



Adrian Horzyk

BEZPIECZEŃSTWO URZĄDZEŃ MOBILNYCH



*Czyli jak zabezpieczyć dane, ich bazy i transfery
w sieci oraz nie dopuścić do wycieku danych?*



ZAGROŻENIA, RYZYKA I BEZPIECZEŃSTWO



Możemy wyróżnić kilka podstawowych grup zagrożeń, które mogą wpływać na kwestie bezpieczeństwa różnych systemów informatycznych:

1. Bezpieczeństwo systemów i aplikacji
2. Bezpieczeństwo przechowywanych i transmitowanych danych
3. Czynniki psychologiczne wpływające na możliwość ujawnienia danych, szyfrów w celu uzyskania dostępu do danych, aplikacji czy systemów (inżynieria społeczna i atak na użytkownika).

W celu zabezpieczenia się stosujemy kryptografię, zaś w przypadku psychologicznego czynnika ludzkiego, niezbędne są szkolenia w celu uniknięcia manipulacji.

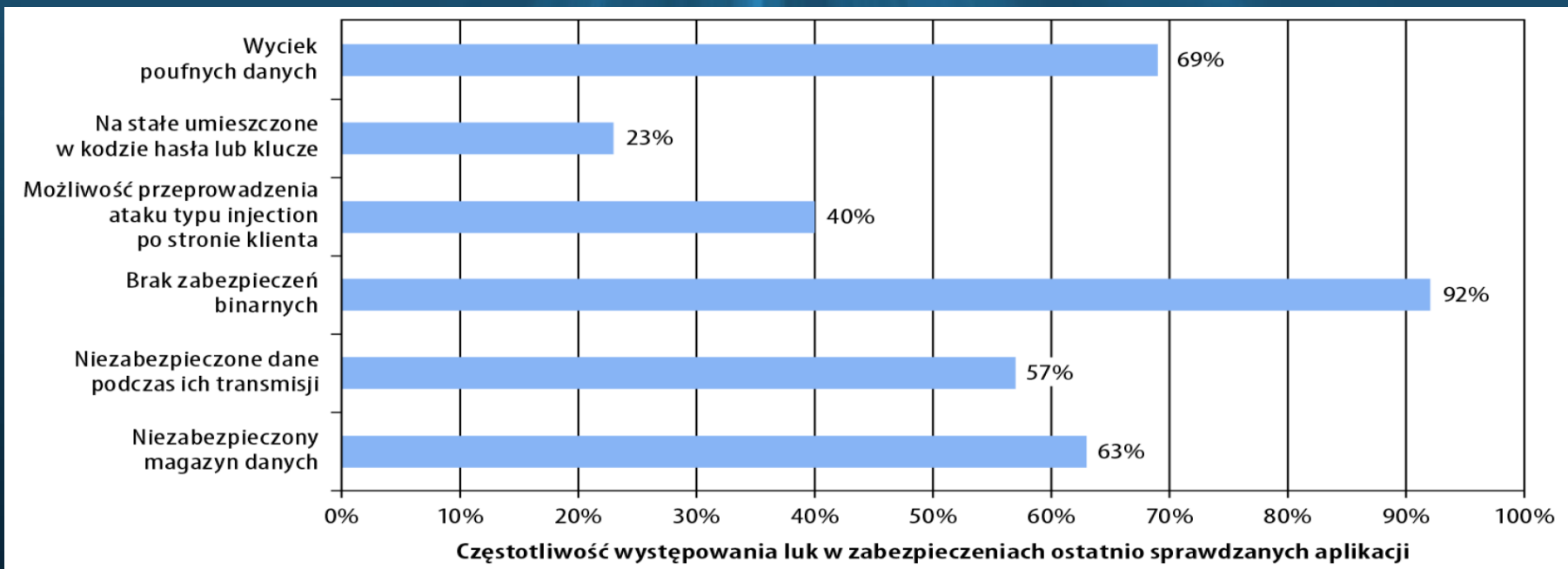


BEZPIECZEŃSTWO



Bezpieczeństwo:

- ✓ Przechowywania, transmisji i dostępu do danych
- ✓ Aplikacji mobilnych w sieciach telekomunikacyjnych
- ✓ Aplikacji mobilnych przez WiFi, Hotspoty, ...





TYPOWE BRAKI W ZABEZPIECZENIACH



Niezabezpieczony
magazyn danych

63%

Niezabezpieczony magazyn danych (63%):

- ✓ różne defekty, które prowadzą do tego, że aplikacja przechowuje w urządzeniu mobilnym dane w postaci zwykłego tekstu. Z reguły jest stosowany zaciemniony format wykorzystujący na stałe umieszczony w kodzie klucz lub też inne rozwiązanie pozwalające osobie atakującej w dość łatwy sposób odczytać informacje przechowywane w magazynie danych.

Niezabezpieczone dane
podczas ich transmisji

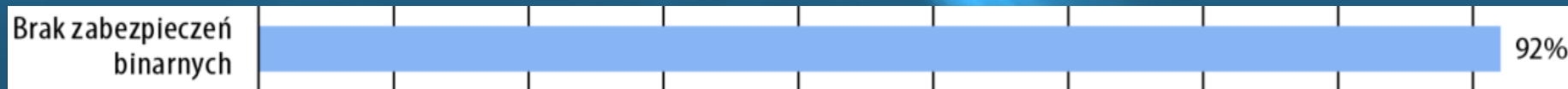
57%

Niezabezpieczone dane podczas ich transmisji (57%):

- ✓ gdy aplikacja nie używa szyfrowania warstwy transportu w celu ochrony
- ✓ danych podczas ich transmisji,
- ✓ Wykorzystano szyfrowanie warstwy transportu, lecz sam mechanizm szyfrowania został zaimplementowany w niewystarczająco bezpieczny sposób.

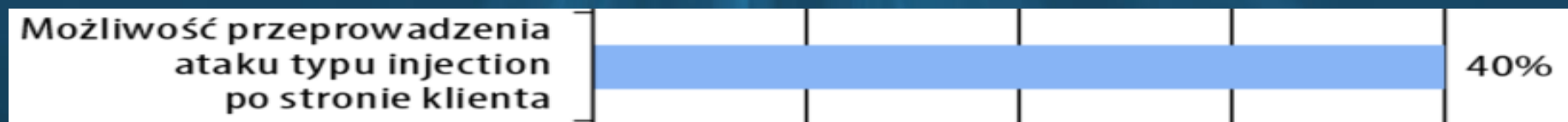


TYPOWE BRAKI W ZABEZPIECZENIACH



Brak zabezpieczeń binarnych (92%):

- ✓ aplikacja nie stosuje żadnej formy mechanizmu obronnego utrudniającego przeprowadzenie procesu inżynierii odwrotnej, złośliwe manipulowanie programem lub jego debugowanie.

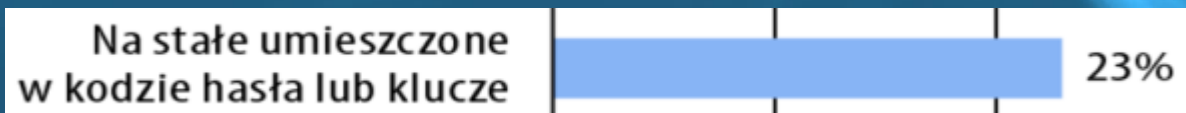


Możliwość ataku typu injection po stronie klienta (40%):

- ✓ scenariusze, w których niezaufane dane wejściowe są przekazywane aplikacji, gdzie następnie są przetwarzane w niebezpieczny sposób.
- ✓ typowe źródła ataków typu injection obejmujące na przykład inne aplikacje zainstalowane w urządzeniu oraz pochodzące z serwera dane wejściowe przekazywane do aplikacji.

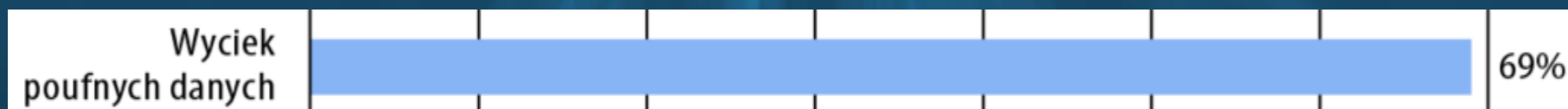


TYPOWE BRAKI W ZABEZPIECZENIACH



Na stałe umieszczone w kodzie hasła lub klucze (23%):

- ✓ programista na stałe umieszcza w kodzie aplikacji pewne informacje wrażliwe, tj. hasło lub klucz szyfrowania.



Wyciek poufnych danych (69%):

- ✓ aplikacja przypadkowo ujawnia poufne dane w wyniku ataku typu side-channel, polegającego na utworzeniu dodatkowego kanału informacyjnego, czyli wyciek danych na skutek użycia pewnego frameworka lub systemu operacyjnego.



OWASP



OWASP:

- ✓ Organizacja **Open Web Application Security Project** mająca na celu stworzenie standardów zabezpieczania aplikacji internetowych.
- ✓ Zajmuje się **kategoryzowaniem ryzyka i luk** w zabezpieczeniach.
- ✓ **Sprawdzanie bezpieczeństwa** aplikacji mobilnych.
- ✓ Koncentruje się na warstwie aplikacji, choć bierze pod uwagę również ryzyko związane ze stosowaniem poszczególnych platform mobilnych.

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



Najważniejsze zagrożenia aplikacji mobilnych



10 najważniejszych zagrożeń aplikacji mobilnych wg OWASP:

1. Słaba kontrola po stronie serwera (M1)
2. Niezabezpieczony magazyn danych (M2)
3. Niewystarczająca ochrona warstwy transportowej (M3)
4. Niezamierzony wyciek danych (M4)
5. Kiepski mechanizm autoryzacji i uwierzytelniania (M5)
6. Nieprawidłowe stosowanie kryptografii (M6)
7. Możliwość przeprowadzenia ataku typu injection po stronie klienta (M7)
8. Podejmowanie ważnych decyzji na podstawie niezaufanych danych wejściowych (M8)
9. Nieprawidłowa obsługa sesji (M9)
10. Brak zabezpieczeń binarnych (M10)



Zagrozenie M1



Słaba kontrola po stronie serwera:

- ✓ Najpoważniejsze zagrożenie dla aplikacji mobilnych
- ✓ Wszelkie luki w zabezpieczeniach serwera, tzn. w usługach sieciowych, konfiguracjach serwerów WWW oraz aplikacjach internetowych.
- ✓ Dokładne omówienie można znaleźć w książce [The Web Application Hacker's Handbook](#)



Zagrożenie M2



Niezabezpieczony magazyn danych:

- ✓ Gdy aplikacja przechowuje poufne dane (np. dane osobowe, numery kart kredytowych, adresy e-mail, numery telefonów) w formacie zwykłego tekstu lub w innej postaci (np. XML) łatwej do odczytania przez osobę atakującą.
- ✓ Uzyskanie dostępu do systemu plików na skutek użycia oprogramowania typu malware lub złamania zabezpieczeń w inny sposób.
- ✓ Wpływ tej kategorii może mieć poważne konsekwencje dla biznesu!
- ✓ Może prowadzić do:
 - ✓ kradzieży tożsamości,
 - ✓ oszustw i kradzieży,
 - ✓ utraty reputacji.



Zagrozenie M3



Niewystarczająca ochrona warstwy transportowej:

- ✓ Dotyczy zabezpieczenia ruchu sieciowego.
- ✓ Pojawia się w każdej sytuacji, gdy dane są przekazywane w postaci zwykłego tekstu lub w innej słabo zabezpieczonej postaci.
- ✓ Wykorzystanie słabych szyfrów lub łatwych do złamania algorytmów szyfrujących.
- ✓ Dopuszczenia samodzielnego podpisywania certyfikatów.
- ✓ Niewystarczającej weryfikacji certyfikatów.

Atak może nastąpić z sieci lokalnej lub z sieci operatora.



Zagrożenie M4



Niezamierzony wyciek danych:

- ✓ Na skutek niewystarczającej wiedzy programistów odnośnie przechowywania danych przez dany system operacyjny (platformę).
- ✓ Przez umieszczanie danych w miejscach, gdzie mogą być łatwo dostępne z poziomu innych aplikacji, które mogą się do nich włamać.
- ✓ Niewłaściwy sposób buforowania danych.
- ✓ Niewłaściwego stosowania migawek (zrzutów ekranów).
- ✓ Niewłaściwego stosowania dzienników zdarzeń.



Zagrozenie M5



Kiepski mechanizm autoryzacji i uwierzytelniania:

- ✓ Błędy podczas uwierzytelniania i autoryzacji.
- ✓ Szczególnie przy dostępie do poufnych danych w trybie off-line.
- ✓ Możliwość obejścia mechanizmu uwierzytelniania i uzyskania dostępu do aplikacji lub danych.
- ✓ Błędy w autoryzacji po stronie serwera mogą spowodować możliwość wykonania pewnych funkcji ponad te przyznane danemu użytkownikowi.



Zagrożenie M6



Nieprawidłowe stosowanie kryptografii:

- ✓ W sytuacji, gdy zakłada się, iż dane są szyfrowane w aplikacji, lecz sposób szyfrowania zawiera pewne słabe strony.
- ✓ W wyniku tego może być możliwe przechwycenie fragmentów lub całości niezaszyfrowanych danych, ew. złamanie szyfru lub jego ujawnienie.
- ✓ Powodem może być kiepskie zarządzanie kluczem szyfrowania, np. umieszczenie klucza prywatnego w aplikacji, w kodzie klucza statycznego lub w miejscu, gdzie może być w łatwy sposób znaleziony w urządzeniu, np. użycie jako klucza identyfikatora urządzenia Android.



Zagrożenie M7



Możliwość przeprowadzenia ataku typu injection po stronie klienta:

- ✓ Gdy aplikacja mobilna akceptuje dane z niezauważanych źródeł, np.:
 - ✓ wewnętrznych urządzenia, czyli innej aplikacji,
 - ✓ zewnętrznych, tj. komponenty działające po stronie serwera.



Zagrożenie M8



Podejmowanie ważnych decyzji na podstawie niezaufanych danych wejściowych:

- ✓ Gdy decyzja dotycząca bezpieczeństwa jest podejmowana na podstawie danych wejściowych pochodzących z zaufanego źródła.
 - ✓ Wiąże się z mechanizmami komunikacji międzyprocesowej (IPC).
 - ✓ Może powstać w przypadku, gdy np. wiele współdziałających aplikacji ma korzystać z tego samego tokenu sesji w celu wyeliminowania potrzeby uwierzytelniania użytkownika w każdej z tych aplikacji.
- Jeżeli mechanizm IPC nie zostanie prawidłowo zabezpieczony, wtedy każda inna aplikacja może pobrać ten token i uzyskać dostęp do sesji użytkownika.



Zagrożenie M9



Nieprawidłowa obsługa sesji:

- ✓ To wszelkie luki w zabezpieczeniach powodujące ujawnienie tokenu sesji osobie atakującej.
- ✓ Sesja to ważny mechanizm służący do przechowywania informacji o stanie podczas pracy z protokołami bezstanowymi, tj. HTTP czy SOAP.



Zagrożenie M10



Brak zabezpieczeń binarnych:

- ✓ Powoduje osłabienie mechanizmów obronnych, które programista powinien wbudować w aplikację.
- ✓ Zabezpieczenia binarne mają na celu spowolnienie działania algorytmów lub osób atakujących, które próbują analizować lub modyfikować kod binarny aplikacji lub przeprowadzić proces inżynierii odwrotnej.



Narzędzia rekomendowane przez OWASP Mobile Security



Narzędzia Open Source wzmocniające aplikacje mobilne:

- ✓ iMAS – to framework bezpiecznej aplikacji dla systemu iOS.
- ✓ GoatDroid – to samodzielne środowisko testowe dla aplikacji Android.
- ✓ iGoat – to aplikacja treningowa pozwalająca na udoskonalenie własnych umiejętności dla platformy iOS.
- ✓ Damn Vulnerable iOS – to projekt pozwalający na wykrycie zabezpieczeń dla systemu iOS, tj. wykrycie poddania urządzenia jailbreakingowi, manipulacje środowiskiem uruchomieniowym aplikacji, modyfikacje plików binarnych aplikacji, a także zagadnienia związane z kryptografią.
- ✓ MobiSec – to środowisko przeznaczone do przeprowadzania testów penetracyjnych aplikacji mobilnych.
- ✓ Androidck – to projekt skoncentrowany na zautomatyzowanych zadaniach analizy śledczej aplikacji na platformie Android, a nie testach penetracyjnych czy doskonaleniu własnych umiejętności. Próbuje zdobyć kluczowe dane aplikacji, tj. pliki APK (android package), dane aplikacji, bazy danych i dzienniki zdarzeń.



Bezpieczeństwo uruchamiania aplikacji na platformie iOS



Podstawowe funkcje bezpieczeństwa oferowane przez platformę iOS:

- ✓ Łańcuch procedur bezpiecznego rozruchu (ang. *secure boot chain*).
- ✓ Podpisywanie kodu.
- ✓ Stosowanie piaskownicy (ang. *sandboxing*) na poziomie procesu.
- ✓ Szyfrowanie przechowywanych danych.
- ✓ Złagodzenie potencjalnych luk w języku natywnym:
 1. przygotowanie w sposób losowy układu przestrzeni adresowej;
 2. obszar pamięci, w którym nie można wykonywać kodu;
 3. ochrona przed zniszczeniem stosu.



Łańcuch procedu bezpiecznego rozruchu dla iOS

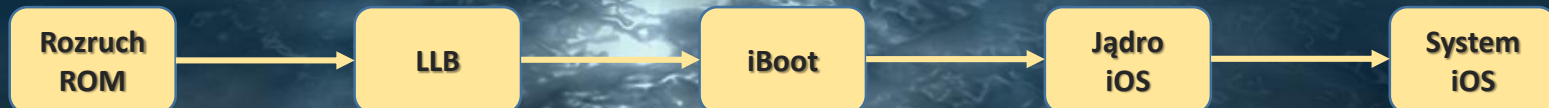


Linie obrony platformy iOS:

- ✓ Oprogramowanie typu firmware jest inicjalizowane i wczytywane w urządzeniu iOS podczas jego rozruchu.
- ✓ Cyfrowo podpisane komponenty przez Apple zostają zweryfikowane w celu zagwarantowania, że nie były zmodyfikowane.

Kolejne kroki łańcucha bezpiecznego rozruchu systemu iOS:

1. Po włączeniu urządzenia z systemem iOS procesor wykonuje najpierw niemodyfikowalny kod rozruchowy zapisany w pamięci ROM procesora.
2. Pamięć ROM zawiera klucz publiczny Apple używany do weryfikacji spójności programu rozruchowego LLB (*low-level bootloader*), który wykonuje procedury konfiguracyjne zanim rozpocznie uruchamianie urządzenia z obrazu iBoot zapisanego w pamięci flash urządzenia. Jeśli podpis cyfrowy nie odpowiada oczekiwanej wartości, urządzenie zostanie uruchomione w trybie odzyskiwania iBoot, który jest odpowiedzialny za sprawdzenie i wczytanie jądra iOS, które z kolei wczytuje system iOS i środowisko trybu użytkownika.





Podpisywanie kodu



Podpisywanie kodu:

- ✓ Uniemożliwia uruchamianie w urządzenia nieautoryzowanych aplikacji.
- ✓ Polega na weryfikacji podpisu cyfrowego aplikacji w trakcie każdego uruchamiania aplikacji.
- ✓ Aplikacja może zostać uruchomiona tylko wtedy, gdy kod jest podpisany za pomocą ważnego i zaufanego podpisu.
- ✓ Wszelkie próby wykonania stron pamięci pochodzących z niepodpisanych źródeł zostaną odrzucone przez jądro iOS.
- ✓ Aby aplikacja mogła zostać uruchomiona w urządzeniu iOS, najpierw musi być podpisana za pomocą zaufanego certyfikatu, który programiści mogą zainstalować wykorzystując Apple profil provisioning, zawierający certyfikat programisty oraz informacje o nadanych uprawnieniach przez programistę.
- ✓ Cały kod aplikacji musi być podpisany przez Apple zanim trafi do App Store.
- ✓ Dzięki temu Apple zachowuje kontrolę nad aplikacjami uruchamianymi na platformie iOS.



Izolacja aplikacji za pomocą piaskownicy



Piaskownica (sandbox):

- ✓ Izoluje aplikacje od innych programów oraz od systemu operacyjnego.
- ✓ Podnosi poziom bezpieczeństwa platformy.
- ✓ Ogranicza ew. szkody, jakie może wyrządzić oprogramowanie typu malware.
- ✓ Każda aplikacja uruchamiana jest we własnym unikatowym katalogu użytkownika urządzenia mobilnego. Izolacja obsługiwana jest przez rozszerzenie jądra o nazwie XNU Sandbox.



Szyfrowanie danych przez system iOS



Szyfrowanie domyślne danych przez system iOS:

- ✓ Oparte jest o algorytm AES
- ✓ Klucz szyfrowania systemu plików generowany jest podczas pierwszego uruchomienia urządzenia i przechowywany jest w bloku nr 1 pamięci masowej NAND flash.
- ✓ Klucz ten służy do odszyfrowania tabeli partycji i partycji systemowej.
- ✓ Po włączeniu urządzenia sprzętowy akcelerator kryptograficzny deszyfruje system plików.
- ✓ Ponadto pliki mogą być szyfrowane za pomocą API Data Protection, wykorzystujący klucz utworzony na podstawie kodu używanego do odblokowania urządzenia.
- ✓ Jeśli jakieś procesy mają działać w tle, tj. w trakcie okresu, gdy urządzenie jest zablokowane, wtedy nie mogą być zaszyfrowane przez API Data Protection.



Ograniczenie wykorzystanie luk w zabezpieczeniach



Metody ograniczające wykorzystanie luk w zabezpieczeniach:

- ✓ Uniemożliwiają jednoczesne oznaczenie strony pamięci do zapisu i wykonania.
- ✓ Strony, na których wykonano zapis, nie mogą być przywrócone do stanu umożliwiającego wykonanie, co chroni system przed uruchamianiem kodu zapisanego w danych (DEP – *data execution protection*).
- ✓ Zabezpieczenie DEP można próbować obejść techniką ROP (return-oriented programming) z zastosowaniem tzw. „ładunków”, ale takie rozwiązanie jest trudne, jeśli zastosowano technikę ASLR i obowiązkowe podpisywanie kodu.
- ✓ ASLR (address space layout randomization) polega na przygotowaniu przestrzeni adresowej w sposób losowy, co uniemożliwia pozostawienie kodu w pamięci zwolnionej przez dane, a następnie uruchomienie ukrytego kodu, gdyż miejsca mapowania danych i kodu przydzielane są losowo, a więc atakujący nie zna położenia fragmentów kodu.
- ✓ Próby obejścia tego mechanizmu wykorzystują mechanizm ujawniania informacji o błędach, w celu wskazania miejsca pamięci do przeprowadzenia ataku.



Zabezpieczenie stosu



Zabezpieczenie stosu systemowego iOS:

- ✓ Chroni stos przed przepełnieniem za pomocą tzw. **kanarków stosu (stack canaries)**, czyli pseudolosowych wartości DWORD wstawianych za zmiennymi lokalnymi.
- ✓ Kanarki sprawdzane są po zakończeniu działania funkcji.
- ✓ Jeśli nastąpi przepełnienie stosu, a kanarek został uszkodzony lub nadpisany, wówczas dochodzi do natychmiastowego zamknięcia aplikacji, aby uniemożliwić jakiegokolwiek niezamierzone działanie mogące prowadzić do uszkodzenia danych w pamięci.



Aplikacje iOS



Wszystkie aplikacje dla iOS należą do jednej z trzech grup:

- ✓ Natywne napisane w języku Objective-C lub Swift
- ✓ Oparte na przeglądarce WWW – to kłona aplikacji internetowej
- ✓ Hybrydowe (korzysta z obu podejść)

Aplikacje hybrydowe są największym zagrożeniem, gdyż kod HTML czy JavaScript nie wymagają podpisywania kodu, zaś natywna część kodu umożliwia dostęp do zasobów urządzenia, np. do aparatu fotograficznego.

Do tworzenia aplikacji dla platformy iOS służy Apple iOS SDK.

Zanim programista uzyska możliwość umieszczenia aplikacji w sklepie App Store, najpierw musi utworzyć bezpłatne konto w serwisie Apple Developer i wykupić udział w programie Apple Developer Program. To umożliwia na otrzymanie certyfikatu, który można wykorzystać do podpisywania aplikacji i jej uruchamiania na maksymalnie 100 różnych urządzeniach iOS z użyciem tymczasowego profilu *provisioning* w celu przetestowania aplikacji na rzeczywistych urządzeniach.

Apple przeprowadza weryfikację aplikacji sprawdzając zgodność z regułami sklepu i wyszukując ew. defekty przed udostępnieniem aplikacji szerszemu gronu odbiorców podpisaną swoim certyfikatem kopię aplikacji.

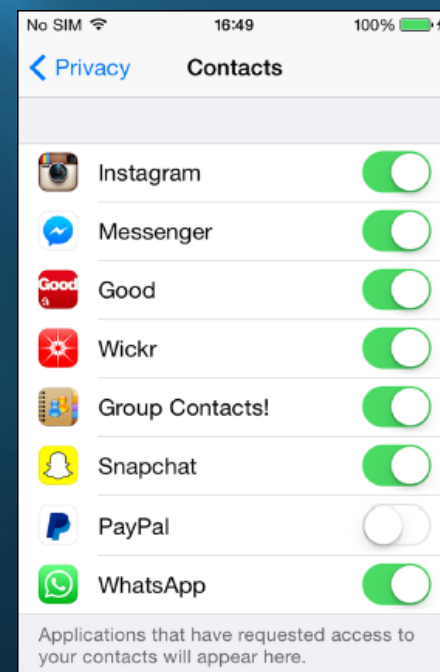
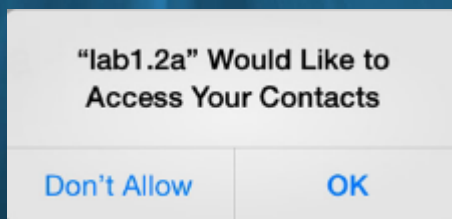


Dostęp do kategorii danych



Aplikacja musi uzyskać zezwolenie użytkownika, jeśli chce uzyskać dostęp do następujących kategorii danych chronionych w iOS:

- ✓ usługi lokalizacji,
- ✓ książka adresowa,
- ✓ kalendarz,
- ✓ zdjęcia,
- ✓ przypomnienia,
- ✓ dostęp do mikrofonu,
- ✓ czujnik ruchu,
- ✓ dostęp do technologii Bluetooth,
- ✓ dane serwisów społecznościowych.



Udzielone zgody mogą zostać cofnięte za pomocą menu ustawień prywatności.



iOS 8 usprawnienia



W iOS 8 wprowadzono wiele usprawnień pozwalające precyzyjniej kontrolować dostęp do niektórych funkcji, np. do informacji o lokalizacji użytkownika.

Do wyboru są trzy opcje:

- ✓ Aplikacja nigdy nie może uzyskać dostępu do danych o lokalizacji użytkownika.
- ✓ Aplikacja może uzyskać dostęp do danych o lokalizacji użytkownika tylko wtedy, gdy jest aktywna i pozostaje w użyciu.
- ✓ Aplikacja zawsze może uzyskać dostęp do danych o lokalizacji użytkownika.



Jailbreaking



Jailbreaking:

- ✓ polega na usunięciu ograniczeń nakładanych przez system iOS, pozwalając użytkownikom na uzyskanie pełnego dostępu do wszystkich zasobów urządzenia;
- ✓ Pozwala na instalację aplikacji pochodzących z nieoficjalnych źródeł, które nie przejdą weryfikacji umożliwiającą umieszczenie ich w sklepie App Store;
- ✓ używają własnego API;
- ✓ pozwalają na obejście ograniczenia związanego z koniecznością podpisywania kodu, a więc umożliwiają uruchamianie aplikacji ze złamanym kodem (piractwo).
- ✓ Umieszczenie w urządzeniu wytrychów, tj. AppSync, pozwala na przeprowadzanie instalacji niepodpisanych pakietów IPA, których spójności i nienaruszalności nie można potwierdzić.



Rodzaje Jailbreakingu



Rodzaje jailbreakingu:

- ✓ **Untethered jailbreak** - najbardziej pożądanym, gdyż pozostaje trwale w urządzeniu nawet po jego ponownym uruchomieniu.
- ✓ **Tethered jailbreak** - nie przetrwa ponownego uruchomienia urządzenia i wymaga podłączonego komputera w celu uruchomienia urządzenia iOS. Jądro nie jest modyfikowane trwale lub w locie, a jeśli urządzenie spróbuje ponownie się uruchomić, pozostanie zablokowane w trybie odzyskiwania.
- ✓ **Semi-tethered jailbreak** – to podejście pośrednie między untethered i tethered jailbreak, ponieważ wymaga obecności komputera w celu uruchomienia urządzenia iOS w stanie jailbreakingu. Uruchomienie lub ponowne uruchomienie urządzenia bez obecności komputera jest możliwe, ale tylko w stanie niepoddanym jailbreakingowi.



Cydia



Cydia (<https://cydia.saurik.com/>):

- ✓ to alternatywa dla sklepu Apple App Store przeznaczona dla urządzeń poddanych jailbreakingowi i instalowana wraz z wieloma aplikacjami typu jailbreak.
- ✓ ma postać aplikacji iOS zapewniającej graficzny interfejs użytkownika dla popularnego narzędzia apt (ang. advanced packaging tool), które jest powszechnie stosowane do zarządzania pakietami w innych systemach z rodziny UNIX i Linux, na przykład w dystrybucji Debian i pochodnych (Ubuntu, Linux Mint itd.).
- ✓ Pozwala na zainstalowanie wielu różnych prekompilowanych pakietów dla urządzenia iOS, między innymi aplikacji, rozszerzeń i narzędzi działających w powłoce. Pakiety oprogramowania są dostarczane w plikach z rozszerzeniem .deb; pliki można pobrać z repozytorium Cydia. Do konfiguracji repozytoriów używana jest opcja Sources w aplikacji Cydia.



Werifikacja Certyfikatu



Działanie protokołów SSL i TLS opiera się na fundamencie koncepcji uwierzytelnienia na podstawie certyfikatu, który ma gwarantować:

- ✓ komunikację z żądanym serwerem,
- ✓ ochronę przed podsłuchiwaniami i modyfikacją danych.

Prawdziwość certyfikatów mają zagwarantować zaufane urzędy certyfikacji (*certificate authority, CA*).

W przypadku nie sprawdzania certyfikatu, może dojść do sytuacji samodzielnego podpisania certyfikatu przez aplikację, więc można prowadzić komunikację z innym podstawionym serwerem.



Zestaw Szyfrowy



Zestaw szyfrowy (*cipher suite*) to połączenie procesu uwierzytelnienia, szyfrowania, uwierzytelniania wiadomości oraz algorytmów wymiany kluczy do negocjowania bezpiecznego połączenia sieciowego za pomocą SSL i TLS.



Deasemblacja



Deasemblacja kodu służy do przeprowadzenia procesu inżynierii odwrotnej aplikacji, w celu zrozumienia sposobu jej działania, w tym zabezpieczeń, które mogą być złamane:

✓ profesjonalna wersja deasemblera Hopper (<http://www.hopperapp.com/>)

```
-[ViewController viewDidLoad]:  
0000b060  push    {r4, r7, lr}                ; Objective C Implementation defined at 0xc098 (instance), XREF=0x40ac  
0000b062  add     r7, sp, #0x4  
0000b064  sub     sp, #0x14  
0000b066  mov     r4, r0  
0000b068  movw    r0, #0x155c  
0000b06c  movt    r0, #0x0  
0000b070  movw    r1, #0x153a  
0000b074  movt    r1, #0x0  
0000b078  add     r0, pc                      ; 0xc5d8  
0000b07a  add     r1, pc                      ; @selector(viewDidLoad)  
0000b07c  str     r4, [sp, #0xc]  
0000b07e  ldr     r0, [r0]  
0000b080  ldr     r1, [r1]  
0000b082  str     r0, [sp, #0x10]  
0000b084  add     r0, sp, #0xc                ; argument #1 for method imp__symbolstub1_objc_msgSendSuper2  
0000b086  blx     imp__symbolstub1_objc_msgSendSuper2  
0000b08a  bl      sub_b1fc  
0000b08e  cbz     r0, 0xb0fc
```



Swizzling Metody



Swizzling metody:

- ✓ Polega na bezpośrednim zastąpieniu implementacji metody w trakcie działania aplikacji.
- ✓ Dokonuje się tego przez wykorzystanie tzw. tabeli *v-table*, która przedstawia mapowanie selektorów na implementację.
- ✓ Selektor używany jest do przedstawienia nazwy metody.
- ✓ Implementacja to wskaźnik prowadzący do początku funkcji.
- ✓ Swizzling odbywa się przez zastąpienie implementacji istniejącego selektora w tabeli *v-table* klasy, dając możliwość wywołania innej implementacji metody.



Wyciek Danych



Aplikacje mogą powodować wyciek danych wrażliwych:

- ✓ w dziennikach zdarzeń i buforach,
- ✓ poprzez schowek systemowy,
- ✓ poprzez schowek operacji wyszukiwania,
- ✓ poprzez schowki niestandardowe.

Buforowanie stosowane jest np. do:

- ✓ buforowania znaków wpisywanych z klawiatury,
- ✓ buforowanie odpowiedzi HTTP,



Tworzenie Bezpiecznych Aplikacji



Osoby atakujące najbardziej zainteresowane są danymi, więc **reguły projektowe** powinny uniemożliwić ich przechwycenie lub ich wyciek, więc trzeba udzielić odpowiedzi na pytania:

- ✓ Jak dane są przechowywane w aplikacji?
- ✓ Jak i kiedy dane powinny być dostępne?
- ✓ W jaki sposób będzie chroniony dostęp do aplikacji?
- ✓ Dlaczego istnieją poszczególne punkty wejścia (tj. dane z urządzeń Bluetooth, procedur obsługi protokołów, kodów QR, iBeacon)?
- ✓ Jaki wpływ na aplikację mają komponenty opracowane przez firmy zewnętrzne?
- ✓ Czy dane są odpowiednio szyfrowane w trakcie ich przechowywania, buforowania oraz transferów.



Zabezpieczanie Danych Poufnych i Wrażliwych



Możliwość szyfrowania poszczególnych plików w systemie plików za pomocą klucza otrzymanego na podstawie wprowadzonego przez użytkownika kodu zabezpieczającego może być niewystarczająca do odpowiedniego zabezpieczenia danych poufnych w celu ich ochrony przed zagrożeniami tj.:

- ✓ Ataki w urządzeniu, na przykład przeprowadzone przez oprogramowanie typu malware lub innego rodzaju oprogramowanie o złośliwym działaniu.
- ✓ Wykorzystanie luk w zabezpieczeniach procedury bezpiecznego rozruchu, które mogłyby pozwolić na zamontowanie systemu plików.
- ✓ Zdefiniowanie przez użytkownika niebezpiecznego lub domyślnego dla danego urządzenia kodu zabezpieczającego.
- ✓ Używanie urządzenia bez zdefiniowanego kodu zabezpieczającego.



Reguły Implementacji Schematu Szyfrowania



W schemacie szyfrowania należy uwzględnić wiele czynników:

- ✓ Użyć doskonale znanego i przetestowanego algorytmu szyfrowania (AES-XTS z kluczem o wielkości 256 jest uznawane za wystarczające w przypadku aplikacji mobilnych, zaś gdy wymagane jest stosowanie wartości hash, wówczas wystarczające jest użycie algorytmu SHA-256 lub silniejszego).
- ✓ Zaimplementować generowanie klucza na podstawie zaakceptowanej funkcji tj. PBKDF2 (*password-based key derivation function*) z dużą liczbą iteracji (np. 10000).
- ✓ Przy podawaniu danych wejściowych wielkością pola nie sugeruj maksymalnej długości klucza, gdyż podanie np. 4-cyfrowego PINu daje zaledwie 10000 możliwych kombinacji, co daje możliwość złamania szyfru atakiem typu *brute force*, czyli sprawdzeniem po kolei wszystkich możliwych kombinacji.
- ✓ Chronić klucz (np. prywatny) szyfrując go przy pomocy klucza głównego lub hasła pochodzącego od użytkownika, co umożliwia zmianę hasła (klucza) bez ponownego zaszyfrowania danych z wykorzystaniem klucza, który się nie zmienia.
- ✓ Używając ciągu zaburzającego (salt) używaj losowo wybranej ilości iteracji (> 10000), co utrudni ataki typu *bruit force* oraz tęcze tablice (*rainbow tables*).



Szyfrowanie Baz Danych



Ochrona danych przechowywanych w bazach danych na urządzeniach mobilnych:

- ✓ Bardzo często stosowaną bazą danych na urządzeniach mobilnych jest SQLite, która jednak nie zapewnia szyfrowania metadanych.
- ✓ Dostępnym na rynku rozwiązaniem jest wykorzystanie SQLCipher (<https://www.zetetic.net/sqlcipher/>), czyli zmodyfikowanej open source implementacji bazy danych SQLite zapewniającą obsługę szyfrowania.
- ✓ Dodatkowo można zastosować własne szyfrowanie danych wrażliwych i poufnych w połączeniu z istniejącymi innymi mechanizmami szyfrowania, n. API Data Protection, pamiętając o nie stosowaniu statycznego i na stałe zdefiniowanego klucza w aplikacji, bo łatwo można go znaleźć a szyfr złamać!



Ochrona w trakcie przesyłania danych



W przypadku konieczności przesyłania danych pomiędzy aplikacją i serwerem lub aplikacjami należy zadbać:

- ✓ O zweryfikowanie przy pomocy certyfikatu serwera odbierającego dane, żeby nie wysłać ich pod fałszywy serwer oraz żeby nie podawać swoich loginów i hasła, gdyż może być wykorzystane do skradzenia tożsamości oraz zalogowania się do prawdziwego serwera przez przestępców.
- ✓ Stosowaniu zasad OWASP.
- ✓ Szyfrowaniu danych.
- ✓ Szyfrowaniu sesji SSL.



Unikanie Ataków SQL Injection



Ataki SQL Injection:

- ✓ Dochodzi do nich w przypadku zmanipulowania danych bezpośrednio w zapytaniu SQL.
- ✓ Można ich unikać przez użycie sparametryzowanych zapytań SQL, w których są zarezerwowane miejsca do zastąpienia ciągami tekstowymi, które mogą być umieszczone w zapytaniu SQL.



Ochrona Aplikacji Poprzez Użycie Zabezpieczeń Binarnych



Zabezpieczenia binarne mają na celu:

- ✓ **Uniemożliwienie działania aplikacji mobilnej w niezaufanym środowisku.**
- ✓ **Zwiększenie poziomu skomplikowania możliwości wykorzystania luk w zabezpieczeniach prowadzących do uszkodzenia pamięci.**
- ✓ **Udaremnienie lub przynajmniej znaczne utrudnienie możliwości przeprowadzenia procesu inżynierii odwrotnej.**
- ✓ **Udaremnienie lub przynajmniej znaczne utrudnienie możliwości przeprowadzenia ataków polegających na modyfikacji aplikacji.**
- ✓ **Wykrywanie ataków przeprowadzanych przez zainstalowane w urządzeniu oprogramowanie typu malware.**
- ✓ **Mogą znacznie utrudnić proces inżynierii odwrotnej, a tym samym zmniejszyć niebezpieczeństwo przeprowadzenia udanego ataku na tak zabezpieczoną aplikację.**



Ochrona Aplikacji Poprzez Użycie Zabezpieczeń Binarnych



Z badań przeprowadzonych w 2013 r. wynikało, iż 86% aplikacji mobilnych nie stosowało odpowiednich zabezpieczeń binarnych, umożliwiając hakerom:

- ✓ Kradzież własności intelektualnej poprzez zastosowanie inżynierii odwrotnej,
- ✓ Ominięcie mechanizmów zabezpieczeń, tj. lokalne uwierzytelnianie, szyfrowanie, licencjonowanie, zarządzanie prawami cyfrowymi, czy też wykrycie przeprowadzenia jailbreakingu urządzenia itd.
- ✓ Utrata zysków ze względu na istnienie pirackich wersji oprogramowania.
- ✓ Szkodzenie wizerunkowi i reputacji marki na skutek istnienia podróbek aplikacji i ataków polegających na modyfikacji kodu aplikacji.



Aplikacje zabezpieczone przeciw jailbreakingowi



Działanie aplikacji zabezpieczonej przeciw jailbreakingowi:

- ✓ Ostrzegają użytkownika i proszą o potwierdzenie akceptacji ryzyka.
- ✓ Uniemożliwiają działanie aplikacji przez jej eleganckie zamknięcie (zakończenie) lub wystąpienie (wywołanie) awarii.
- ✓ Usuwają wszelkie dane poufnych przechowywane w urządzeniu.
- ✓ Przekazują odpowiednią informację do serwera macierzystego, w którym może zostać podjęta pewna akcja, na przykład oznaczenie użytkownika jako podejrzanego.



Osłabienie Piaskownicy



Jednym ze sposobów przeprowadzenia ataku na urządzenia mobilne jest osłabienie piaskownicy, która uniemożliwia im współdziałanie z innymi aplikacjami oraz zawęża dostęp do systemu operacyjnego poprzez:

- ✓ W przypadku urządzenia iOS niepoddanego jailbreakingowi działanie funkcji `fork()` zawsze powinno zakończyć się niepowodzeniem, ponieważ aplikacje opracowane przez firmy trzecie (czyli nie Apple) nie mogą tworzyć nowych procesów. Można wykorzystać do ustalenia, czy nastąpiło osłabienie piaskownicy, co wskazywałoby na przeprowadzenie jailbreakingu.
- ✓ Czasami aplikacje zainstalowane poprzez zewnętrzne repozytoria aplikacji mogą działać z podniesionymi uprawnieniami (np. użytkownika root) zamiast z uprawnieniami standardowego użytkownika urządzenia mobilnego. W takiej sytuacji pewne ograniczenia piaskownicy nie będą miały zastosowania. Dlatego też próbę zapisania pliku poza piaskownicą można potraktować jako rodzaj testu sprawdzającego zabezpieczenie urządzenia i jego piaskownicy.



Zabezpieczenie Aplikacji przed Modyfikacjami



Zabezpieczenie to ma na celu:

- ✓ Sprawdzenie spójności aplikacji i potwierdzenie, że zasoby statyczne aplikacji, takie jak pliki HTML i biblioteki współdzielone, a także wewnętrzne struktury kodu nie zostały zmodyfikowane.
- ✓ Sprawdzenie poprawności jest często implementowane za pomocą sumy kontrolnej. Popularnym wyborem jest zastosowanie CRC32 ze względu na szybkość działania i prostotę rozwiązania.
- ✓ Dużo lepszym podejściem jest wykorzystanie możliwości kompilatora LLVM (*low-level virtual machine*) oraz umożliwienie kodowi natywnemu w aplikacjach iOS czy Android przeprowadzania samodzielnej weryfikacji. Dzięki temu można zastosować optymalizacje wykorzystujące kompilację typu *just-in-time* (JIT) w LLVM do programowej kompilacji i modyfikacji kodu bajtowego LLVM. Ta strategia pozwala na automatyczne generowanie sumy kontrolnej dla funkcji skompilowanej z użyciem JIT oraz wstawienie procedur weryfikacji do plików binarnych podczas procesu kompilowania aplikacji, co odbywa się bez jakiejkolwiek modyfikacji kodu programu.



Zabezpieczenie przed Debuggowaniem



Debugowanie:

- ✓ to popularna technika używana podczas procesu inżynierii odwrotnej.
- ✓ Pozwala zrozumieć sposób działania aplikacji i umożliwia osobie atakującej modyfikację przepływu kontroli w aplikacji bądź jej wewnętrznych struktur kodu.
- ✓ Pozwala uzyskać wpływ na zachowanie aplikacji.
- ✓ Może umożliwić wydobywanie z aplikacji klucza kryptograficznego, np. zapisanego w aplikacji na stałe.

Zabezpieczenie przed debugowaniem:

- ✓ Uniemożliwienie dołączenia debuggera do aplikacji.
- ✓ Zastosowanie pomiaru czasu jako czynnika obliczeniowego, który zmienia sposób działania aplikacji lub szyfrowania, jeśli proces wykonywania aplikacji przedłuża się na skutek działania debuggera.
- ✓ Zaciemnianie kodu źródłowego aplikacji w celu utrudniania jej zrozumienia.



Zaciemnianie kodu źródłowego aplikacji



Techniki wykorzystywane do zaciemniania kodu źródłowego:

- ✓ Zaciemnianie nazw klas, metod, stałych i zmiennych.
- ✓ Wstawianie fikcyjnego kodu.
- ✓ Modyfikacja kontroli przepływu działania programu.
- ✓ Użycie szyfrowania ciągów tekstowych.
- ✓ Zmiana kodu w taki sposób, aby wydawał się znacznie bardziej skomplikowany. Odbywa się to na przykład przez użycie refleksji.
- ✓ Spłaszczanie kontroli przepływu działania programu.
- ✓ Wykorzystywanie takich fragmentów kodu, których działanie zależy od zdeszyfrowanych danych, bez których dalsza analiza i przewidywanie skutków działania kodu jest bardzo trudne.



System Android



System Android:

- ✓ to platforma mobilna, która ma największy udział w rynku ze wszystkich dostępnych mobilnych systemów operacyjnych,
- ✓ to zmodyfikowane, okrojone jądro systemu Linux,
- ✓ posiada maszynę wirtualną przeznaczoną do uruchamiania aplikacji utworzonych w języku programowania Java,
- ✓ zapewnia, iż każda aplikacja instalowana w urządzeniu z Androidem ma przypisywany unikatowy identyfikator użytkownika (*unique user identifier*, UID) i identyfikator grupy (*group identifier*, GID).
- ✓ wykorzystuje unikatowe oprogramowanie DVM (*dalvik virtual machine*), które zostało przygotowane do działania w sprzęcie charakteryzującym się ograniczonymi zasobami mocy obliczeniowej i pamięci oraz jest znacznie lżejsze niż zwykła maszyna wirtualna Javy.
- ✓ Od Android 4.4 zostało wprowadzone oprogramowanie Dalvik zamiennik środowiska uruchomieniowego o nazwie ART (*android runtime*), które ma oferować znacznie większą szybkość działania aplikacji w tym systemie.



Aplikacje Android



Aplikacje w systemie Android:

- ✓ Instalowane są z postaci skompresowanego archiwum ZIP (*.apk).
- ✓ Nowo instalowane aplikacje mają przypisywany identyfikator UID o kolejnej wartości, począwszy od 10000 aż do 99999.
- ✓ Każdy pakiet APK ma ściśle zdefiniowaną strukturę katalogów.
- ✓ Każda aplikacja Androida ma nadaną jej przez programistę unikatową nazwę pakietu (stosowane są tylko małe litery i odwrotna domena internetowa organizacji), np. com.google.translate, co umożliwia łatwe ich sortowanie.
- ✓ Mogą być instalowane w różny sposób i z różnych miejsc ze względu na to, iż platforma Android jest otwarta i bezpłatna, nie tylko ze sklepu Google Play Store, ale np. też z Samsung Apps, Amazon Appstore, GetJar, SlideMe, F-Droid.



Struktura Katalogów Pakietów APK



Struktura katalogów pakietów APK zawiera:

- ✓ `/assets` – pliki dołączone w tym katalogu zostaną dołączone do aplikacji.
- ✓ `/res` – zawiera układy, obrazy i pliki z zasobami, do których programista chce mieć dostęp w poziomym kodzie.
- ✓ `/lib` – zawiera biblioteki natywne, dołączane do aplikacji.
- ✓ `/META-INF` – zawiera certyfikat aplikacji oraz pliki zawierające informacje o pozostałych plikach umieszczonych w archiwum ZIP.
- ✓ `AndroidManifest.xml` – plik manifestu zawierających wszystkie informacje konfiguracyjne dotyczące aplikacji oraz parametry związane z zabezpieczeniem.
- ✓ `classes.dex` – zawiera kod bajtowy aplikacji Dalvik.
- ✓ `resources.asrc` – to zasoby, np. ciągi tekstowe aplikacji, które mogą być też umieszczone w katalogu `res`.



BusyBox w Androidzie



Busy Box jest zestawem standardowych narzędzi systemu Linux, stosowane również w Android, które działa w ramach kontekstu użytkownika i uprawnień procesu, który je wywołał:

acpid, add-shell, addgroup, adduser, adjtimex, arp, arping, ash, awk, base64, basename, beep, blkid, blockdev, bootchartd, brctl, bunzip2, bzip2, cal, cat, catv, chat, chattr, chgrp, chmod, chown, chpasswd, chpst, chroot, chrt, chvt, cksum, clear, cmp, comm, conspy, cp, cpio, crond, crontab, cryptpw, ctttyhack, cut, date, dc, dd, dealloctv, delgroup, deluser, depmod, devmem, df, dhcprelay, diff, dirname, dmesg, dnsd, dnsdomainname, dos2unix, du, dumpkmap, dumpleases, echo, ed, egrep, eject, env, envdir, envuidgid, etherwake, expand, expr, fakeidentd, false, fbset, fbsplash, fdflush, fdformat, fdisk, fgconsole, fgrep, find, findfs, flock, fold, free, freeramdisk, fsck, fsck.minix, fsync, ftpd, ftpget, ftpput, fuser, getopt, getty, grep, groups, gunzip, gzip, halt, hd, hdparm, head, hexdump, hostid, hostname, httpd, hush, hwclock, id, ifconfig, ifdown, ifenslave, ifplugd, ifup, inetd, init, insmod, install, ionice, iostat, ip, ipaddr, ipcalc, ipcrm, ipcs, iplink, iproute, iprule, iptunnel, kbd_mode, kill, killall, killall5, klogd, last, less, linux32, linux64, linuxrc, ln, loadfont, loadkmap, logger, login, logname, logread, losetup, lpd, lpq, lpr, ls, lsattr, lsmod, lsof, lspci, lsusb, lzcat, lzma, lzop, lzopcat, makedevs, makemime, man, md5sum, mdev, msg, microcom, mkdir, mkdosfs, mke2fs, mkfifo, mkfs.ext2, mkfs.minix, mkfs.vfat, mknod, mkpasswd, mkswap, mktemp, modinfo, modprobe, more, mount, mountpoint, mpstat, mt, mv, nameif, nanddump, nandwrite, nbd-client, nc, netstat, nice, nmeter, nohup, nslookup, ntpd, od, openvt, passwd, patch, pgrep, pidof, ping, ping6, pipe_progress, pivot_root, pkill, pmap, popmaildir, poweroff, powertop, printenv, printf, ps, pscan, pstree, pwd, pwdx, raidautorun, rdate, rdev, readahead, readlink, readprofile, realpath, reboot, reformime, remove-shell, renice, reset, resize, rev, rm, rmdir, rmmod, route, rpm, rpm2cpio, rtcwake, run-parts, runlevel, runsv, runsvdir, rx, script, scriptreplay, sed, sendmail, seq, setarch, setconsole, setfont, setkeycodes, setlogcons, setserial, setsid, setuidgid, sh, sha1sum, sha256sum, sha3sum, sha512sum, showkey, slattach, sleep, smemcap, softlimit, sort, split, start-stop-daemon, stat, strings, stty, su, sulogin, sum, sv, svlogd, swapoff, swapon, switch_root, sync, sysctl, syslogd, tac, tail, tar, tcpsvd, tee, telnet, telnetd, test, tftp, tftpd, time, timeout, top, touch, tr, traceroute, traceroute6, true, tty, ttysize, tunctl, udhcpc, udhcpd, udpsvd, umount, uname, unexpand, uniq, unix2dos, unlzma, unlzop, unxz, unzip, uptime, users, usleep, uudecode, uuencode, vconfig, vi, vlock, volname, wall, watch, watchdog, wc, wget, which, who, whoami, whois, xargs, xz, xzcat, yes, zcat, zcip.



Drozer



Drozer to narzędzie w Androidzie do:

- ✓ Zebrania informacji oraz oceny poziomu bezpieczeństwa w urządzeniu Android.
- ✓ Wyszukiwanie luk w zabezpieczeniach aplikacji lub urządzeń, pozwalające na określenie roli zainstalowanej aplikacji Android oraz prowadzenie interakcji z innymi aplikacjami i systemem operacyjnym w poszukiwaniu luk w zabezpieczeniach.
- ✓ Dostarczania programów pozwalających na wykorzystanie znanych luk w zabezpieczeniach. Odbywa się to przez utworzenie plików lub stron internetowych o złośliwym działaniu, które pozwalają na wykorzystanie znanych luk w zabezpieczeniach do instalacji drozer jako zdalnego narzędzia administracyjnego.
- ✓ Umożliwia atak na urządzenia Android poprzez wykorzystanie luk w zabezpieczeniach.
- ✓ <https://labs.mwrinfosecurity.com/tools/drozer/>



drozer



Komponenty i Frameworki Android



Aplikacje Android:

- ✓ Mają postać modułarną, zapewniającą komunikację między nimi.
- ✓ Wykorzystują przygotowane do ich tworzenia frameworki.
- ✓ System komunikacji międzyprocesowej *binder* narzuca sposób i ułatwia komunikację między procesami w odpowiednio zdefiniowany sposób.
- ✓ Korzystają z czterech standardowych komponentów kierowanych przez *bindera*:
 - **Czynność** – to widok aplikacji, w których użytkownik może przeprowadzać różne operacje.
 - **Usługi** – to komponent zapewniający wykonanie długo trwających zadań w tle, kontynuując ich działanie nawet po uruchomieniu innej aplikacji lub zamknięciu wszystkich czynności aplikacji zawierających tę usługę. Można je odczytać z karty *Running* w menedżerze aplikacji.
 - **Odbiorca komunikatów** – pozwala na rejestrację otrzymywania określonych zdarzeń systemowych lub aplikacji, np. o otrzymaniu SMSa, co uruchamia pewien fragment kodu kontekstowo. Dzięki temu unikamy stosowania mechanizmu odpytywania (*polling*), wykorzystując mechanizm oparty o zdarzenia.
 - **Dostawcy treści** – to magazyny danych aplikacji (zwykle SQLite <http://www.sqlite.org/>) zapewniające pobieranie, usuwanie i modyfikowanie danych.



Uruchamianie Aplikacji Android



Procedura uruchamiania aplikacji Android:

- ✓ Wykorzystuje pojedynczą maszynę wirtualną uruchomioną z systemem operacyjnym i nasłuchuje żądań uruchomienia nowych aplikacji.
- ✓ Po otrzymaniu tego rodzaju żądania wspomniana aplikacja maszyny wirtualnej wywołuje w samej sobie wywołanie `fork()` z parametrami nowej aplikacji przeznaczonej do uruchomienia.
- ✓ Proces nasłuchujący żądań uruchomienia nowych aplikacji jest określany mianem zygoty, w której podstawowe biblioteki są współdzielone między wszystkimi maszynami wirtualnymi.
- ✓ Kiedy użytkownik kliknie ikonę aplikacji, następuje sformułowanie intencji i wysłanie jej za pomocą `startActivity()`, obsługiwanej przez usługę menedżera czynności, która przekazuje do zygoty komunikat zawierający wszystkie parametry wymagane do uruchomienia aplikacji.
- ✓ Uprawnienia wymienionego katalogu powodują, że może być używany jedynie przez użytkownika system lub root.



Środowiska Uruchomieniowe w Androidzie



Środowisko Dalvik (od wersji Android 4.4):

- ✓ Interpretuje kod w trakcie działania aplikacji, używając podejścia JIT (*just-in-time*), czyli przeprowadza w locie kompilację kodu bajtowego na natywny.
- ✓ Tego rodzaju kompilacja wprowadza pewne opóźnienie i powoduje zużycie większej ilości zasobów.
- ✓ Wykorzystują pliki DEX.

Środowisko ART:

- ✓ Dokonuje kompilacji AOT (*ahead-of-time*), czyli powoduje konwersję aplikacji na kod natywny już podczas jej instalacji.
- ✓ Proces instalacji będzie trwał dłużej (niż dla Dalvik) i zabierze większą ilość miejsca na dysku, choć skróci się czas uruchamiania aplikacji, która ponadto będzie działała nieco szybciej.
- ✓ Wykorzystują pliki OAT.

Google dąży do zastąpienia środowiska Dalvik środowiskiem ART w przyszłości.



Zabezpieczenia platformy Android



Podstawowe założenia:

- ✓ Aplikacje działające w tym samym urządzeniu powinny mieć dostęp do swoich danych bez konieczności autoryzacji.
- ✓ Aplikacje działające w tym samym urządzeniu nie powinny mieć wpływu na działanie innych programów bez wyraźnej zgody użytkownika.
- ✓ Android oferuje wbudowane sposoby sprawdzania, która encja utworzyła daną aplikację.
- ✓ Tożsamość autora aplikacji jest zarządzana przez podpisanie kodu.
- ✓ Android nie pozwoli na zainstalowanie niepodpisanej aplikacji.
- ✓ Podpisanie pakietu Android odbywa się w sposób kryptograficzny za pomocą certyfikatu, którego klucz prywatny znajduje się w posiadaniu jedynie programisty danej aplikacji.
- ✓ Podpisywanie kodu ma na celu potwierdzenie tożsamości twórcy aplikacji w celu ustalenia pewnego poziomu zaufania w pozostałych aspektach modelu zapewniania bezpieczeństwa.
- ✓ Większość aplikacji wykorzystuje samodzielnie podpisane certyfikaty, których system operacyjny nie sprawdza w żadnym repozytorium internetowym, a certyfikat jest weryfikowany jedynie podczas instalowania aplikacji, a po jego wygaśnięciu aplikacja będzie działała normalnie. Uaktualnienia można pobrać i zainstalować tylko wtedy, gdy certyfikat nie utraci ważności.



Nadawanie Uprawnień na platformie Android



System nadawania uprawnień aplikacjom:

- ✓ Określa rodzaj zasobów, z których użytkownik pozwala korzystać aplikacji.
- ✓ Niebezpieczeństwem jest, gdy użytkownik nie czyta, na co pozwala aplikacji, zatwierdzając dostęp do zasobów.
- ✓ W celu otrzymania szczegółowych informacji o pakiecie wykorzystaj moduł `app.package.info`, który zwraca aplikacje korzystające z danych uprawnień, np.: `run app.package.list -p android.permission.READ_SMS`



Piaskownica Androida



Piaskownica aplikacji na platformie Android :

- ✓ Wykorzystuje wiele funkcji, które zostały opracowane w celu zagwarantowania, że jedna aplikacja nie będzie mogła zaszkodzić drugiej lub odczytywać jej danych bez wyraźnego zezwolenia.
- ✓ Prywatny katalog danych każdej aplikacji ma uprawnienia pozwalające na uzyskanie do niego dostępu jedynie użytkownikowi tej aplikacji.
- ✓ Próba uzyskania dostępu do tego katalogu z poziomu innego dowolnego użytkownika pozbawionego uprawnień skutkuje odmową dostępu.
- ✓ Jeśli grupa aplikacji korzysta z tej możliwości, wówczas będą działały w ramach dokładnie tego samego identyfikatora UID, co umożliwia zachowanie możliwości swobodnego odczytu i zapisu w ich prywatnych katalogach danych.
- ✓ System operacyjny sprawdza, czy wywołująca aplikacja ma odpowiednie uprawnienia pozwalające jej na uruchomienie danej czynności.
- ✓ Wszystkie wywołania API Androida do udostępnionych komponentów aplikacji są kontrolowane, a model uprawnień jest ściśle stosowany podczas uzyskiwania dostępu do wspomnianych komponentów.



Szyfrowanie Systemu Plików



Szyfrowanie systemu plików:

- ✓ Obejmuje zaszyfrowanie całego dysku bądź woluminu, a nie pojedynczych plików.
- ✓ Wymusza podanie hasła przez użytkownika jedynie na początku procesu uruchomienia urządzenia, a następnie szyfrowanie i deszyfrowanie wszystkich danych odczytywanych i zapisywanych na dysku odbywa się w sposób szybki i niewidoczny dla użytkownika.
- ✓ Oprogramowanie przeznaczone do pełnego szyfrowania dysku wykorzystuje zdefiniowany przez użytkownika klucz, na podstawie którego jest tworzony klucz użyty do szyfrowania danych.
- ✓ Wykorzystuje moduł dm-crypt w jądrze w celu niewidocznego dla użytkownika szyfrowania i deszyfrowania danych na poziomie bloków urządzenia.
- ✓ Kod PIN lub hasło wpisywane przez użytkownika w celu odblokowania urządzenia są używane również przez oprogramowanie pełnego szyfrowania dysku.
- ✓ Urządzenie Android generuje hasło, które następnie jest szyfrowane za pomocą klucza utworzonego na podstawie podanego przez użytkownika kodu PIN lub hasła zabezpieczającego. Uwaga: karta SD nie będzie uwzględniona w używanym przez Androida standardowym schemacie pełnego szyfrowania dysku.



Uzyskiwanie Uprawnień Użytkownika root



Istnieją dwa podstawowe sposoby na uzyskanie uprawnień użytkownika *root* w urządzeniu Android:

- ✓ przez wykorzystanie luki w zabezpieczeniach,
- ✓ za pomocą odblokowanego programu rozruchowego, np. poprzez modyfikację obrazu przeznaczonego do odzyskiwania systemu.



Inżynieria Odwrotna Aplikacji



Inżynieria Odwrotna Aplikacji:

- ✓ Umożliwia zrozumienie, co tak naprawdę dzieje się „pod maską” aplikacji, gdy nie masz dostępu do jej kodu źródłowego, ale tylko kodu wykonywalnego.
- ✓ Kluczowe znaczenie ma uzyskanie dostępu do pliku APK wybranej aplikacji, więc trzeba najpierw ustalić ścieżkę dostępu lokalizacji, w której znajduje się ten pakiet APK, co można zrobić używając powłoki ADB.
- ✓ Kod skompilowany i interpretowany (DEX, ODEX – zoptymalizowany DEX) może być dezasemblowany na postać czytelną dla człowieka, np. instrukcji Dalvik.
- ✓ Do dezasemblacji kodu (z DEX) służą również narzędzia, tj. smali, baksmali, IDA, JEB, apktool, Jadx, JAD.
- ✓ Narzędzie dex2jar jest przeznaczone do przeprowadzania konwersji plików Android DEX na postać zwykłych plików klas Javy.
- ✓ Pliki OAT dla środowiska ART są trudniejsze do deasemblacji.



Atak Powodujący Obniżenie Poziomu Ochrony Uprawnień



Scenariusz ataku powodującego obniżenie poziomu ochrony uprawnień (*protection level downgrade attack*):

1. Zainstalowana aplikacja o złośliwym działaniu definiuje zbiór uprawnień o dobrze znanych nazwach, przeznaczonych dla popularnych aplikacji na poziomie ochrony *normal*.
2. Użytkownik instaluje popularną aplikację, a system operacyjny ustala, że jedno z uprawnień zostało już wcześniej zdefiniowane. To prowadzi do zignorowania przez system operacyjny poziomu ochrony uprawnienia i pozostania przy parametrach zdefiniowanych przez aplikację o złośliwym działaniu.
3. Uprawnienia używane do ochrony komponentów aplikacji zostają obniżone do poziomu *normal* zamiast poziomu zapewniającego większe bezpieczeństwo, tj. *signature*. Wprawdzie uprawnienie zostało przez daną aplikację zdefiniowane z poziomem ochrony *signature*, ale dla systemu Android nie ma różnicy między tym uprawnieniem i zdefiniowanym przez aplikację o złośliwym działaniu.
4. Aplikacja o złośliwym działaniu może korzystać z niechronionych dłużej komponentów innej aplikacji, dla której nastąpiło obniżenie poziomu ochrony uprawnień.



Intencje



Intencja to obiekt danych ogólnie definiujący zadanie przeznaczone do wykonania:

- ✓ Może zawierać dane oraz wszystkie ważne informacje związane z akcją do wykonania na danych.
- ✓ Może zawierać tylko pojedyncze pole informacji.
- ✓ Może zostać przekazana do różnych wyeksportowanych komponentów w celu rozpoczęcia pracy z nimi.
- ✓ Aby zainicjować czynność, intencja może zostać przekazana z metodą `startActivity(Intent)` klasy `Context`. Metody `sendBroadcast(Intent)` i `startService(Intent)` mogą być używane z odbiorcami komunikatów i usługami.
- ✓ Obiekt intencji zalicza się do ogólnego typu, nie jest w żaden sposób dostosowany jedynie do typu otrzymywanego komponentu.
- ✓ Android oferuje dwa zupełnie odmienne typy intencji: jawne i niejawne.
- ✓ Jawna intencja bezpośrednio wskazuje komponent, który musi ją otrzymać, co powoduje pominięcie przeprowadzanego przez system operacyjny procesu rozwiązywania intencji i bezpośrednio dostarcza intencję konkretnemu komponentowi.
- ✓ Intencja niejawna nie określa komponentu, do którego musi zostać dostarczona.



Tapjacking



Tapjacking to mobilny odpowiednik znanej z Internetu luki w zabezpieczeniach o nazwie clickjacking (*UI redressing*):

- ✓ Która wyświetla specjalnie spreparowany interfejs użytkownika nad czynnością innej aplikacji, aby skłonić użytkownika do kliknięcia tego, czego nie zamierzał kliknąć.
- ✓ To jest możliwe dzięki funkcji interfejsu użytkownika o nazwie *toast*, który jest najczęściej używany do wyświetlania użytkownikowi niewielkich fragmentów informacji, z którymi nie ma on możliwości prowadzenia interakcji.
- ✓ Toast to nieinwazyjna i przezroczysta warstwa nad czynnością otwartą w danym momencie, którą można w pełni dostosować do własnych potrzeb i na przykład wyświetlić na całym ekranie z projektem przypominającym prawidłową czynność.
- ✓ Najniebezpieczniejsze jest to, że kiedy użytkownik spróbuje coś nacisnąć na tej nowej „czynności”, wprowadzone przez niego dane wejściowe mogą zostać otrzymane przez czynność znajdującą się pod toastem, np. do otworzenia czynności sklepu Google Play i zainstalowania aplikacji.



Schowek Systemowy Android



Globalny schowek systemowy Android działa podobnie jak w innych systemach operacyjnych i może być odczytywany przez wszystkie aplikacje zainstalowane w urządzeniu:

- ✓ System Android nie wymaga posiadania jakichkolwiek uprawnień do odczytu i zapisu danych w schowku.
- ✓ Schowek należy uznać za całkowicie niebezpieczną formę komunikacji między aplikacjami.
- ✓ Szczególnie niebezpieczne jest korzystanie z menedżera haseł, które w trakcie każdej operacji skopiowania hasła do schowka powoduje emisję zdarzenia, dzięki któremu aplikacja o złośliwym działaniu będzie mogła pobrać to hasło.



Testy penetracyjne



Testy penetracyjne:

- ✓ Służą do wykrywania typowych luk w zabezpieczeniach systemów informatycznych, w celu ich uszczelnienia.
- ✓ Uniemożliwienie wprowadzania błędnych danych lub przeprowadzania ataków typu injection.
- ✓ Zwykle są automatyczne.
- ✓ Nie zapewniają jednak zwykle pełnego wykrycia wszystkich możliwych sposobów włamania się systemu czy wycieku danych ze względu na różną specyfikę poszczególnych systemów.



Najczęściej Atakowane



Pewne cechy, funkcjonalności oraz uprawnienia aplikacji są wybierane na cel hackerów, gdyż umożliwiają dostęp do systemu bądź instalację aplikacji czy jej komponentów:

- ✓ Podanie kategorii `android.intent.category.BROWSABLE` pozwala na wywołanie z poziomu przeglądarki WWW czynności pozwalającej na zainstalowanie aplikacji.
- ✓ Nadanie uprawnień `INSTALL_PACKAGES` stanowi niezwykle cenny cel dla hackerów i aplikacja posiadająca takie uprawnienia powinna zostać natychmiast poddana analizie, gdyż posiada własny mechanizm uaktualniania, który może być wykorzystany do zainstalowania niechcianych pakietów.
- ✓ Android pozwala też aplikacjom na wczytywanie nowego kodu w trakcie działania programu za pomocą API Java Reflection poprzez niezabezpieczony kanał, a to oznacza umożliwienie wykonania kodu wywołanego zdalnie.



Dostęp do Urządzenia



Korzystając z różnych luk w zabezpieczeniach, hacker może uzyskać dostęp do urządzenia na wiele różnych sposobów:

- ✓ Dostęp na poziomie powłoki urządzenia
- ✓ Dostęp przez uszkodzenie pamięci przeglądarki WWW
- ✓ Dostęp na poziomie powłoki ADB
- ✓ Dostęp w ramach konta użytkownika systemu
- ✓ Dostęp w ramach konta użytkownika root
- ✓ Dostęp przez kanały komunikacji zewnętrznej: sieć telekomunikacyjna GSM, WiFi, Bluetooth, NFC
- ✓ Dostęp na poziomie niezabezpieczonych programów rozruchowych
- ✓ Dostęp przez ominięcie blokady ekranu
- ✓ Dostęp przez usunięcie wzorców blokady bądź haseł
- ✓ Dostęp przez zastosowanie ataków typu „man in the middle”, np. przez bezpłatny hosting sieci hackera czy organizację oferującą usługi internetowe.



Częste Cele Ataków



Do najczęstszych celów ataków hackerów należy:

- ✓ Uzyskanie dostępu do danych wrażliwych lub poufnych
- ✓ Kradzież plików z karty SD urządzenia
- ✓ Odczytywanie z książki adresowej
- ✓ Odczytywanie i wysyłanie wiadomości SMS
- ✓ Odczytanie danych geolokacyjnych użytkownika
- ✓ Przechwycenie ekranu urządzenia użytkownika
- ✓ Nagrywanie dźwięku z mikrofonu
- ✓ Nagrywanie wideo z kamery
- ✓ Wyodrębnienie kluczy WiFi
- ✓ Łamanie wzorców, kodów i haseł, np. PINów.



Podstawowe mechanizmy obronne



Chcąc bronić się przed atakami należy przyjrzeć się możliwościom obrony i zdefiniować mechanizmy obronne:

- ✓ Przegląd punktów wejścia w komponentach aplikacji
- ✓ Wykrywanie węszenia w menedżerze zadań
- ✓ Wykrywanie tapjackingu
- ✓ Wyłączenie dodawania elementów do słownika Android
- ✓ Obrona przed atakami polegającymi na wstrzykiwaniu fragmentów kodu
- ✓ Ustalenie granic zaufania
- ✓ Maskowanie i zabezpieczanie haseł
- ✓ Oczyszczanie aktywności zaliczanych do kategorii BROWSABLE
- ✓ Zabezpieczanie dostawcy treści, odbiorcy komunikatu
- ✓ Bezpieczne tworzenie plików i katalogów
- ✓ Bezpieczne udostępnianie plików innym aplikacjom
- ✓ Użycie szyfrowania

Bibliografia, Odnosiniki i Literatura

1. Dominic Chell, Tyrone Erasmus, Shaun Colley, Ollie Whitehouse, „Bezpieczeństwo aplikacji mobilnych. Podręcznik hakera”, Helion, 2018.
2. Marcin Karbowski, Podstawy Kryptografii, Wydanie III, Helion, 2015.
3. Michael, Welschenbach, Kryptografia w C i C++, MIKOM, Warszawa, 2002
4. Kryptografia: <http://www.cryptography.ovh.org>
5. Kryptoanaliza: <http://www.crypto-it.net/pl/ataki/index.html>
6. Szyfr Cezara: http://eduinf.waw.pl/inf/alg/001_search/0063.php
7. Algorytm RSA: http://eduinf.waw.pl/inf/alg/001_search/0067.php
8. Szyfr Enigmy: http://eduinf.waw.pl/inf/alg/001_search/0066.php
9. Algorytmy struktury danych: http://eduinf.waw.pl/inf/alg/001_search/index.php



Google: [Horzyk](#)