



OWASP

Mobile Security Project

OWASP Mobile Top 10 – 2016

가장 심각한 모바일 어플리케이션 보안 위험 Top 10

※ BlackFalcon(번역:이지혜, 감수:장경칩) 번역
공식 한국어 버전이 아님을 알려드립니다.



Creative Commons (CC) Attribution Share-Alike
Free version at <https://www.owasp.org>



OWASP 소개

OWASP

오픈 웹 어플리케이션 보안 프로젝트(OWASP)는 개방 커뮤니티로서, 기관이 신뢰할 수 있는 어플리케이션과 API를 개발, 구입, 유지 관리하는 데 기여하고 있습니다. OWASP는 무상 제공하는 것들은 다음과 같습니다.

- 어플리케이션 보안 도구와 표준
- 어플리케이션 보안성 테스트, 안전한 코드 개발, 코드 보안성 검토와 관련된 서적
- 표준 보안 통제와 라이브러리
- 전 세계 지부
- 최신 연구
- 대규모 전세계 컨퍼런스
- 메일링 리스트

<https://www.owasp.org> 에서 더 많은 정보를 확인

어플리케이션 보안성 향상에 관심 있는 모든 이에게 OWASP의 도구, 문서, 포럼, 지부에 대한 모든 것들이 무료입니다. OWASP는 어플리케이션 보안에 대해 사람, 프로세스, 그리고 기술의 문제로서 접근하고자 합니다. 어플리케이션 보안에 대한 가장 효과적인 접근은 이러한 모든 부분에서 향상이 요구됩니다.

OWASP는 새로운 형태의 조직입니다. 상업적인 목적이 나 이윤 압박이 없기 때문에, 어플리케이션 보안에 대해 공정하고, 실질적이고, 효율적인 정보를 제공할 수 있게 합니다. OWASP는 잘 알려진 상업적 목적의 보안 기술에 대한 정보를 제공하고 있지만, 어떠한 기업과도 제휴나 협약을 맺고 있지 않습니다. OWASP도 다른 오픈 소스 소프트웨어 프로젝트와 마찬가지로 협업과 공개적인 방식으로 여러 자료를 만듭니다.

OWASP 재단은 프로젝트의 장기적 성공을 보장하기 위한 비영리기구입니다. OWASP 이사회, 글로벌위원회, 채터 대표, 프로젝트 리더와 프로젝트 회원을 포함하여, OWASP에 참여하는 거의 모든 분들은 자원 봉사자입니다. OWASP는 혁신적인 보안 연구에 대한 자금 및 인프라를 제공하고 있습니다.

OWASP와 함께하시기 바랍니다!

OWASP Mobile Security

OWASP 모바일 보안 프로젝트는 개발자와 보안팀에게 안전한 모바일 어플리케이션을 구축하고 유지 관리하는데 필요한 리소스를 제공하기 위한 중앙집중식 리소스입니다. 이 프로젝트를 통해 우리의 목표는 모바일 보안 위험을 분류하고 발발 통제를 제공하여 악용 가능성 또는 영향도를 줄이는 것입니다.

최우선순위는 어플리케이션 계층입니다. 위협 모델링 및 관리 기능을 구현할 때, 기본 모바일 플랫폼 및 통신 사업자 고유의 위험을 고려하지만 평균적으로 개발자가 변화를 일으킬 수 있는 영역을 목표로 삼고 있습니다. 또한 최종 사용자 장치에 배포된 모바일 어플리케이션 뿐만 아니라 모바일 앱과 통신하는 광범위한 서버 측 인프라에도 중점을 둡니다. 우리는 모바일 어플리케이션, 원격 인증 서비스 및 클라우드 플랫폼 별 기능 간의 통합에 중점을 두고 있습니다.

이 프로젝트는 아직 진행중인 작업입니다. 우리는 이 일을 하는 소그룹으로 더 많은 도움을 받을 수 있습니다! 관심이 있으시면 프로젝트 리드 중 한 곳에 연락하거나 메일링 리스트를 방문하십시오.

저작권 및 라이선스

영문저작권 © 2003 – 2017 The OWASP Foundation
(현재 보고 계신 문서는 공식 한글 번역본이 아님을 알려드립니다.)

이 문서는 Creative Commons Attribution ShareAlike 3.0 license의 보호를 받는다. 재사용 또는 배포할 경우, 이 저작물에 적용된 저작권을 명확하게 나타내야 합니다.



OWASP Mobile Top 10 – 2016

Release Candidate를 90일간 검토하고 배포한 결과 Release Candidate가 최종 배포판(Final)으로 준비를 마쳤습니다. 2016 리스트가 발표되었으며, OWASP 홈페이지에서 확인할 수 있습니다

2015년에 설문 조사를 실시하고, 전세계로 피드백을 요청하였습니다. 이는 OWASP Mobile Top 10 – 2016을 분석하고 다시 분류하는 데 도움이 되었습니다. 따라서 상위 10개 항목은 이제 서버가 아닌 모바일 어플리케이션에 더 중점을 둡니다. Top 10 2016의 목표는 다음과 같습니다.

- 위키 콘텐츠 업데이트; 테스트 가이드와 링크, 풍부한 그림 예제 등
- 더 많은 데이터
- PDF 배포

이 목록은 커뮤니티에서 90일간 피드백을 받은 후 확정되었습니다. 피드백을 바탕으로 데이터 수집과 유사한 방식으로 데이터를 논리적이고 일관된 방식으로 그룹화한 다음 Mobile Top 10 2016을 발표했습니다.

OWASP Mobile Top 10 – 2014(이전)	OWASP Mobile Top 10 – 2016(신규)
M1 – 약한 서버측 제어	M1 – 적절하지 않은 플랫폼 사용
M2 – 취약한 데이터 저장소	M2 – 취약한 데이터 저장소
M3 – 취약한 전송계층 방어	M3 – 취약한 통신
M4 – 의도하지 않은 데이터 노출	M4 – 취약한 인증
M5 – 잘못된 권한부여 및 인증	M5 – 취약한 암호화
M6 – 취약한 암호화	M6 – 취약한 권한부여
M7 – 클라이언트측 인젝션	M7 – 취약한 코드품질
M8 – 신뢰할 수 없는 입력값을 통한 보안 결정	M8 – 코드 변조
M9 – 부적절한 세션처리	M9 – 리버스 엔지니어링
M10 – 바이너리 보호 결여	M10 – 불필요한 기능

M1 – 적절하지 않은 플랫폼 사용

이 항목은 플랫폼 기능의 오용 또는 잘못된 플랫폼 보안 설정에 대해 다룹니다. Android intent, 플랫폼 권한, TouchID 오용, 키 체인 또는 모바일 OS의 일부인 다른 보안 컨트롤이 포함될 수 있습니다. 모바일 앱이 이런 위험을 겪을 몇 가지 사항들이 있습니다.

M2 – 취약한 데이터 저장소

이 새로운 항목은 Mobile Top 10 2014의 M2(취약한 데이터 저장소)와 M4(의도하지 않은 데이터 노출)을 통합하였습니다. 이는 안전하지 않은 데이터 저장 및 의도하지 않은 데이터 유출에 대해 다루고 있습니다.

M3 – 취약한 통신

이 항목은 악의적인 핸드 셰이킹, 잘못된 SSL버전, 약한 협상, 민감정보의 평문통신 등을 다룹니다.

M4 – 취약한 인증

이 항목은 최종 사용자 인증 또는 잘못된 세션 관리에 대해 다룹니다. 다음사항을 포함합니다.

- 사용자 식별을 못하는 경우
- 사용자 신원이 유지되지 않는 경우
- 세션 관리 취약점

M5 – 취약한 암호화

중요한 정보 자산에 암호화를 적용합니다. 그러나 암호는 어떤 면에서는 충분하지 않습니다. M3(취약한 통신)에서 TLS 또는 SSL과 관련된 사항들이 있습니다. 또한 앱이 암호화 사용을 해야 할 때, 암호화 사용을 하지 않았다면, M2(취약한 데이터 사용)에 속합니다. 이 항목은 암호화 시도 문제에 대한 이슈를 다루지만, 자세하게 다루지 않습니다.

M6 – 취약한 권한 부여

이 항목은 잘못된 권한(Authorization)부여(예:클라이언트 측의 권한 부여, 강제 검색 등)를 다룹니다. 이는 인증(Authentication) 문제(예:기기 등록, 사용자 식별 등)와 다릅니다. 사용자 인증을 하지 않은 모든 상황(예: 인증된 후, 권한이 부여 되어 접근이 필요할 때, 일부 리소스 또는 서비스에 익명 접근 허용)이라면, 권한부여가 잘못된 것이 아니라 인증이 잘못된 것입니다.

M7 – 취약한 코드품질

이 항목은 사용이 낮은 항목 중 하나인 M8:신뢰할 수 없는 입력을 통한 보안 결정이었습니다. 이것은 모바일 클라이언트의 코드 수준 구현 문제에 대한 포괄적인 내용입니다. 이는 서버 측 코딩 실수와 구별됩니다. 또한, 버퍼오버 플로우, 포맷스트링 취약점 및 모바일 장치에서 실행되는 일부 코드를 재작성해야 하는 코드 레벨에서 실수하는 것들을 다룹니다.

M8 – 코드 변조



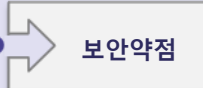


이 범주에는 바이너리 패치, 로컬 리소스 수정, 메서드 후킹, 메서드 변경 및 동적 메모리 수정이 포함됩니다. 어플리케이션을 모바일 장치에 설치하면, 코드 및 데이터 리소스가 해당 모바일 장치에 존재합니다. 공격자는 코드를 직접 수정하거나, 메모리 내용을 동적으로 변경하거나, 어플리케이션이 사용하는 시스템 API를 변경 및 대체하거나, 어플리케이션의 데이터와 자원을 수정할 수 있습니다. 이것은 공격자에게 소프트웨어의 의도된 사용을 개인적 또는 금전적 이득을 위해 파괴하는 직접적인 방법을 제공할 수 있습니다.

M9 – 리버스 엔지니어링

이 항목에는 소스 코드, 라이브러리, 알고리즘 및 기타 자산을 결정하기 위한 최종 코어 바이너리의 분석이 포함됩니다. IDA Pro, Hopper, o'toole 및 기타 바이너리 검사 도구와 같은 소프트웨어는 공격자가 어플리케이션의 내부 동작을 파악할 수 있도록 합니다. 이는 백엔드 서버, 암호화 상수 및 암호 및 지적 재산에 대한 정보를 공개하는 것 외에도 어플리케이션의 초기 취약점을 악용하는 데 사용될 수 있습니다.

M10 – 불필요한 기능

종종 개발자는 숨겨진 백도어 기능 또는 프로덕션 환경으로 배포하지 않을 예정인 기타 내부 개발 보안 컨트롤을 포함합니다. 예를 들어 개발자가 실수로 하이브리드 앱에 댓글로 비밀번호를 포함시킬 수 있습니다. 또 다른 예는 테스트 중 2중 인증을 사용하지 못하도록 합니다.

 위험원	 공격방법	 보안약점		 기술적 영향도	 사업적 영향도
특정 어플리케이션	공격가능성 쉬움	취약점 분포 일반	탐지가능성 평균	영향도 심각	어플리케이션 / 특정 비즈니스
플랫폼 기능 오용 또는 플랫폼 보안 설정 사용 실패를 다룹니다. Android intent, 플랫폼 권한, TouchID 오용, 키 체인 또는 모바일 OS의 일부인 다른 보안 설정이 포함될 수 있습니다.	공격 벡터는 전통적인 OWASP Top 10을 통해 사용할 수 있는 공격 벡터와 동일합니다. 노출된 API 호출은 이 취약점의 공격 벡터로 사용될 수 있습니다	이 취약점으로 익스플로잇하려면, 모바일 앱에서 사용하는 웹 서비스 또는 API 호출이 노출되어야 합니다. 노출된 서비스 또는 API 호출은 취약한 코딩 기술로 구현되며, OWASP Top 10 항목으로 취약하게 구현되어 있어야 합니다. 모바일 인터페이스를 통해 공격자는 악의적인 입력이나 예기치 않은 이벤트 시퀀스를 취약한 엔드포인트에 제공할 수 있습니다. 따라서 공격자는 서버에 있는 OWASP Top 10 취약점을 인지합니다.		이 취약점의 기술적 영향은 상대방이 모바일 장치를 통해 악용된 취약점(OWASP Top 10에 정의) 기술적 영향에 해당합니다. 예를 들어, 공격자는 모바일 장치로 XSS 취약점을 익스플로잇 할 수 있습니다. 이는 OWASP Top 10 A3 - XSS에 해당하며 기술적 영향은 보통입니다	이 취약점의 비즈니스 영향은 상대방이 모바일 장치를 통해 악용된 취약점(OWASP Top 10에서 정의) 비즈니스 영향에 해당합니다. 예를 들어, 공격자는 모바일 장치로 XSS 취약점을 악용할 수 있습니다. 이는 OWASP Top 10 A3 - XSS 비즈니스 영향에 해당합니다.

취약점 확인

이 범주에서 위험 요소라고 불리는 특징은 플랫폼(iOS, Android, Windows Phone 등)이 문서화되고 잘 이해되는 성능 또는 기능을 제공합니다. 앱이 해당 기능을 사용하지 못하거나 잘못 사용합니다. 이는 설계 및 구현이 앱 개발자의 문제가 아니므로 모바일 Top 10 항목과 다릅니다. 모바일 앱의 이런 위험을 확인할 수 있는 몇 가지 방법이 있습니다.

- 1. 배포된 가이드 라인 위반.** 모든 플랫폼에는 보안 (c.f., ((Android)), ((iOS)), ((Windows Phone)))에 대한 개발 지침이 있습니다. 앱이 제조업체가 권장하는 모범 사례와 모순되면, 위험에 노출됩니다. 예를 들어, iOS Keychain을 사용하는 방법이나 Android의 exported 서비스 보호 방법에 대한 가이드 라인이 있습니다. 이 가이드 라인을 위반하는 앱은 이러한 위험에 처해집니다.
- 2. 관습 또는 일반 사례 위반.** 모든 모범 사례가 제조업체 지침을 만드는 것은 아닙니다. 경우에 따라서 모바일 앱에 흔히 발생하는 실제 모범 사례가 있습니다.
- 3. 의도하지 않은 오용.** 일부 앱은 작업을 빠르게 수행하려고 하지만 실제로 구현의 일부를 잘못 처리합니다. 이것은 API 호출에 잘못된 플래그를 설정하는 것과 같은 단순한 버그일 수 있습니다. 또는 보호 기능이 어떻게 작동하는지 오해가 될 수 있습니다.

보안대책

안전한 코딩 및 설정 방법은 모바일 어플리케이션 서버 측에서 사용해야 합니다. 특정 취약점 정보는 OWASP Web Top 10 또는 Cloud Top 10 프로젝트를 참조하십시오.

참고자료

OWASP

- [OWASP iOS Developer Cheat Sheet](#)

외부자료

- [Google Androids Developer Security Topics 1](#)

공격 시나리오 샘플

수백 또는 수천 개의 API가있는 플랫폼이 여러 개 있기 때문에 이 섹션의 예제는 가능하다면, 전체적으로 훑어볼 수 있습니다.

키 체인을 대신할 어플리케이션 저장소. iOS 키 체인은 앱과 시스템 데이터의 안전한 저장 장치입니다. iOS에서 앱은 보안이 필요한 작은 데이터(세션 키, 비밀번호, 기기 정보 등)를 저장하는 데 사용해야 합니다. 일반적으로 앱 로컬 저장소에 이런 항목들을 저장하는 실수를 저지릅니다. 앱 로컬저장소에 저장된 데이터는 암호화되지 않은 iTunes 백업(예: 사용자 컴퓨터)에 사용할 수 있습니다. 일부 가이드 라인에 따르지 않으면 취약할 수 있습니다. M1 취약점을 제거해야 할 많은 위험과 취약점 항목을 Cloud Top 10 Risk, OWASP Top 10에서 확인할 수 있습니다.

최악의 문제점

다음은 모바일 어플리케이션에서 가장 자주 볼 수 있는 OWASP 취약점 리스트입니다.(오른쪽 참고)

A. 불량한 웹서비스 고착화

논리적 오류

Testing for business logic flaws
Business Logic Security Cheat Sheet

취약한 인증

OWASP Top Ten Broken Authentication Section
Authentication Cheat Sheet
Developers Guide for Authentication
Testing for Authentication

취약한 세션 관리(또는 세션관리 없음)
고정된 세션
GET 방식으로 전송된 민감정보

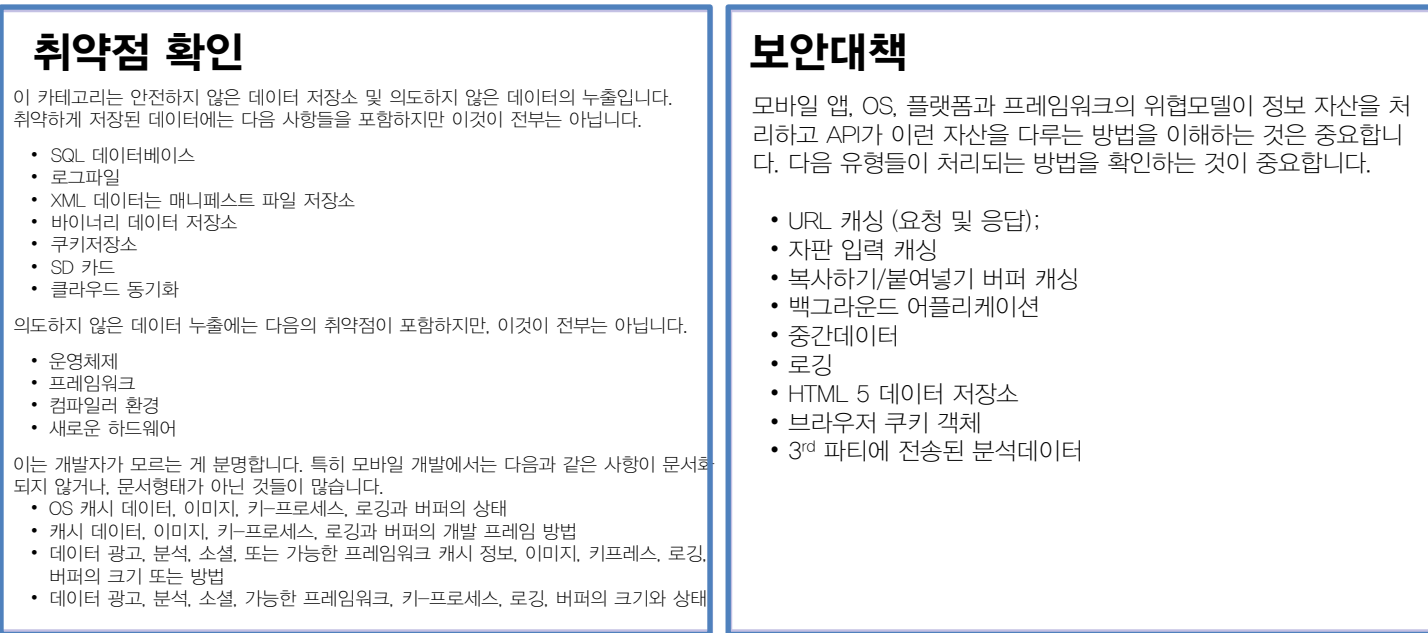
B. 안전하지 않은 웹서버 구성 - 기본컨텐츠, 관리자 인터페이스

C. 웹서비스 및 모바일 웹사이트에서 인젝션(SQL, XSS, Command)

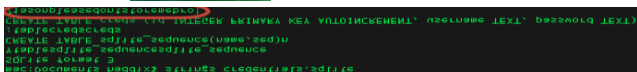
D. 인증 및 세션관리 취약점

E. 접근 제어 취약점

F. 로컬 및 원격 파일 업로드(LF/RFI)








Goat는 보안 커뮤니티를 위해 이러한 유형의 취약점을 처음으로 확인할 수 있도록 취약하게 만든 모방일 앱입니다. 아래 연설에서 각각 증명을 일컫고자 위조 은행 어플리케이션에 로그인합니다. 그리고 파일 시스템으로 이동합니다. applications 디렉토리에서 "credentials.sqlite"라는 DB를 볼 수 있습니다. 이 DB를 탐색하여 어플리케이션이 사용자 이름과 자격 증명 (Jason: pleasedontstoremebro!)를 평문으로 저장하고 있습니다.



- [OWASP IOS Developer Cheat Sheet](#)

- [Google Androids Developer Security Topics 1](#)
- [Google Androids Developer Security Topics 2](#)
- [Apple's Introduction to Secure Coding](#)
- [Fortify On Demand Blog – Exploring The OWASP Mobile Top 10: Insecure Data Storage](#)

 위협원	 공격방법	 보안약점		 기술적 영향도	 사업적 영향도
특정 어플리케이션	공격가능성 쉬움	취약점 분포 일반	탐지가능성 평균	영향도 심각	어플리케이션 / 특정 비즈니스
모바일 어플리케이션을 설계할 때 일반적으로 데이터는 클라이언트 - 서버 방식으로 교환됩니다. 솔루션이 데이터를 전송할 때 모바일 장치 이동은 통신사 네트워크와 인터넷을 통과해야 합니다. 위협원은 취약점을 악용하여 민감정보 전송 도중 차단할 수 있습니다. 다음 위협요소가 있습니다. <ul style="list-style-type: none">• 로컬 네트워크를 공유하는 침입자(손상되었거나 모니터링중인 Wi-Fi).• 캐리어/네트워크 장비(라우터, 셀 타워, 프록시 등)• 모바일 장치 악성코드	안전하지 않은 통신 범위에 대한 네트워크 모니터링의 악용 가능한 요소. 이동 통신사 네트워크를 통한 트래픽 모니터링은 카페의 트래픽 모니터링보다 어렵습니다. 일반적으로 타겟이 된 목표물에 공격하는 것이 더 쉽습니다.	모바일 어플리케이션은 빈번히 네트워크 트래픽을 보호하지 않습니다. 인증 중에는 SSL/TLS를 사용할 수 있지만 다른 곳에서는 사용할 수 없습니다. 일정치 않은 상황으로 인해 데이터 및 세션 ID가 차단될 위험이 있습니다. 전송구간 보안을 한다고 해서 앱이 올바르게 구현된 것은 아닙니다. 기본적인 결함을 탐지하려면 핸드폰 네트워크 트래픽을 관찰하십시오. 더 상세한 취약점은 어플리케이션 및 어플리케이션 UI 및 설정을 확인해야 합니다.	이 결함은 개별 사용자의 데이터를 노출하고 계정 도용으로 이어질 수 있습니다. 공격자가 관리자 계정을 가로채면 전체 사이트가 노출될 수 있습니다. 취약한 SSL 설정은 피싱 및 MITM 공격이 가능합니다.	영향도가 적더라도, 통신 채널을 통한 민감정보 가로채기는 개인정보 노출을 초래합니다. 사용자의 기밀성을 침해하면 다음과 같은 결과가 발생할 수 있습니다. <ul style="list-style-type: none">• 신분도용 및 오용• 명예훼손	

취약점 확인

이 위험은 데이터를 A에서 B로 가져오는 모든 측면을 다루지만 안전하지 않습니다. 여기에는 모바일에서 모바일로 통신, 어플리케이션 간 통신 또는 모바일에서 다른 쪽의 통신을 포함됩니다. 이 위험 요소에 모바일 장치에서 사용할 수 있는 모든 통신 기술(TCP/ IP, WiFi, Bluetooth/ Bluetooth-LF, NFC, 오디오, 적외선, GSM, 3G, SMS 등)이 포함됩니다.

모든 TLS 통신 문제가 이 취약점에 있습니다. NFC, 블루투스 및 WiFi 문제도 마찬가지입니다.

주목할만한 점은 민감정보를 포장하여 장치 내부 또는 외부로 전송하는 것입니다. 중요 정보의 예로 암호화 키, 암호, 개인 사용자 정보, 계정 정보, 세션 토큰, 문서, 메타데이터 및 바이너리가 있습니다. 민감정보는 서버에서 기기로 전송되는 경우, 앱에서 서버로 전송되는 경우, 기기와 로컬(예: NFC 단말 또는 NFC 카드)간에 전송될 수 있습니다.

이 위험 요소의 특징은 두 개의 장치가 있고 그 사이에 일부 데이터가 전달된다는 것입니다. 데이터가 기기 자체에 로컬로 저장되는 경우는 #취약한 정보입니다. 세션 세부 정보(예: 강력한 TLS 연결로) 안전하게 전달되지만 세션 식별자 자체가 취약한 경우(예측 가능, 낮은 엔트로피 등), 통신 문제가 아닌 #취약한 인증 문제입니다.

안전하지 않은 통신의 일반적인 위험은 데이터 무결성, 데이터 기밀성 및 원본 무결성과 관련됩니다. 전송 중 데이터가 변경되어(예: 중간자 공격을 통해) 변경 사항을 감지할 수 없는 경우, 이는 취약점 통신의 좋은 예입니다. 통신수단을 감시하여 기밀정보를 유출, 생산, 습득(예: 도청)하거나 대화 내용을 기록하여, 추후에 공격(오프라인 공격)하는 경우 또한 취약한 통신 문제입니다. TLS 연결(예: 인증서 확인, 취약한 암호, 기타 TLS 구성 문제)을 올바르게 설정하고 유효성을 검사하지 못하면 모두 취약한 통신이 됩니다.

공격 시나리오 샘플

모바일 앱의 통신 보안을 검사할 때, 침투 테스터가 자주 발견하는 몇 가지 일반적인 시나리오가 있습니다.

인증서 검사 부족. 모바일 앱과 엔드포인트는 TLS 핸드셰이크를 성공적으로 연결하고 수행하여 보안 채널을 설정합니다. 그러나 모바일 앱은 서버가 제공한 인증서를 검사하지 못하고 모바일 앱은 서버에서 제공한 인증서를 무조건 받아들입니다. 이는 모바일 앱과 엔드 포인트 간의 상호 인증 기능을 파괴합니다. 모바일 앱은 TLS 프로кси를 통한 중간자 공격(man-in-the-middle attack)에 취약합니다.





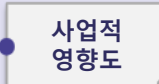
악한 핸드셰이크 협상. 모바일 앱과 엔드포인트 연결은 핸드셰이크의 일부로 암호 모음(cipher suite)에 성공적으로 연결하고 협상합니다. 클라이언트는 악한 암호 모음을 사용하기 위해 서버와 협상을 성공적으로 수행합니다. 악한 암호 모음은 공격자가 쉽게 복호화할 수 있는 취약한 암호를 발생시킵니다. 이는 모바일 앱과 엔드포인트 간 채널 기밀성을 위태롭게 합니다.

개인정보유출. 모바일 앱은 개인 식별 정보를 SSL대신 비보안 채널을 통해 엔드포인트로 전송합니다. 이는 모바일 앱과 엔드 포인트간 개인정보 관련 데이터 기밀성을 위태롭게 합니다.

참고자료

OWASP & 외부자료

- 홈페이지(www.owasp.org) 참조

 위협원	 공격방법	 보안약점		 기술적 영향도	 사업적 영향도
특정 어플리케이션	공격가능성 쉬움	취약점 분포 일반	탐지가능성 평균	영향도 심각	어플리케이션 / 특정 비즈니스
<p>모바일 어플리케이션을 설계할 때 일반적으로 데이터는 클라이언트 - 서버 방식으로 교환됩니다. 솔루션이 데이터를 전송할 때 모바일 장치 이동은 통신사 네트워크와 인터넷을 통과해야 합니다. 위협원은 취약점을 악용하여 민감정보 전송 도중 차단할 수 있습니다. 다음 위협요소가 있습니다.</p> <ul style="list-style-type: none"> • 로컬 네트워크를 공유하는 침입자(손상되었거나 모니터링중인 Wi-Fi). • 캐리어/네트워크 장비(라우터, 셀 타워, 프록시 등) • 모바일 장치 악성코드 	<p>안전하지 않은 통신 범위에 대한 네트워크 모니터링의 악용 가능한 요소. 이동 통신사 네트워크를 통한 트래픽 모니터링은 카페의 트래픽 모니터링보다 어렵습니다. 일반적으로 타겟이 된 목표물에 공격하는 것이 더 쉽습니다.</p>	<p>모바일 어플리케이션은 빈번히 네트워크 트래픽을 보호하지 않습니다. 인증 중에는 SSL/TLS를 사용할 수 있지만 다른 곳에서는 사용할 수 없습니다. 일정치 않은 상황으로 인해 데이터 및 세션 ID가 차단될 위험이 있습니다. 전송구간 보안을 한다고 해서 앱이 올바르게 구현된 것은 아닙니다. 기본적인 결함을 탐지하려면 핸드폰 네트워크 트래픽을 관찰하십시오. 더 상세한 취약점은 어플리케이션 및 어플리케이션 UI 및 설정을 확인해야 합니다.</p>		<p>이 결함은 개별 사용자의 데이터를 노출하고 계정 도용으로 이어질 수 있습니다. 공격자가 관리자 계정을 가로채면 전체 사이트가 노출될 수 있습니다. 취약한 SSL 설정은 피싱 및 MITM 공격이 가능합니다.</p>	<p>영향도가 적더라도, 통신 채널을 통한 민감정보 가로채기는 개인정보 노출을 초래합니다. 사용자의 기밀성을 침해하면 다음과 같은 결과가 발생할 수 있습니다.</p> <ul style="list-style-type: none"> • 신분도용 및 오용 • 명예훼손

보안대책

일반 모범 사례

- 네트워크 계층이 안전하지 않고 도청의 위험이 있다고 가정합니다.
- 모바일 앱이 민감한 정보, 세션 토큰 또는 기타 중요한 데이터를 백엔드 API 또는 웹 서비스로 전송하는 데 사용할 채널을 전송하기 위해 SSL/TLS를 적용합니다.
- 어플리케이션이 브라우저/웹킷을 통해 일상적으로 실행될 때 SSL 버전을 사용하여 타사 분석 회사, 소셜 네트워크와 같은 외부 실체를 고려합니다.
- 어플리케이션이 일반적으로 브라우저/웹킷으로 실행될 때, SSL을 통해 3rd 파티 분석 회사, 소셜 네트워크 등과 같은 아웃사이드 엔터티를 고려합니다.
- 적절한 키 길이를 가진 강력한 업계 표준 암호 제품군을 사용하십시오.
- 신뢰할 수 있는 CA 공급자가 서명한 인증서를 사용하십시오.
- 자체 서명 인증서를 절대로 허용하지 말고, 어플리케이션 보안이 되어있는 인증서만 사용하십시오..
- 항상 SSL 체인을 확인하십시오.
- 키 체인에서 신뢰할 수 있는 인증서를 사용하여 엔드포인트 서버의 ID를 검증한 후에 보안 연결을 설정하십시오.
- 모바일 앱이 잘못된 인증서를 발견하면 UI를 통해 사용자에게 경고합니다.
- 민감정보를 다른 채널(예 : SMS, MMS 또는 알림)을 통해 보내지 마십시오.
- 가능하다면, SSL로 제공되기 전 민감정보에 별도의 암호화를 적용하십시오.
- SSL 구현에서 향후 취약점이 발견되는 경우 암호화된 데이터는 기밀성 위반에 대한 보조 방어 수단을 제공합니다.

최신 위협은 모바일 장치의 SSL 라이브러리가 암호화하여 네트워크 트래픽을 대상 서버로 전송하기 직전, 모바일 장치 내의 트래픽을 가로채서 공격자가 중요한 트래픽을 도청할 수 있게 합니다.

보안대책

iOS 모범 사례

최신 버전의 iOS의 기본 클래스는 SSL 암호화 길이 협상을 잘 처리합니다. 개발자가 개발 장애물을 수용하기 위해 이러한 기본값을 무시하는 코드를 임시로 추가할 때 문제가 발생합니다. 일반 사례를 제외한 모범 사례는 다음과 같습니다.

- 인증서가 유효하고 Fail close를 확인하십시오.
- CFNetwork를 사용할 때는 보안 전송 API를 사용하여 신뢰할 수 있는 클라이언트 인증서를 지정하는 것이 좋습니다. 거의 모든 상황에서 `NSURLSessionSecurityLevelTLSv1`은보다 높은 표준 암호 강도에 사용해야 합니다.
- 개발 후 모든 NSURL 호출(또는 NSURL의 래퍼)이 NSURL 클래스 메서드 `setAllowsAnyHTTPSCertificate`와 같은 자체 서명되었거나 유효하지 않은 인증서를 허용하지 않도록 하십시오.
- 다음을 수행하여 인증서를 고정하십시오. 인증서를 내보내 앱 번들에 포함시키고 신뢰한 오브젝트에 고정시킵니다.
- NSURL 메소드 연결 사용 : `willSendRequestForAuthenticationChallenge`: 이제 인증서를 수락합니다.

안드로이드 모범 사례

- 어플리케이션이 `org.apache.http.conn.ssl.AllowAllHostnameVerifier` 또는 `SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER`와 같은 모든 인증서를 허용할 수 있도록 개발주기 이후 모든 코드를 제거하십시오. 이는 모든 인증서를 신뢰하는 것과 같은 의미입니다.

- `SSLSocketFactory`를 확장한 클래스를 사용하는 경우, `checkServerTrusted` 메소드가 올바르게 구현되어 서버 인증서가 올바르게 검사되는지 확인하십시오.

위협원	공격방법	보안약점		기술적 영향도	사업적 영향도
특정 어플리케이션	공격가능성 쉬움	취약점 분포 일반	탐지가능성 평균	영향도 심각	어플리케이션 / 특정 비즈니스
인증 취약점을 악용하는 위협원은 일반적으로 사용 가능한 도구 또는 사용자 정의 도구를 사용하는 자동 공격을 통해 이를 수행합니다.	공격자가 인증 스키마가 취약한 것을 이해하면 공격자는 모바일 앱의 백엔드 서버에 서비스 요청하고 모바일 앱과의 직접적인 상호 작용을 우회하여 인증을 우회합니다. 이 서브미션 프로세스는 일반적으로 장치 내 모바일 멀웨어 또는 공격자가 소유한 봇넷으로 수행됩니다.	<p>불량하거나 누락된 인증 방식을 사용하면 공격자들이 익명으로 모바일 앱 또는 모바일 앱에서 사용하는 백엔드 서버 내에서 기능을 실행할 수 있습니다. 휴대 기기의 입력 폼 팩터 때문에 모바일 앱 인증이 취약합니다. 폼 팩터는 종종 단순한 4자리 PIN 기반으로 하는 짧은 암호를 권장합니다.</p> <p>모바일 어플리케이션의 인증 요구 사항은 가용성 요구 사항으로 기존 웹 인증 방식과 상당히 다를 수 있습니다.</p> <p>기존의 웹 어플리케이션에서 사용자는 백엔드 서버로 실시간 온라인 인증을 받을 수 있습니다. 이 세션으로 인터넷에 지속적으로 액세스할 수 있다는 합리적 기대가 있습니다.</p> <p>모바일 앱에서 사용자는 세션이 유지되어 항상 온라인 상태가 될 것으로 생각하지 않습니다. 모바일 인터넷 연결은 전통적인 웹 연결보다 훨씬 신뢰성이 낮거나 예측 가능합니다. 따라서 모바일 앱에는 오프라인 인증이 필요한 가동 시간 요구 사항이 있을 수 있습니다. 이 오프라인 요구 사항은 모바일 인증을 구현할 때 개발자가 고려해야 할 사항에 중대한 파급 효과가 있을지도 모릅니다.</p> <p>취약한 인증 방식을 탐지하기 위해 테스트는 모바일 앱이 '오프라인'모드 일 때, 바이너리 공격을 수행할 수 있습니다. 이 공격을 통해 테스트는 앱이 오프라인 인증을 거치지 않고 오프라인 인증이 필요한 기능을 실행합니다 (바이너리 공격에 대한 자세한 내용은 M10 참조). 또한 테스트는 모바일 앱 기능에 대한 POST/GET 요청에서 세션 토큰을 제거하여 익명으로 모든 백엔드 서버 기능을 실행해야 합니다.</p>	잘못된 인증의 기술적 인 영향은 솔루션이 작업 요청을 수행하는 사용자를 식별할 수 없다는 것입니다. 즉시 사용자의 신원을 확인할 수 없으므로 솔루션에서 사용자 활동을 기록하거나 감시할 수 없습니다. 이것은 공격의 원천, 근본적인 악용의 본질 또는 미래의 공격을 방지하는 방법을 발견하지 못하는 데 기여할 것입니다.	인증 실패로 인해 권한 부여 실패로 노출될 가능성이 내재되어 있습니다. 인증 제어가 실패하면 솔루션은 사용자의 신원을 확인할 수 없습니다. 이 ID는 사용자의 역할 및 관련 권한에 연결됩니다. 침입자가 중요한 기능을 익명으로 실행할 수 있는 경우, 이는 내재된 코드가 이 행위에 대한 요청을 발한 사용자 권한을 확인할 수가 없다. 따라서 익명으로 코드를 실행하면 인증 및 권한 제어에서 실패할 수 밖에 없습니다.	취약한 인증의 비즈니스 영향은 일반적으로 다음과 같은 결과를 가져옵니다. <ul style="list-style-type: none"> • 명예훼손 • 정보 절도 • 권한없는 데이터 접속

취약점 확인

모바일 앱에 안전하지 않은 인증이 적용될 수 있는 여러 방법이 있습니다.

- 모바일 앱이 액세스 토큰을 제공하지 않고 익명으로 백엔드 API 서비스 요청을 실행할 수 있는 경우 이 애플리케이션은 안전하지 않은 인증을 받습니다.
- 모바일 앱이 기기에 로컬로 비밀번호나 공유 비밀을 저장하는 경우 보안되지 않은 인증이 가장 자주 발생합니다.
- 모바일 앱이 취약한 비밀번호 정책을 사용하여 비밀번호를 간단히 입력하는 경우 안전하지 않은 인증이 적용됩니다. 또는
- 모바일 앱이 TouchID와 같은 기능을 사용하는 경우 불안정한 인증이 적용됩니다.

공격 시나리오 샘플

다음 시나리오는 모바일 앱에서 취약한 인증 또는 권한 제어를 보여줍니다.

시나리오#1: 숨겨진 서비스 요청: 개발자는 인증 된 사용자만 모바일 앱이 처리를 위해 백엔드로 제출하는 서비스 요청을 생성할 수 있다고 가정합니다. 요청을 처리하는 동안 서버 코드는 들어오는 요청이 알려진 사용자와 연관되어 있는지 확인하지 않습니다. 따라서 공격자들은 백엔드 서비스에 서비스 요청을 제출하고 솔루션의 합법적인 사용자에게 영향을 주는 기능을 익명으로 실행합니다.

시나리오#2: 인터페이스 의존성: 개발자는 승인된 사용자만 모바일 앱에서 특정 기능을 볼 수 있다고 가정합니다. 따라서 정당한 권한을 가진 사용자만 모바일 장치에서 서비스에 대한 요청을 발급 할 수 있을 것으로 예상합니다. 요청을 처리하는 백엔드 코드는 요청과 관련된 ID에 서비스를 실행할 권한이 있는지 확인합니다. 따라서 공격자는 상당히 낮은 권한이 있는 사용자 계정을 사용하여 원격 기능을 실행할 수 있습니다.

시나리오#3: 사용 요구 사항: 사용 요구 사항으로 인해 모바일 앱은 4자리 암호를 허용합니다. 서버 코드는 해시된 암호 버전을 올바르게 저장합니다. 그러나 암호의 길이가 매우 짧기 때문에 공격자는 레인보우 해시 테이블을 사용하여 원래 암호를 빠르게 추론할 수 있습니다. 서버의 암호 파일 (또는 데이터 저장소)이 유출된 경우 공격자는 사용자의 암호를 신속하게 추측할 수 있습니다.

위협원	공격방법	보안약점		기술적 영향도	사업적 영향도
특정 어플리케이션	공격가능성 쉬움	취약점 분포 일반	탐지가능성 평균	영향도 심각	어플리케이션 / 특정 비즈니스
인증 취약점을 악용하는 위협원은 일반적으로 사용 가능한 도구 또는 사용자 정의 도구를 사용하는 자동 공격을 통해 이를 수행합니다.	공격자가 인증 스키마가 취약한 것을 이해하면 공격자는 모바일 앱의 백엔드 서버에 서비스 요청하고 모바일 앱과의 직접적인 상호 작용을 우회하여 인증을 우회합니다. 이 서버미션 프로세스는 일반적으로 장치 내 모바일 멀웨어 또는 공격자가 소유한 봇넷으로 수행됩니다.	<p>불량하거나 누락된 인증 방식을 사용하면 공격자들이 익명으로 모바일 앱 또는 모바일 앱에서 사용하는 백엔드 서버 내에서 기능을 실행할 수 있습니다. 휴대 기기의 입력 폼 팩터 때문에 모바일 앱 인증이 취약합니다. 폼 팩터는 종종 단순한 4자리 PIN 기반으로 하는 짧은 암호를 권장합니다.</p> <p>모바일 어플리케이션의 인증 요구 사항은 가용성 요구 사항으로 기존 웹 인증 방식과 상당히 다를 수 있습니다.</p> <p>기존의 웹 어플리케이션에서 사용자는 백엔드 서버로 실시간 온라인 인증을 받을 수 있습니다. 이 세션으로 인터넷에 지속적으로 액세스할 수 있다는 합리적 기대가 있습니다.</p> <p>모바일 앱에서 사용자는 세션이 유지되어 항상 온라인 상태가 될 것으로 생각하지 않습니다. 모바일 인터넷 연결은 전통적인 웹 연결보다 훨씬 신뢰성이 낮거나 예측 가능합니다. 따라서 모바일 앱에는 오프라인 인증이 필요한 가동 시간 요구 사항이 있을 수 있습니다. 이 오프라인 요구 사항은 모바일 인증을 구현할 때 개발자가 고려해야 할 사항에 중대한 파급 효과가 있을지도 모릅니다.</p> <p>취약한 인증 방식을 탐지하기 위해 테스트는 모바일 앱이 '오프라인'모드 일 때, 바이너리 공격을 수행할 수 있습니다. 이 공격을 통해 테스트는 앱이 오프라인 인증을 거치지 않고 오프라인 인증이 필요한 기능을 실행합니다 (바이너리 공격에 대한 자세한 내용은 M10 참조). 또한 테스트는 모바일 앱 기능에 대한 POST/GET 요청에서 세션 토큰을 제거하여 익명으로 모든 백엔드 서버 기능을 실행해야 합니다.</p>	잘못된 인증의 기술적 인 영향은 솔루션이 작업 요청을 수행하는 사용자를 식별할 수 없다는 것입니다. 즉시 사용자의 신원을 확인할 수 없으므로 솔루션에서 사용자 활동을 기록하거나 감시할 수 없습니다. 이것은 공격의 원천, 근본적인 악용의 본질 또는 미래의 공격을 방지하는 방법을 발견하지 못하는 데 기여할 것입니다.	인증 실패로 인해 권한 부여 실패로 노출될 가능성이 내재되어 있습니다. 인증 제어가 실패하면 솔루션은 사용자의 신원을 확인할 수 없습니다. 이 ID는 사용자의 역할 및 관련 권한에 연결됩니다. 침입자가 중요한 기능을 익명으로 실행할 수 있는 경우, 이는 내재된 코드가 이 행위에 대한 요청을 발한 사용자 권한을 확인할 수가 없습니다. 따라서 익명으로 코드를 실행하면 인증 및 권한 제어에서 실패할 수 밖에 없습니다.	취약한 인증의 비즈니스 영향은 일반적으로 다음과 같은 결과를 가져옵니다.
					<ul style="list-style-type: none">• 명예훼손• 정보 절도• 권한없는 데이터 접속

보안대책

취약한 패턴 회피



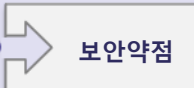

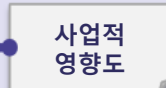
다음과 같은 안전하지 않은 모바일 어플리케이션 인증 UI 패턴을 피하십시오.

- 웹 어플리케이션을 해당 모바일 환경으로 이식하는 경우 모바일 어플리케이션의 인증 요구 사항이 웹 어플리케이션 구성 요소의 요구 사항과 일치해야 합니다. 따라서 웹 브라우저보다 적은 인증 요소로 인증할 수 없어야 합니다.
- 로컬에서 사용자를 인증하면 클라이언트 측 바이패스 취약점이 발생할 수 있습니다. 어플리케이션이 데이터를 로컬로 저장하는 경우 런타임 루틴을 조작하거나 바이너리를 수정하여 탈옥 장치에서 인증 루틴을 생략할 수 있습니다. 오프라인 인증을 위한 강력한 비즈니스 요구 사항이 있는 경우 M10에서 모바일 앱에 대한 바이너리 공격 보안대책에 대한 추가 지침을 참조하십시오.
- 가능하면 모든 인증 요청이 서버 측에서 수행되는지 확인하십시오. 인증에 성공하면 어플리케이션 데이터가 모바일 장치로 로드됩니다. 이렇게하면 성공적인 인증 후 어플리케이션 데이터를 사용할 수 있습니다.
- 클라이언트 측의 데이터 저장소가 필요한 경우 사용자의 로그인 자격 증명에서 안전하게 파생된 암호화 키를 사용하여 데이터를 암호화해야 합니다.

- 바이너리 공격을 통해 데이터가 복호화되는 추가적인 위험이 있습니다. 로컬 데이터 유출로 이어지는 바이너리 공격 보안대책에 대한 추가 지침은 M9를 참조하십시오.
- 모바일 어플리케이션 내에서 구현되는 영구 인증(Remember Me) 기능은 사용자의 암호를 절대로 장치에 저장하지 않아야 합니다.
- 이상적으로 모바일 어플리케이션은 사용자가 모바일 어플리케이션 내에서 폐기할 수 있는 장치별 인증 토큰을 사용해야 합니다. 이렇게하면 앱이 도난 당하거나 분실된 기기에서 무단 접근을 방지할 수 있습니다.
- 사용자를 인증할 때 스푸핑될 수 있는 값을 사용하지 마십시오. 여기에는 기기 식별자 또는 지리적 위치가 포함됩니다.
- 모바일 어플리케이션 내의 영구 인증은 Opt-in으로 구현되어야 하며 기본적으로 활성화되어서는 안됩니다.
- 가능하면 사용자가 인증암호에 4자리 암호를 허용하지 마십시오.

취약한 패턴 회피

- 개발자는 모든 클라이언트 측 인증 및 인증 제어가 악의적인 사용자에게 의해 우회될 수 있다고 가정해야 합니다. 가능한 경우 서버 측에서 권한 부여 및 인증 제어를 다시 시행해야 합니다.

 위협원	 공격방법	 보안약점		 기술적 영향도	 사업적 영향도
특정 어플리케이션	공격가능성 평균	취약점 분포 일반	탐지가능성 평균	영향도 심각	어플리케이션 / 특정 비즈니스
위협원에는 부적절하게 암호화된 데이터에 대한 물리적 접근 권한을 가진 사람 또는 공격자를 대신하여 행동하는 모바일 멀웨어가 포함됩니다.	공격 벡터는 다음을 포함합니다. • 장치 또는 네트워크 트래픽 캡처에 대한 물리적 액세스 또는 암호화된 데이터에 대한 접근 권한이 있는 악의적인 어플리케이션을 통한 데이터 복호화	이 약점을 악용하려면 공격자는 암호화 프로세스 내 약한 암호화 알고리즘 또는 결함으로 인해 암호화된 코드 또는 중요한 데이터를 원래의 암호화되지 않은 형식으로 반환해야 합니다.		이 취약점으로 인해 모바일 장치에서 중요한 정보를 무단으로 검색할 수 있습니다.	이 취약점은 여러 비즈니스 영향을 미칠 수 있습니다. 일반적으로 암호화가 깨지면 다음과 같은 결과가 발생합니다. • 개인정보침해 • 정보/코드 유출 • 지적재산 도용 • 명예 훼손

취약점 확인

암호화를 사용하는 대부분의 모바일 앱에서 안전하지 않은 암호화 사용이 일반적입니다. 모바일 앱에 표시되는 취약한 암호화 기본적인 두 가지 방법이 있습니다. 첫째, 모바일 앱은 근본적으로 결함이 있는 암호화/복호화 프로세스를 사용할 수 있으며, 민감한 데이터를 복호화하기 위해 공격자가 악용할 수 있습니다. 둘째, 모바일 앱은 본질적으로 취약하고 공격자에 의해 직접 복호화 될 수 있는 암호화/복호화 알고리즘을 구현하거나 활용할 수 있습니다. 다음 하위 섹션에서는 두 개의 시나리오 모두 자세히 살펴봅니다.

내장된 코드 암호화 프로세스 의존도

기본적으로 iOS 어플리케이션은 코드 암호화를 통해 리버스 엔지니어링에서 보호됩니다(이론적으로). iOS 보안 모델은 탈옥되지 않은 환경에서 실행하기 위해 앱을 신뢰할 수 있는 소스로 암호화하고 서명해야 합니다. 시작시, iOS 앱 로더는 메모리에 있는 앱을 복호화하고 iOS에서 서명을 확인한 후 코드를 계속 실행합니다.

이 기능은 이론상 공격자가 iOS 모바일 앱에 대해 바이너리 공격을 수행하는 것을 방지합니다. ClutchMod 또는 GBD와 같이 무료로 사용할 수 있는 도구를 사용하면, 공격자가 암호화된 앱을 탈옥장치에 다운로드 하고 iOS 로더가 메모리에 로드하여 복호화 후(로더 실행 직전) 복호화된 앱의 스냅샷을 가져옵니다. 공격자가 스냅샷을 디스크에 저장하면 공격자는 IDA Pro 또는 Hopper와 같은 도구를 사용하여 어플리케이션 램 정적/동적 분석을 수행하고 더 많은 바이너리 공격을 쉽게 수행할 수 있습니다.

기본으로 제공된 암호화 알고리즘을 우회하는 것은 아주 사소한 일입니다. 항상 상대방이 기본 모바일 OS에서 제공하는 내장코드 암호화 우회할 수 있다고 가정합니다. 리버스 엔지니어링 방식에 대한 추가 계층을 제공하기 위해 할 수 있는 자세한 내용은 M9를 참조하십시오.

A. 취약한 키 관리 프로세스

키를 잘못 관리하면 최고의 알고리즘은 중요하지 않습니다. 많은 사람들이 올바른 암호화 알고리즘을 사용하기 위해 자체 프로토콜을 구현하는 실수를 저지릅니다. 문제의 예는 다음과 같습니다.

- 암호화된 콘텐츠와 동일한 공격자가 읽을 수 있는 디렉터리에 키를 포함시킵니다.
- 그렇지 않으면 공격자가 사용할 수 있는 키를 제작합니다.
- 바이너리 내에서 하드 코딩된 키를 사용하지 마십시오. 그리고
- 바이너리 공격으로 키가 가로챌 수 있습니다. 바이너리 공격 방식에 대한 자세한 내용은 M10을 참조하십시오.

B. 커스텀 암호화 프로토콜 제작 및 사용

자신의 암호화 알고리즘이나 프로토콜을 만들어 사용하려는 것보다 모바일을 비롯한 다른 방법으로 암호화를 잘못 다루는 쉬운 방법은 없습니다.

보안 커뮤니티에서 강력히 추천하는 최신 알고리즘을 항상 사용하고 가능하다면, 모바일 플랫폼에서 최신 암호화 API를 활용하십시오. 바이너리 공격은 바이너리에서 하드 코딩된 키와 함께 사용하는 공용 라이브러리를 공격자에게 알려줄 수 있습니다. 암호화 관련 보안 요구 사항이 매우 높은 경우에는 화이트 박스 암호화 사용을 강력히 고려해야 합니다.

공통 라이브러리의 악용으로 이어질 수 있는 바이너리 공격 보안 대책의 자세한 내용은 M10을 참조하십시오.

C. 취약한 알고리즘 및 미검증 알고리즘 사용

많은 암호화 알고리즘 및 프로토콜은 현저한 약점을 보이거나 현 대적인 보안 요구 사항에 충분하지 않기 때문에 사용해서는 안됩니다. 여기에는 다음이 포함됩니다.

- RC2
- MD4
- MD5
- SHA1



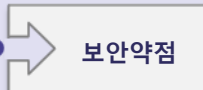

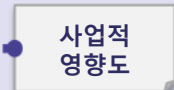
참고자료

OWASP

- [OWASP Cryptographic Storage Cheat Sheet](#)
- [OWASP Key Management Cheat Sheet](#)

외부자료

- [NIST Encryption Guidelines](#)

 위협원	 공격방법	 보안약점		 기술적 영향도	 사업적 영향도
특정 어플리케이션	공격가능성 평균	취약점 분포 일반	탐지가능성 평균	영향도 심각	어플리케이션 / 특정 비즈니스
권한 부여 취약점을 악용하는 위협원은 일반적으로 사용 가능한 도구 또는 사용자 정의 도구를 사용하는 자동 공격을 통해 이를 수행합니다.	공격자가 권한 부여 체계의 취약점을 이해하면 공격자는 합법적인 사용자로 어플리케이션에 로그인합니다. 그들은 성공적으로 인증을 받으면 취약한 엔드포인트에 강제로 찾아가서 관리 기능을 실행합니다. 이 서버에선 프로세스는 일반적으로 장치 내의 모바일 멀웨어 또는 공격자가 소유한 봇넷으로 수행됩니다.	낮은 권한 부여 스키마를 테스트하기 위해 테스트는 모바일 앱에 대해 바이너리 공격을 수행하고 모바일 앱이 '오프라인'모드 일 때, 더 높은 권한을 가진 사용자만 실행할 수 있는 특정 기능을 실행하려고 합니다(바이너리 공격에 대한 자세한 내용은 M9 및 M10 참조). 또한 테스트는 백엔드 서버에 중요한 기능에 대한 해당 POST/GET 요청 내에서 낮은 권한의 세션 토큰을 사용하여 권한있는 기능을 실행해야 합니다. 권한 부여 스키마가 잘못되거나 누락되면 공격자는 인증되었지만 권한이 낮은 모바일 앱 사용자는 사용할 수 없는 기능을 실행할 수 있습니다. 인증 요구 사항은 원격 서버가 아닌 모바일 장치에서 권한 결정을 내릴 때 더욱 취약합니다. 이는 오프라인 사용에 대한 모바일 요구 사항 때문입니다.	잘못된 인증의 기술적 영향은 특성상 빈약한 인증의 기술적 영향과 유사합니다. 기술 영향은 본질적으로 광범위하며 실행되는 특권이 부여된 기능 특성에 따라 달라질 수 있습니다. 예를 들어, 원격 또는 로컬 관리 기능 권한이 초과되어 시스템이 손상되거나 중요한 정보에 접근할 수 있습니다.	사용자 (익명 또는 확인된 사용자)가 권한을 초과하여 기능을 실행할 수 있는 경우 비즈니스는 다음과 같은 영향을 받을 수 있습니다. <ul style="list-style-type: none">• 명예훼손• 사기• 정보유출	

취약점 확인

인증(Authentication)과 권한부여(Authorization)의 차이를 인식하는 것이 중요합니다. 인증은 개인을 식별하는 행위입니다. 권한부여는 식별된 개인이 행위를 수행하는 데 필요한 권한이 있는지 확인하는 행위입니다. 인증 확인은 항상 모바일 장치에서 들어오는 요청 인증에 따라야 하므로 이 두 항목은 밀접한 관련이 있습니다.

조직이 모바일 장치에서 요청한 API 엔드포인트를 실행하기 전 인증을 받지 못하고 개인이 실패하면, 코드는 자동으로 안전하지 않은 권한이 적용됩니다. 호출자의 신원이 확인되지 않은 경우 들어오는 요청을 권한 부여 검사는 본질적으로 불가능합니다.

모바일 엔드포인트가 안전하지 않은 권한을 가지고 있는지 확인할 때, 따라야 할 몇 가지 간단한 규칙이 있습니다.

- **안전하지 않은 직접 객체 참조(IDOR) 취약점의 존재.** 안전하지 않은 직접 객체 참조 취약점(IDOR)이 표시되는 경우 코드는 유효한 권한 부여 검사를 수행하지 않을 가능성이 큼니다. 그리고
- **숨겨진 엔드 포인트.** 일반적으로 개발자는 숨겨진 기능이 올바른 역할의 사용자만 볼 수 있다고 가정하므로 백엔드 히든 기능에 대한 권한 검사를 수행하지 않습니다.
- **사용자 역할 또는 권한 전송.** 모바일 앱의 요청 일부로 백엔드 시스템에 사용자의 역할 또는 권한을 전송하는 경우 안전하지 않은 권한이 부여됩니다.

공격 시나리오 샘플

시나리오 #1: 취약한 직접 개체 참조

사용자는 액터 ID와 OAuth 베어러 토큰을 포함하는 백엔드 REST API에 대한 API 엔드 포인트 요청을 작성합니다. 사용자는 액터 ID를 들어오는 URL 일부로 포함시키고 액세스 토큰을 요청 표준 헤더로 포함시킵니다. 백엔드는 베어러 토큰의 존재를 검증하지만, 베어러 토큰과 연관된 액터 ID의 유효성을 검증하지 못합니다. 결과적으로 사용자는 REST API 요청 일부로 액터 ID를 조정하고 다른 사용자의 계정 정보를 얻을 수 있습니다.

시나리오 #2: LDAP 를 전송

사용자는 사용자가 속한 LDAP 그룹의 목록을 포함하는 헤더와 함께 표준 OAuth 베어러 토큰을 포함하는 API 엔드 포인트 요청을 백엔드 REST API에 작성합니다. 백엔드 요청은 베어러 토큰의 유효성을 검사한 다음 들어오는 LDAP 그룹에서 올바른 그룹 구성원 자격을 검사하여 민감한 기능을 계속 수행합니다. 그러나 백엔드 시스템은 LDAP 그룹 구성원의 독립적인 검증을 수행하지 않고 대신 사용자로부터 들어오는 LDAP 정보를 사용합니다. 사용자는 수신 헤더와 보고서를 임의로 LDAP 그룹의 구성원으로 조정하고 관리 기능을 수행할 수 있습니다

보안대책

안전하지 않은 권한 검사를 방지하려면 다음을 수행하십시오.

- 백엔드 시스템에 포함 된 정보만을 사용하여 인증된 사용자의 역할 및 권한을 확인하십시오. 모바일 장치 자체에서 오는 모든 역할 또는 사용 권한 정보에 의존하지 마십시오.
- 백엔드 코드는 신원 정보와 함께 오는 요청(조작된 피연산자)과 관련된 모든 수신 식별자가 들어오는 신원과 일치 하는지를 독립적으로 검증해야 합니다.

참고자료

OWASP & 외부자료

- [홈페이지\(www.owasp.org\)](http://www.owasp.org) 참조

위협원	공격방법	보안약점	기술적 영향도	사업적 영향도
특정 어플리케이션	공격가능성 어려움	취약점 분포 일반	탐지가능성 어려움	영향도 적당
위협 에이전트에는 모바일 코드 내에서 수행된 메소드 호출에 신뢰할 수 없는 입력을 전달하는 엔티티가 포함됩니다. 이런 유형은 보안 문제가 꼭 아니더라도 보안 취약점을 유발할 수 있습니다. 예를 들어, 사파리의 구버전(코드 품질이 나쁜 취약점)의 버퍼오버 플로우로 인해 본격적인 탈옥 공격으로 높은 위험이 초래되었습니다.	공격자는 일반적으로 신중하게 작성한 정보를 피해자에게 제공하여 이 카테고리의 취약점을 악용합니다. 이러한 입력은 착취가 발생하는 모바일 장치 내에 상주하는 코드로 전달됩니다. 일반적인 유형의 공격은 메모리 누수 및 버퍼오버 플로우를 악용합니다.	코드 품질 문제는 대부분의 모바일 코드에서 널리 퍼져 있습니다. 좋은 소식은 대부분의 코드 품질 문제는 상당히 양심적이며 나쁜 프로그래밍 습관을 초래한다는 것입니다. 수동적인 코드 리뷰를 통해 이러한 유형의 문제를 탐지하는 것은 일반적으로 어렵습니다. 정적 분석을 사용하면 성능 분석 도구를 사용하는 것이 가능합니다. 이러한 유형의 도구는 일반적으로 메모리 누수, 버퍼오버 플로우 및 잘못된 프로그래밍 실수를 초래하는 등 조금 심각한 문제들을 확인합니다. 극단적인 낮은 지식과 전문 지식을 갖춘 해커는 이러한 유형의 문제를 효과적으로 활용할 수 있습니다. 일반적인 주요 목표는 모바일 코드의 주소 공간에서 외부 코드를 실행하는 것입니다.	이 범주에 속하는 대부분의 공격은 원격 서버 종점(모바일 장치 아님)에서 외부 코드 실행 또는 서비스 거부를 초래합니다. 그러나 버퍼 오버플로우/오버런이 모바일 장치 내에 존재하고 외부에서 입력을 패싱시킬 수 있는 경우 기술적 영향이 심각할 수 있으므로 수정해야 합니다.	비즈니스 영향은 익스플로잇에 따라 크게 다릅니다. 원격코드 실행되어 코드 품질 문제로 다음과 같은 영향을 미칠 수 있습니다 <ul style="list-style-type: none"> 정보유출 명예훼손 지적재산 도용 이 범주에 속하지 않는 덜 심각한 기타 기술 문제로 성능, 메모리 사용량/불량한 프런트 엔드 아키텍처가 저하될 수 있습니다.

취약점 확인

이것은 모바일 클라이언트의 코드 수준 구현 문제에 대한 포괄적인 문제입니다. 이는 서버 측 코딩 실수와 구별됩니다. 버퍼오버 플로우, 형식 문자열 취약성 및 모바일 장치에서 실행중인 일부 코드를 다시 작성하는 다양한 코드 수준의 실수와 같은 취약점의 위험을 파악합니다.

이는 일반적으로 프로그래밍 언어 자체(예 : Java, Swift, Objective C, JavaScript)를 참조하기 때문에 부적절한 플랫폼 사용과는 다릅니다. Webview 모바일 앱에서 C 또는 DOM 기반 XSS의 버퍼오버 플로우는 코드 품질 문제가 됩니다.

이 위험 요소의 주요 특징은 코드가 모바일 장치에서 실행되고 코드가 상당히 현지화된 방식으로 변경되어야 한다는 것입니다. 대부분의 위험을 수정하려면 코드를 변경해야 하지만 코드 품질의 경우 잘못된 API를 사용하거나 API를 안전하지 않게 사용하거나 안전하지 않은 언어 구문을 사용하거나 기타 코드 수준의 문제가 발생할 수 있습니다.

중요: 이것은 서버에서 실행되는 코드가 아닙니다. 이것은 모바일 장치 자체에서 실행되는 잘못된 코드를 포착하는 위험입니다.

공격 시나리오 샘플

시나리오 #1: 버퍼오버 플로우 예제

```
include <stdio.h>
```

```
int main(int argc, char **argv)
{
    char buf[8]; // buffer for eight characters
    gets(buf); // read from stdio (sensitive function!)
    printf("%s\n", buf); // print out data stored in buf
    return 0; // 0 as return value
}
```

이 예제에서는 이 페이지에서 가져온 버퍼오버 플로우를 보안하기 위해 gets 함수를 사용하지 않아야 합니다. 이것은 대부분의 정적 분석 도구가 코드 품질 문제로 보고하는 예제입니다.

보안대책






일반적으로 코드 품질 문제는 다음을 수행하여 피할 수 있습니다.

- 조직의 모든 사람이 동의하는 일관된 코딩 패턴을 유지합니다.
- 읽기 쉽고 문서화가 잘 된 코드를 작성하십시오.
- 버퍼를 사용할 때는 항상 들어오는 버퍼 데이터의 길이가 대상 버퍼의 길이를 초과하지 않는지 확인하십시오.
- 자동화를 통해 타사 정적 분석 도구를 사용하여 버퍼오버 플로우 및 메모리 누출을 식별합니다. 그리고
- 다른 '코드 품질' 문제보다 버퍼오버 플로우 및 메모리 부족을 우선적으로 해결하십시오.

참고자료

OWASP/외부자료

- 홈페이지 참조

 위험원	 공격방법	 보안약점		 기술적 영향도	 사업적 영향도
특정 어플리케이션	공격가능성 쉬움	취약점 분포 일반	탐지가능성 평균	영향도 심각	어플리케이션 / 특정 비즈니스
일반적으로 공격자는 타사 앱 스토어에서 호스팅되는 악성 앱을 통해 코드 수정을 악용합니다. 공격자는 피싱 공격을 통해 사용자를 속여 앱을 설치할 수도 있습니다.	일반적으로 공격자는 이 범주를 악용하기 위해 다음 작업을 수행합니다. <ul style="list-style-type: none">• 어플리케이션 프로그램 램 패키지의 코어 바이너리로 직접 바이너리 변경• 어플리케이션 패키지 안의 리소스를 직접 바이너리값으로 변경하십시오.• 시스템 API를 리다이렉션하거나 대체하여 악의적인 외부코드를 가로채고 실행합니다.	<p>어플리케이션의 수정된 품은 생각보다 훨씬 더 일반적입니다. 앱 스토어 내에서 승인되지 않은 버전의 모바일 앱을 감지하고 제거하는 데 기반을 둔 보안 업체가 있습니다. 코드 수정 탐지 문제를 해결하기 위한 접근법에 따라 조직은 승인되지 않은 코드 탐지 방법을 매우 성공적으로 제한할 수 있다.</p> <p>이 항목에는 바이너리 패치, 로컬 리소스 수정, 메서드 후킹, 메서드 변경 및 동적 메모리 수정이 포함됩니다.</p> <p>어플리케이션이 모바일 장치로 전달되면 코드 및 데이터 리소스가 해당 모바일 장치에 상주합니다. 공격자는 코드를 직접 수정하거나, 메모리 내용을 동적으로 변경하거나, 어플리케이션이 사용하는 시스템 API를 변경하거나 대체하거나, 어플리케이션의 정보와 리소스를 수정할 수 있습니다.</p> <p>이는 공격자에게 소프트웨어의 의도적으로 개인적 또는 금전적 이득을 위해 파괴하는 직접적인 방법을 제공할 수 있습니다.</p>	<p>코드 수정 영향은 수정 자체의 성격에 따라 다양할 수 있습니다. 일반적인 유형의 영향은 다음과 같습니다.</p> <ul style="list-style-type: none">• 미승인된 새 기능• 신분 도용 또는 사기	<p>코드 수정으로 인한 비즈니스 영향은 일반적으로 다음과 같습니다.</p> <ul style="list-style-type: none">• 불법 복제로 인한 수익 손실 또는 명예 훼손	

취약점 확인

기술적으로 모든 모바일 코드는 코드 변조에 취약합니다. 모바일 코드는 코드를 생성하는 조직의 통제하에 있지 않은 환경에서 실행됩니다. 동시에 코드가 실행되는 환경을 변경하는 다양한 방법이 있습니다. 공격자들은 코드를 수정하고 마음대로 수정할 수 있습니다.



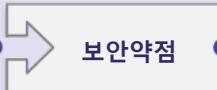

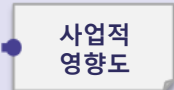
모바일 코드가 본질적으로 취약하지만, 권한없는 코드 수정을 탐지하고 시도하는 것이 가치가 있는지 스스로에게 질문하는 것이 중요합니다. 특정 비즈니스 카테고리(예: 게임)용으로 작성된 앱은 코드 수정의 영향에 대해 다른 앱(예: 엔터테인먼트보다 훨씬 취약합니다. 따라서 이 위험을 처리할지 여부를 결정하기 전에 비즈니스 영향을 고려하는 것이 중요합니다.

공격 시나리오

앱 스토어에서 사용할 수 있는 수많은 가짜 애플리케이션이 있습니다. 일부는 멀웨어 페이로드를 포함합니다. 수정된 앱의 대부분은 수정된 형태의 원래 코어 바이너리 및 관련 리소스를 포함합니다. 침입자는 새로운 어플리케이션으로 다시 패키징하고 제3자 스토어에 배포했습니다.

• 시나리오#1:
게임이 공격하기에 특화된 대상입니다. 공격자는 게임의 모든 프리미엄 기능을 지불하는 데 관심이 없는 사람들을 끌어 들일 것입니다. 코드 내에서 공격자는 어플리케이션 내 구입이 성공했는지 여부를 감지하는 부분을 식제합니다. 이 우회로 희생자는 게임아이템이나 새로운 능력을 지불하지 않고도 얻을 수 있습니다. 또한 공격자는 사용자의 신원을 도용 할 스파이웨어를 삽입했습니다.

• 시나리오#2:
은행 어플리케이션은 또 다른 공격 대상입니다. 이러한 앱은 일반적으로 공격자가 유용할만한 중요 정보를 처리합니다. 공격자는 사용자 개인식별 정보(PII)를 사용자 이름/암호와 함께 제3자 사이트로 전송하는 변조된 앱을 만들 수 있습니다. 이것은 제우스 악성 코드에 해당하는 데스크톱을 연상시킵니다. 이는 일반적으로 사기를 초래합니다.

 위협원	 공격방법	 보안약점		 기술적 영향도	 사업적 영향도
특정 어플리케이션	공격가능성 쉬움	취약점 분포 일반	탐지가능성 평균	영향도 심각	어플리케이션 / 특정 비즈니스
일반적으로 공격자는 타사 앱 스토어에서 호스팅되는 악성 앱을 통해 코드 수정을 악용합니다. 공격자는 피싱 공격을 통해 사용자를 속여 앱을 설치할 수도 있습니다.	일반적으로 공격자는 이 범주를 악용하기 위해 다음 작업을 수행합니다. <ul style="list-style-type: none">• 어플리케이션 프로그램 램 패키지의 코어 바이너리로 직접 바이너리 변경• 어플리케이션 패키지 안의 리소스를 직접 바이너리값으로 변경하십시오.• 시스템 API를 리다이렉션하거나 대체하여 악의적인 외부코드를 가로채고 실행합니다.	어플리케이션의 수정된 품은 생각보다 훨씬 더 일반적입니다. 앱 스토어 내에서 승인되지 않은 버전의 모바일 앱을 감지하고 제거하는 데 기반을 둔 보안 업체가 있습니다. 코드 수정 탐지 문제를 해결하기 위한 접근법에 따라 조치는 승인되지 않은 코드 탐지 방법을 매우 성공적으로 제한할 수 있다. 이 항목에는 바이너리 패치, 로컬 리소스 수정, 메서드 후킹, 메서드 변경 및 동적 메모리 수정이 포함됩니다. 어플리케이션이 모바일 장치로 전달되면 코드 및 데이터 리소스가 해당 모바일 장치에 상주합니다. 공격자는 코드를 직접 수정하거나, 메모리 내용을 동적으로 변경하거나, 어플리케이션이 사용하는 시스템 API를 변경하거나 대체하거나, 어플리케이션의 정보와 리소스를 수정할 수 있습니다. 이는 공격자에게 소프트웨어의 의도적으로 개인적 또는 금전적 이득을 위해 파괴하는 직접적인 방법을 제공할 수 있습니다.	코드 수정 영향은 수정 자체의 성격에 따라 다양할 수 있습니다. 일반적인 유형의 영향은 다음과 같습니다. <ul style="list-style-type: none">• 미승인된 새 기능• 신분 도용 또는 사기	코드 수정으로 인한 비즈니스 영향은 일반적으로 다음과 같습니다. <ul style="list-style-type: none">• 불법 복제로 인한 수익 손실 또는 명예 훼손	

보안대책

모바일 앱은 컴파일 타임에 코드 무결성에 대해 알고 있거나 변경된 코드를 런타임에 감지할 수 있어야 합니다. 앱이 런타임에 코드 무결성 위반에 적절하게 대응할 수 있어야 합니다.

이러한 유형의 위험에 대한 수정 전략은 OWASP 리버스 엔지니어링 및 코드 수정 방지 프로젝트에서 보다 자세하게 기술되어 있습니다.

안드로이드 루팅 보안대책

일반적으로 수정된 앱은 탈옥 또는 루팅된 환경에서 실행됩니다. 따라서 이러한 유형의 손상된 환경을 런타임에 시도하고 적절히 대응하는 것이 합리적입니다(서버에 보고 또는 종료). 루팅된 Android 기기를 감지하는 몇 가지 일반적인 방법이 있습니다. 테스트 키 확인

- build.prop 라인에 개발자 빌드를 가리키는 ro.build.tags=test-keys를 가리키는 개발자 빌드 또는 비공식 ROM이 있는지 확인하십시오.
- OTA 인증서 확인
/etc/security/otacerts.zip 파일이 있는지 확인하십시오.
- 몇 가지 알려진 루팅된 apk를 확인하십시오.
com.noshufou.android.su, com.thirdparty.superuser
eu.chainfire.supersu, com.koushikdutta.superuser
- SU 바이너리 확인
/system/bin/su, /system/xbin/su, /sbin/su
/system/su, /system/bin/ext/su
- SU 명령어를 직접 시도하십시오.
su 명령어를 실행하여 사용자ID를 확인한 후, 0을 반환한다면, su 명령어는 성공적으로 실행



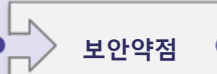
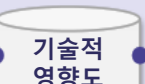
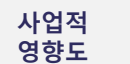
참고자료

OWASP

- OWASP Reverse Engineering and Code Modification Prevention Project

외부자료

- [1] Anxan Research: State of Security in the App Economy, Volume
- [2] HP Research: HP Research Reveals Nine out of 10 Mobile Applications Vulnerable to Attack
- [3] North Carolina State University: Dissecting Android Malware: Characterization and Evolution
- [4] Tech Hive: Apple Pulls Ripoff Apps from its Walled Garden
- [5] Tech Crunch: Developer Spams Google Play With RipOffs of Well-Known Apps... Again
- [6] Extreme Tech: Chinese App Store Offers Pirated iOS Apps Without the Need To Jailbreak
- [7] OWASP: Architectural Principles That Prevent Code Modification or Reverse Engineering
- [8] Gartner report: Avoiding Mobile App Development Security Pitfalls
- [9] Gartner report: Emerging Technology Analysis: Mobile Application Shielding
- [10] Gartner report: Proliferating Mobile Transaction Attack Vectors and What to Do About Them
- [11] Gartner report: Select a Secure Mobile Wallet for Proximity
- [12] Forrester paper: Choose The Right Mobile Development Solutions For Your Organization
- [13] John Wiley and Sons, Inc.: iOS Hacker's Handbook, Published
- [14] McGraw Hill Education: Mobile Hacking Exposed, Published
- [15] Publisher Unannounced: Android Hacker's Handbook, To Be Published
- [16] Software Development Times: More than 5,000 apps in the Google Play Store are copied APKs, or 'thief-ware'
- [17] InfoSecurity Magazine: Two Thirds of Personal Banking Apps Found Full of Vulnerabilities,
- [18] InfoSecurity Magazine: Mobile Malware Infects Millions: LTE Spurs Growth,

 위험원	 공격방법	 보안약점		 기술적 영향도	 사업적 영향도
특정 어플리케이션	공격가능성 쉬움	취약점 분포 일반	탐지가능성 쉬움	영향도 적당	어플리케이션 / 특정 비즈니스
공격자는 일반적으로 앱 스토어에서 대상 앱을 다운로드하고 다양한 도구들을 사용하여 자신의 로컬 환경에서 분석합니다.	공격자는 최종 코어 바이너리를 분석하여 앱에 포함된 원문자열 테이블, 소스코드, 라이브러리, 알고리즘 및 리소스를 확인합니다. 공격자의 환경에서 IDA Pro, Hopper, otool, 문자열 및 기타 바이너리 검사 도구와 같이 비교적 저렴하고 잘 이해된 도구를 사용합니다.	일반적으로 모든 모바일 코드는 리버스 엔지니어링의 영향을 받습니다. 일부 앱은 다른 앱보다 감염되기 쉽습니다. 런타임에 동적 인트로 스코프 (Java, .NET, Objective C, Swift)를 허용하는 언어/프레임워크로 작성된 코드는 특히 리버스 엔지니어링의 위험에 처해 있습니다. 리버스 엔지니어링 민감성 탐지는 상당히 간단합니다. 먼저 앱의 앱 스토어 버전을 해독합니다 (바이너리 암호화가 적용되는 경우). 그런 다음, 이 문서의 "공격 벡터"에 설명된 도구를 바이너리 파일과 비교하여 사용하십시오. 앱의 제어 흐름 경로, 문자열표 및 이 도구로 생성된 코드/소스 코드를 이해하기가 쉽습니다.		공격자는 리버스 엔지니어링으로 다음 중 하나를 이룰 수 있습니다. <ul style="list-style-type: none">백엔드 서버정보 공개암호상수와 암호공개지적 재산 훔치기백엔드 시스템 공격 수행 또는코드 수정에 필요한 지식을 얻으십시오.	리버스 엔지니어링의 비즈니스 영향은 매우 다양합니다. 여기에는 다음이 포함됩니다. <ul style="list-style-type: none">지적 재산 도용명예 훼손신분 도용 또는백엔드 시스템 손상

취약점 확인

일반적으로 대부분의 어플리케이션 코드 고유 특성으로 리버스 엔지니어링의 영향을 받습니다. 오늘날 앱을 작성하는 데 사용되는 대부분의 언어는 프로그래머가 앱을 디버깅하는 데 도움이 되는 메타 데이터가 풍부합니다. 이 같은 기능은 공격자가 앱의 작동 방식을 이해하는 데 도움이 됩니다.

공격자가 다음 중 하나를 수행할 수 있는 경우 앱은 리버스 엔지니어링의 영향을 받을 수 있다고 합니다.

- 바이너리 문자열 테이블의 내용을 명확하게 이해합니다.
- 교차 기능 분석을 정확하게 수행합니다.
- 바이너리에서 소스 코드를 합리적으로 정확하게 재현합니다.

대부분의 앱은 리버스 엔지니어링의 영향을 받기 쉽지만 이 리스크를 완화할지 여부를 고려할 때 리버스 엔지니어링의 잠재적인 비즈니스 영향을 조사하는 것이 중요합니다. 리버스 엔지니어링만으로 수행할 수 있는 작업의 간단한 예는 아래를 참조하십시오.

보안대책

대부분의 앱은 리버스 엔지니어링의 영향을 받기 쉽지만 이 리스크를 완화할지 여부를 고려할 때, 리버스 엔지니어링의 잠재적인 비즈니스 영향을 조사하는 것이 중요합니다. 리버스 엔지니어링만으로 수행할 수 있는 작업에 대한 간단한 샘플은 아래 예제를 참조하십시오.

좋은 난독화는 다음과 같은 사항을 포함합니다:

- 난독화 할 메소드/코드 세그먼트를 좁히십시오.
- 성능 영향의 균형을 맞추기 위해 난독화 정도를 조정하십시오.
- IDA Pro 및 Hopper와 같은 도구로 분석이 어려울 수 있습니다. 메소드뿐만 아니라 문자열 테이블을 난독화합니다.

참고자료 OWASP

- OWASP Reverse Engineering and Code Modification Prevention Project

외부자료

- [1] Arvan Research: State of Security in the App Economy, Volume
- [2] HP Research: HP Research Reveals Nine out of 10 Mobile Applications Vulnerable to Attack
- [3] North Carolina State University: Dissecting Android Malware: Characterization and Evolution
- [4] Tech Hive: Apple Pulls Ripoff Apps from its Walled Garden
- [5] Tech Crunch: Developer Spams Google Play With RipOffs of Well-Known Apps... Again
- [6] Extreme Tech: Chinese App Store Offers Pirated iOS Apps Without the Need To Jailbreak
- [7] OWASP: Architectural Principles That Prevent Code Modification or Reverse Engineering
- [8] Gartner report: Avoiding Mobile App Development Security Pitfalls
- [9] Gartner report: Emerging Technology Analysis: Mobile Application Shielding
- [10] Gartner report: Proliferating Mobile Transaction Attack Vectors and What to Do About Them
- [11] Gartner report: Select a Secure Mobile Wallet for Proximity
- [12] Forrester paper: Choose The Right Mobile Development Solutions For Your Organization
- [13] John Wiley and Sons, Inc: iOS Hacker's Handbook, Published
- [14] McGraw Hill Education: Mobile Hacking Exposed, Published
- [15] Publisher Unannounced: Android Hacker's Handbook, To Be Published
- [16] Software Development Times: More than 5,000 apps in the Google Play Store are copied APKs, or 'thief-wares'
- [17] InfoSecurity Magazine: Two Thirds of Personal Banking Apps Found Full of Vulnerabilities
- [18] InfoSecurity Magazine: Mobile Malware Infects Millions; LTE Spurs Growth

공격 시나리오 샘플

시나리오 #1: 문자열 테이블 분석

공격자는 암호화되지 않은 앱에 대해 '문자열'을 실행합니다. 문자열 테이블 분석의 결과로 공격자는 인증 자격 증명을 백엔드 DB에 포함하는 하드코딩된 연결 문자열을 찾습니다. 공격자는 이러한 자격 증명을 사용하여 DB에 접근합니다. 공격자는 앱의 사용자에게 관한 방대한 양의 개인식별정보(PII)를 훔칩니다.


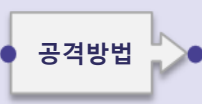
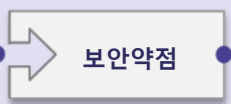

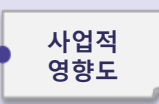
시나리오 #2: 교차 기능 분석

공격자는 암호화되지 않은 앱에 대해 IDA Pro를 사용합니다. 교차 기능 참조와 결합된 문자열 테이블 분석의 결과로 공격자는 본문 탐지 코드를 발견합니다. 그런 다음 Dex2Jar 사용하여 공격자가 jar 파일을 쉽게 변환할 수 있습니다. 다음 단계에서는 JDGui와 같은 자바 디컴파일러가 코드를 제공합니다.

시나리오 #3: 소스 코드 분석

뱅킹 안드로이드 어플리케이션을 고려해보십시오. APK 파일은 7zip/Winrar/WinZip/Gunzip을 사용하여 쉽게 추출할 수 있습니다. 일단 추출되면, 공격자는 파일, 자산, 자원 및 가장 중요한 class.dex 파일을 명시합니다. 그런 다음 Dex2Jar 사용하여 공격자가 jar 파일을 쉽게 변환할 수 있습니다. 다음 단계에서는 JDGui와 같은 자바 디컴파일러가 코드를 제공합니다.

M10 불필요한 기능

 위험원	 공격방법	 보안약점		 기술적 영향도	 사업적 영향도
특정 어플리케이션	공격가능성 쉬움	취약점 분포 일반	탐지가능성 평균	영향도 심각	어플리케이션 / 특정 비즈니스
일반적으로 공격자는 백엔드 시스템에서 숨겨진 기능을 발견하기 위해 모바일 응용 프로그램 내의 관계없는 기능을 이해하려고 합니다. 침입자는 일반적으로 최종 사용자의 개입없이 자체 시스템에서 직접 관련없는 기능을 악용합니다.	검색자의 로컬 환경에서 모바일 어플리케이션을 다운로드하고 검사하십시오. 셀파일과 구성 파일 및 바이너리 자체 검사를 실행하면 드라이버가 실행되지 않습니다. 공격자는 당신의 컴퓨터에 접근할 수 있습니다.	주어진 모바일 앱에 인터페이스로 사용자에게 직접 노출되지 않는 불필요한 기능이 포함될 가능성이 높습니다. 이 추가 코드의 대부분은 본질적으로 양성이며 공격자에게 백엔드 기능에 대한 추가 통찰력을 주지 않습니다. 그러나 일부 외부 기능은 공격자에게 매우 유용할 수 있습니다. 백엔드 테스트, 데모, 스테이징 또는 UAT 환경과 관련된 정보를 제공하는 기능을 프로덕션 빌드에 포함되면 안됩니다. 또한 관리 API 엔드 포인트 또는 비공식 엔드포인트는 최종 프로덕션 빌드에 포함되지 않아야 합니다. 관계없는 기능을 탐지하는 것은 까다로울 수 있습니다. 자동화된 정적 및 동적 분석 도구는 낮은 매달린 과일(로그 문)을 선택할 수 있습니다. 그러나 일부 백도어는 자동화된 방법으로 탐지하기 어렵습니다. 따라서 수동 코드 리뷰로 이 문제를 방지하는 것이 가장 좋습니다.		불필요한 기능의 기술적 영향은 다음과 같습니다. <ul style="list-style-type: none">백엔드 시스템 작동 방식에 대한 노출허가되지 않은 최고 권한 실행	불필요한 기능의 비즈니스 영향은 다음과 같습니다. <ul style="list-style-type: none">민감 기능에 대한 무단 접근명예 훼손지적 재산 도난

취약점 확인

종종 개발자는 숨겨진 백도어 기능 또는 프로덕션 환경으로 배포되지 않을 예정인 기타 내부 개발 보안 컨트롤을 포함합니다. 예를 들어 개발자가 실수로 하이브리드 앱에 댓글로 비밀번호를 포함할 수 있습니다. 또 다른 예는 테스트 중 2팩터 인증을 사용하지 못하도록 하는 것입니다.

이 위험 요소를 정의하는 것은 앱에 공개되지 않을 기능을 앱에 그대로 두는 것입니다.

보안대책

이 취약점을 방지하는 가장 좋은 방법은 이 코드에서 가장 잘 알고있는 보안 전문가 또는 서브젝트 전문가를 사용하여 수동으로 보안 코드를 검토하는 것입니다. 이들은 다음을 수행해야 합니다.

1. 숨겨진 스위치를 찾기 위해 앱의 구성 설정을 검사하십시오.
2. 모든 테스트 코드가 앱의 최종 제작 빌드에 포함되어 있지 않은지 확인합니다.
3. 모바일 앱이 접근하는 모든 API 엔드포인트를 검사하여 이러한 엔드포인트가 잘 문서화되고 공개적으로 사용 가능한지 확인하십시오.
4. 모든 로그 문을 검사하여 백엔드를 과도한 설명이 로그에 기록되지 않도록 하십시오.

공격 시나리오 샘플

시나리오 #1: 관리자 엔드포인트 노출

모바일 엔드포인트 테스트 일환으로 개발자는 모바일 대시보드에 관리되는 대시보드를 표시하는 숨겨진 인터페이스를 포함시켰습니다. 이 대시 보드는 백엔드 API 서버를 통해 관리자 정보에 액세스했습니다. 프로덕션 버전 코드에서 개발자는 대시보드를 표시하는 코드를 항상 포함하지 않았습니다. 그러나 백엔드 관리 API에 액세스 할 수 있는 기본 코드가 포함되었습니다. 침입자는 바이너리에 대한 문자열 테이블 분석을 수행하고 하드 코딩된 URL을 관리 REST 엔드포인트로 발견했습니다. 그 후 공격자는 백엔드 관리 기능을 실행하기 위해 'Curl'을 사용했습니다.

개발자는 네이티브 인터페이스로 직접 도달 할 수 없는 코드를 포함하여 불필요한 코드를 모두 제거해야 합니다.

시나리오 #2: 설정 파일의 디버그 플래그:

공격자는 로컬 앱의 .properties파일에 수동으로 "debug = true"를 추가하려고 합니다. 시작 시 어플리케이션 과도하게 설명적이고 공격자가 백엔드 시스템을 이해하는 데 도움이 되는 로그 파일을 출력합니다. 그런 다음 공격자는 로그 결과로 백엔드 시스템 내의 취약성을 발견합니다.

개발자는 모바일 앱의 프로덕션 빌드에서 '디버그 모드' 활성화를 방지해야 합니다.

참고자료

OWASP

- 홈페이지 참조

외부자료

- 홈페이지 참조

THE BELOW ICONS REPRESENT WHAT OTHER VERSIONS ARE AVAILABLE IN PRINT FOR THIS TITLE BOOK.

ALPHA: "Alpha Quality" book content is a working draft. Content is very rough and in development until the next level of publication.

BETA: "Beta Quality" book content is the next highest level. Content is still in development until the next publishing.

RELEASE: "Release Quality" book content is the highest level of quality in a books title's lifecycle, and is a final product.



ALPHA
PUBLISHED



BETA
PUBLISHED



RELEASE
PUBLISHED

YOU ARE FREE:



to share - to copy, distribute and transmit the work



to Remix - to adapt the work

UNDER THE FOLLOWING CONDITIONS:



Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).



Share Alike. - If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.



OWASP

The Open Web Application Security Project

The Open Web Application Security Project (OWASP) is a worldwide free and open community focused on improving the security of application software. Our mission is to make application security "visible," so that people and organizations can make informed decisions about application security risks. Everyone is free to participate in OWASP and all of our materials are available under a free and open software license. The OWASP Foundation is a 501c3 not-for-profit charitable organization that ensures the ongoing availability and support for our work.

OWASP(Open Web Application Security Project)는 안전한 웹 및 어플리케이션을 개발할 수 있도록 지원하기 위해 미국에서 2004년 4월부터 시작된 비영리 단체이며, 전 세계 기업, 교육기관 및 개인이 만들어가는 오픈 소스 어플리케이션 보안 프로젝트를 진행하고 있습니다. OWASP는 중립적, 실무적이면서도 비용효과적인 어플리케이션 보안 가이드라인을 무료로 제공하고 있습니다.

현재 보고 계신 OWASP Mobile Top 10 2016 한국어판은 Black Falcon팀에서 번역하였습니다.

OWASP Mobile Top 10 2016(원문)은 Owasp.org에 게시되어 있고, 한국어 버전은 문서로 제작하였습니다.

— 번역 : 이지혜, 감수 : 장경침

All for one, one for all BI@ckFALCON