요구하는 더욱 믿을 만한 방법이 만들어졌다. Strict Transport Security HTTP Header 가 바로 그것이다. 이 헤더는 브라우저가 헤더를 한 번 수신했던 적이 있는 사이트에 대해 HTTPS 를 요구하도록 한다. 그러나 브라우저는 그저 Strict Transport Security HTTP Header 를 실행하게 시작하는 것이고, 모바일 브라우저 지원은 늦어진다.

사용자에게 안전한 연결을 보장하지만 유효화 된 HTTPS 세션에는 의존하지 않는 아이콘/언어를 피한다. 마지막으로, 사용자 교육은 SSLStrip 공격의 위험을 줄이는 중요한

구성 요소이다. 사용자가 대면하는 모든 문서, 소통, 트레이닝 등에서 HTTPS 의 중요성을 강조하라.

41. 증서 피닝 (Certificate Pinning)

만약 당신의 앱이 SSL 을 통해 서버에 전화를 걸면, 연결의 진실성을 유효화하기 위해 CA 서명에 위존할 이유가 없다(초기 설정 상의 행동이기도 한). CA 증명서는 당신이 사전에 목적지를 모를 때의 일반 목적 네트워크 소통을 위한 것이다. 앱이 연결한 서버의 주소를 알고 있기 때문에, 사실상 올바른 증명서인 당신의 앱에 나타난 증명을 유효화 시킬 수 있다. 이를 증서 피닝이라고 부르며, 이는 널리 공용화된 DigiNotar 와 Comodo 사고와 같은 CA 피해를 막을 수 있다.

구글은 이미 구글 서비스를 위해 크롬 브라우저에서 실행 중이며, 공식 안드로이드 트위터 고객은 증서 피닝을 포함한다.

데이터 센터 레이어 보호하기

데이터 센터는 많은 웹과 모바일 공격의 최종 목표인데, 이는 이곳이 가치 있는 데이터들이 움집해 있는 대형 시장이기 때문이다. 모바일 앱을 가지고서 일부 개발자들은 그들이 여전히 공적 웹 서비스를 노출하고 있고, 모바일 앱을 위해 백엔드를 제공하는 서버를 보호해야 한다는 점을 잊는다. 데이터 센터 레이어에서 보안을 위한 최적의 실천 방법을 따르는 것은 개인 앱 사용자를 보호하는 동시에 데이터베이스나 웹 서버의 막심한 피해를 주는 위반 현상을 피하도록 도와준다.

42. 고객으로부터 입력을 유효화 시킨다.

좋은 웹 앱 보안이 그러듯이, 고객으로부터의 모든 입력은 신뢰되지 않아야 한다. 서비스는 반드시 앱으로부터 입력을 철저히 필터링하고 유효화 시켜야 한다. 설령 데이터가 당신의 앱에서 온다고 하여도, 이를 가로채고 조작하는 것이 가능하다. 이는 앱 충돌, 버퍼 오버플로우, SQL 주입 그리고 다른 공격들을 유발하는 공격들을 포함한다. 전송 전에 모든 사용자 입력을 적절히 검사한다. 여기에 추가로, 어플리케이션 안에서 로직 검사를 넣지 않는데, 이는 이런 로직 검사는 공격자가 당신의 어플리케이션을 장악하여 피해갈 수 있기 때문이다. 이러한 로직 검사는 서버 측에서 실시한다.

43. 웹 서버: 체크 세션 세팅

사용자의 세션은 대부분의 앱에서 취약할 수도 있는 쿠키를 통해 유지된다.

웹 언어(예: 자바, .NET)는 세션 관리를 제공하는데, 이는 잘 개발되고 보안성이 검증된 것들이다. 보안 패치를 통해 서버 소프트웨어를 최신으로 업데이트한다. 당신의 세션 관리를 굴리는 것은 더욱 위험하다. 이런 위험은 뛰어난 전문 기술을 갖고서만 떠맡도록 한다.

세션 쿠키의 크기가 충분한지 확인한다. 짧거나 예측 가능한 세션 쿠키는 공격자가 세션에 대해 예측, 장악 혹은 다른 공격을 펼치는 것을 가능케 해준다. 세션 구성 설정에서는 고도의 보안 설정을 사용한다.

44. framing 과 클릭재킹을 방지한다.

framing 은 위장된 사이트를 통해 클릭재킹 공격을 유도하여 iFrame 을 이용, 의도하지 않은 웹/WAP 사이트로 전달하는 포직을 포함한다. 클릭재킹은 매우 현실적인 위협으로 페이스북이 정보를 훔치거나 사용자들이 사이트를 공격하도록 만드는 것을 포함하여 사이트에서 악용되어 왔다.

framing 의 방지를 위한 한 방법은 client-side 자바스크립트를 이용하는 것이다. 대부분의 웹 사이트는 더 이상 자바스크립트 없이 설계되거나 실행되는 것이 불가능하기 때문에, 자바스크립트로 보안 조치를 실행하는 것은 하나의 선택이다(자바스크립트 없이 사이트를 무력화시키는 것도). 비록 고객측이기 때문에 간섭에 영향을 받지 않기에, 이 레이어는 공격자에게 바를 눈에 띄게 올려준다.

아래는 프레임을 "top"하도록 강요하는 자바스크립트 코드의 한 예로, 이를 통해 사이트에서 실행됐던 프레임을 "busting" 한다.

```
if( self != top ) {
      top.location = self.location ;
}
```

프레임 버스팅 코드를 막기 위해 공격자가 그들의 프레임에 추가할 수 있는 추가적인 절차가 있다. 여기에는 언로드에서 사용자에게 종료하지 말라고 부탁하는 경고가 포함된다. 더 복잡한 자바스크립트는 이러한 기술들을 마주칠 수도 있다. 최소의 기본 프레임 버스팅 코드를 포함하는 것은 간단한 framing 을 훨씬 더 어려운 과정으로 만든다.

X-FRAME-OPTIONS-HEADER - 새롭고 더 나은 반 framing 옵션이 최근 일부 브라우저에서 응답으로 보내진 HTTP 헤더에 기반해 실행되었다. 웹서버 차원에서 이 헤더의 구성 설정을 함으로서, 브라우저는 프레임이나 iFrame 에서 응답 내용을 보여주지 않도록 지시되었다.

Apache 구성 설정 파일에서 이것의 실행 예시는 다음과 같다.

```
Header add X-FRAME-OPTIONS "DENY"
```

또 다른 옵션은 이 값을 동일한 도메인에서 온 프레임만을 허용할 "SAMEORIGIN"으로 설정하는 것이다. 이 헤더는 iOS 4 의 Safari 를 포함한 다양한 브라우저에서 검증되었고, iFrame 에서 페이지 전시를 방지하는 것으로 확인되었다. 만약 iFrame 에서 전달에 관한 아무런 요구 사항이 없다면, DENY 를 사용하는 것을 추천한다.

45. 웹 서버 구성 설정

웹 서버에서의 특정 설정은 보안을 강화할 수 있다. 웹 서버 상에서 한 가지 흔히 간과되는 취약점은 정보 유출이다. 정보 유출은 심각한 문제로 이어질 수 있는데, 서버에서 오는 공격자가 얻을 수 있는 모든 정보의 조각은 공격을 가하기 더 쉽게 만들어준다.

정보 유출을 줄이는 간단한 방법은 장황한 오류를 없애는 것이다. 장황한 오류는 개발 환경에서는 유용할 수 있지만, 생산 환경에서는 웹 프레임웍 정보와 버전 등의 치명적인 정보를 유출할 수 있다. 공격자는 이런 정보를 특정 결함 실행을 악용하도록 설계된 목표 지향성 공격을 하는데 사용할 수 있다.

정보 유출을 줄이는 또 다른 간단한 방법은 서버 응답에서 최소량의 정보만을 돌려보내는 것이다. 초기 설정에서, Apache 는 자신의 버전 번호, 실행 중인 OS 그리고 실행 중인 플러그인을 돌려보낼 것이다. 구성 설정 파일에서 단 하나의 줄을 바꾸는 것만으로, 서버가 Apache 를 실행 중이기 때문에 유출하는 것만으로 줄일 수 있다.

서버에서 보안을 현격하게 높일 수 있는 한 가지 구성 설정 변화는 아무 초기 설정의 디렉토리를 바꾸는 것이다. 공격자들은 초기 설정의 로그인, 쉽게 추측 가능한 관리자 인터페이스 그리고 "숨겨진" 디렉토리를 위한 간단한 이름 짓기 체계 등의 "low-hanging-fruit"(낮게 걸린 열매. 쉽게 할 수 있는 일)을 가진 사이트를 자주 인터넷을 통해 본다. 웹 접근이 필요한 서버에서 모든 민감한 페이지의 위치를 불명료하게 하는 것은 좋은 정책이다.

관리 또는 다른 제한된 분야는 필요치 않는 한 공공연하게 웹 접근이 가능해서는 안되며, 반드시 강제 공격에 저항해야 한다. 잠금 보호가 없는 HTTP 인증이나 형식 인증은 강제적인 힘에 의해 공격 당할 수 있다(당할 것이다).

46. SSL 구성 설정

SSL 증명서가 올바르게 설치되었고 가능한 최고 강도로 암호화되었는지 확인한다. 많은 웹 서버는 낮은 강도의 암호화 설정을 허용하는데, 아주 약한 export-grade-40-bit 암호화가 여기에 포함된다. 매우 강한 암호(128 비트 이상)와 SSLv3/TLSv1 만을 활성화한다.

TLSv1 은 10 년이 넘었고 2009 년에는 "재협상 공격"에 취약한 것으로 드러났다. TLSv1 프로토콜을 사용하는 대부분의 서버는 이 취약점을 없애기 위해 패치되었지만, 이는 확인되어야 한다. TLSv1 프로토콜은 업데이트 됐고, 더욱 최근의 TLSv1.2 는 사용 가능한 최신 기술과 최고 강도의 암호를 제공한다. 더 새로운 버전의 TLS 로 업데이트하는 것은 이를 더욱 굳건히 하고 미래에도 경쟁력을 유지하게 해줄 것이다.

47. 형식 토큰과 함께인 CSRF 로부터 보호한다.

CSRF(Cross-site Request Forgery)는 알려진 또는 예측 가능한 form 값과 로그인 된 브라우저 세션에 의존한다. 각각의 form 제출은 form 으로 또는 사용자 세션 시작 때에 실행된 토큰을 갖고 있어야 한다. 사용자가 만들었지는를 확인하기 위해 POST 를 수신할 때 서버에서 이 토큰을 체크해 본다. 이 기능은 주요 웹 플랫폼과 함께 제공되었고, 최소의 커스텀 개발과 함게 form 에서 실행될 수 있다.

48. 웹 서비스를 보호하고 모의침투 진단(Pentest)해본다.

일반적으로, 생산 웹 서버는 철저히 검사되고 악성 공격에 대해 강화되어야 한다. 훼손된 서버는 사용자 증명서를 가로채고 앱 사용자에게 다른 공격들을 가할 위험이 있다.

생산 서버 소프트웨어는 현재 보안 버전으로 업데이트되어야 하고, 서버 소프트웨어와 인터페이스에 관한 정보 유출을 막기 위해 보안 강화되어야 한다.

인증 form 은 사용자 이름이 존재하는지를 반영해서는 안 된다. 만약 공격자가 유효한 사용자 이름을 파악할 방법이 있다면, 그들은 강제 공격과 피싱 공격에 대한 시발점을 갖게 된다. "무효한 사용자/혼합 넘어가기"와 "사용자 이름 검색 불가" 둘 모두의 고객에게 동일한 응답을 제공함으로서 사용자 이름 수확을 방지한다.

모든 로그인 형식과 민감한 데이터를 교환하는 형식/페이지는 HTTPS 를 실행하고 요구해야 한다. 웹 서버는 그러한 자원을 위해 SSL 없이 고객 연결을 허용해서는 안 된다. 장황한 오류를 없애고, 불필요한 사이트나 페이지 레거시를 제거하고, 잠재적 공격에 대해 계속해서 웹 자원을 강화한다.

49. 내부 자원을 보호한다.

관리자 로그인 형식과 같은 내부 사용을 위한 자원은 외부 접근으로부터 막아져야 한다. 이런 자원들은 잠금 없는 HTTPS 혹은 form 인증과 같은 강제적 힘에 저항할 수 없는 인증 절차를 자주 이용한다. 관리 혹은 다른 내부 자원의 훼손은 확장 데이터 손실과 다른 피해로 이어질 수 있다.

공공 인터넷 접근을 요구하지 않는 모든 자원은 방화벽 규칙과 네트워크 분할을 사용하여 제한되어야 한다. 만약 로그인 페이지, 관리자 영역 혹은 다른 자원이 외부로부터 접근이 가능하다면, 이는 악의적 사용자에게 발견될 것이고, 강제적 힘에 의해 공격 당할 것이다.

2.9. 당황하지 마라

어플리케이션을 안전하게 하는것은 언제나 힘든 일이었다. 모바일 앱은 심지어 더욱 어렵다. 그러나 하늘은 무너지지 않고 우리는 슬기로운 조언을 따라야 한다. **당황하지 마라** 왜냐하면: **모바일 앱을 안전하게 만드는 것은 가능하다.**

(이하 번역 생략)

We hope our 42+ Best Practices: Secure mobile development for iOS and Android provides development and security teams with empirical knowledge in meeting this challenge. Through awareness, education, knowledge sharing and a focus on security, we can all enjoy a safer mobile environment.

If you find our 42+ Best Practices for Secure Mobile Development useful, you might want to take a look at appSecure®, our mobile application security audit and certification program. In addition, you'll probably find **Santoku Linux**, our free mobile security, malware and forensics platform quite helpful.

viaForensics is partnering with IT certification and education leader **CompTIA** to create certifications in secure development for iOS and Android. These certifications are scheduled to be available later in 2012, so stay tuned.

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