Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Marcel Kołodziejczyk
Nr albumu: 219533

Luki w bezpieczeństwie systemu operacyjnego Android

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dra Marcina Peczarskiego
Instytut Informatyki

Czerwiec 2013

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

Streszczenie

krótkie streszczenie pracy

Słowa kluczowe

android, arm, atak, bezpieczeństwo, przepełnienie bufora, metasploit, exploit, shellcode

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

D. Software
D.4. Operating Systems
D.4.6. Security and Privacy Protection

Tytuł pracy w języku angielskim

Vulnerabilities in Android operating system

Spis treści

Todo list

Wprowadzenie

Rozdział 1

Platforma sprzętowa i programowa

Android jest systemem operacyjnym i zestawem aplikacji dedykowanym przede wszystkim dla urządzeń przenośnych z ekranami dotykowymi, takimi jak np. smartphone, tablet. Jądro systemu, zostało oparte na jądrze Linuksa. System ten został zaprojektowany i stworzony głównie z myślą o urządzeniach wyposażonych w procesor w architekturze ARM, aczkolwiek podejmowane są prace nad dostosowaniem Androida do innych architektur, np. x86.

W rozdziale tym zostaną opisane podstawy architektury procesorów ARM. Następnie zostanie omówiona architektura oraz model bezpieczeństwa systemu Android.

1.1. Architektura procesorów ARM

ARM jest 32-bitową architekturą procesorów typu RISC. Główne jej cechy to:

- proste tryby adresowania
- instrukcje stałej długości co ułatwia adresowanie
- architektura typu *load/store* - operacje wykonywane są na rejestrach a nie bezpośrednio na pamięci
- duża liczba 32-bitowych rejestrów
- zredukowana liczba instrukcji

Z biegiem czasu ukazywały się kolejne wersje architektury ARM. Niniejsza praca bazuje na wersji 7 (ARMv7), które jest obecnie najbardziej powszechnie wykorzystywana w urządzeniach przenośnych.

1.1.1. Thumb-2

Instrukcje ARM są stałej, 32-bitowej długości. W celu zwiększenia gęstości kodu został wprowadzony drugi, uproszczony zestaw 16-bitowych instrukcji Thumb-2. Ponieważ w obydwu trybach adresy instrukcji muszą być odpowiednio wyrównane, ostatni bit adresu może być wykorzystany w celu zmiany trybu pracy procesora. Instrukcja skoku do adresu, którego ostatni bit jest zapalony, wymusza zmianę trybu na Thumb-2 i dalsze wykonywanie instrukcji spod adresu odpowiednio wyrównanego. Analogicznie instrukcja skoku do parzystego adresu powoduje przejście w 32-bitowy zestaw instrukcji ARM.

1.1.2. Rejestry

Z punktu widzenia programisty dostępnych jest szesnaście 32-bitowych rejestrów R0-R15. Trzy z nich mają dedykowane przeznaczenie:

- SP (Stack Pointer) - R13 - wskaźnik stosu
- LR (Link Register) - R14 - zawiera adres następnej instrukcji przy instrukcjach skoku wywołania podprogramu
- PC (Program Counter) - R15 - przechowuje adres następnej instrukcji

Dodatkowo występuje rejestr statusowy procesora CPSR (ang. Current Processor Status Register). Przechowuje on m. in. flagi Negative, Zero, Carry, oVerflow. Większość instrukcji może być wykonywanych warunkowo, w zależności od stanu tych flag.

Szczegółowe informacje na ten temat można znaleźć w [12].

1.1.3. Standard wywołania procedur

Do wykonywania procedur służą następujące instrukcje:

- **B** - Branch
- **BL** - Branch with Link
- **BX** - Branch and Exchange
- **BLX** - Branch with Link and Exchange

Instrukcja *Branch* umożliwia wykonanie skoku o maksymalnie 32 MB w przód lub w tył od bieżącej instrukcji. *Branch with link* dodatkowo zachowuje adres powrotu w rejestrze LR (R14). Pozostałe dwie instrukcje jako argument przyjmują rejestr - skok jest wykonywany do adresu, jaki znajduje się w przekazanym rejestrze.

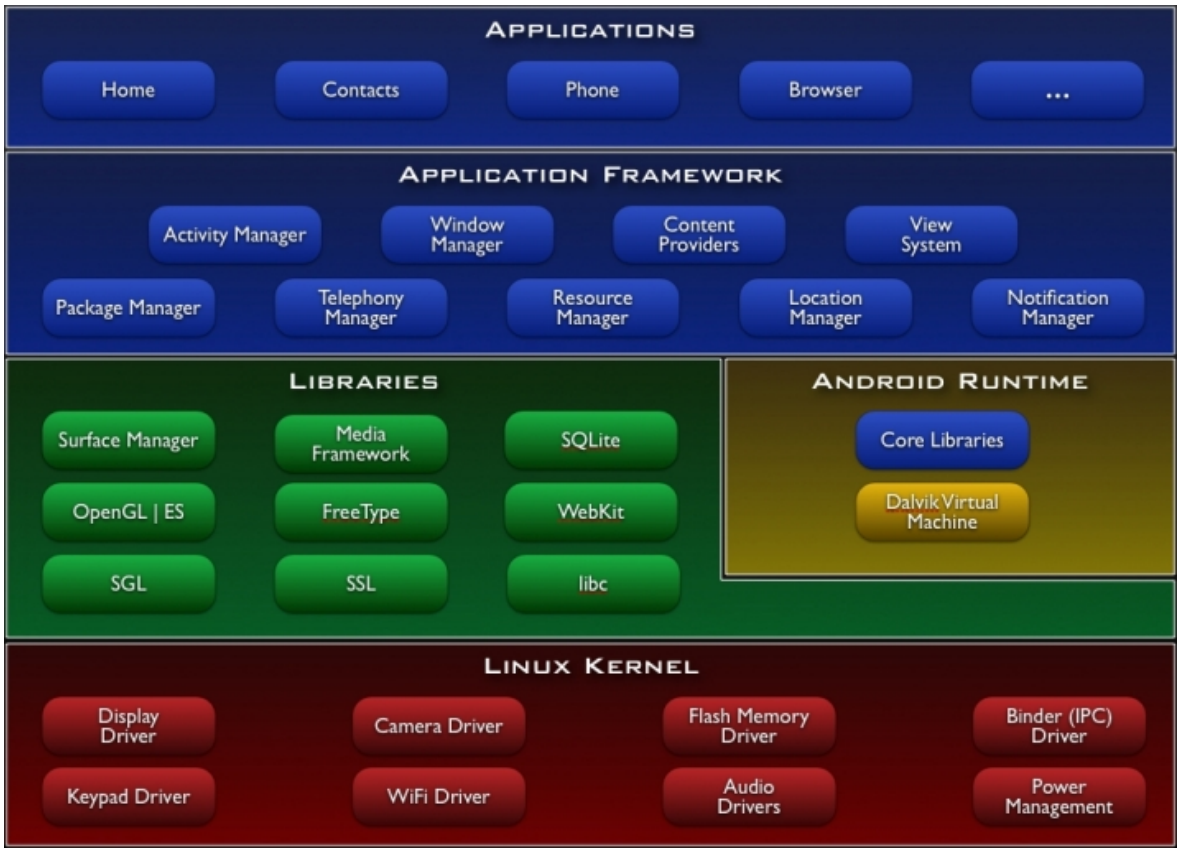
Architektura ARM określa następujące zasady wywoływania procedur:

- Do przekazywania argumentów i zwracania wyniku procedury używane są rejestry R0-R3. Kolejne parametry mogą być przekazywane na stosie.
- rejestry R4-R11 mogą być wykorzystywana do przechowywania zmiennych lokalnych
- Zawartość rejestrów R4-R12 powinna być zachowana w trakcie wykonania procedury. Zazwyczaj w prologu procedury rejestry te są odkładana na stos, aby przywrócić ich wartości w epilogu.
- Stos rośnie w kierunku mniejszych adresów pamięci.

Kompletną dokumentację standardu wołania procedur można znaleźć w [13].

1.2. Architektura systemu Android

Diagram 1.1 przedstawia najważniejsze komponenty systemu Android. Zostaną one w skrócie omówione w kolejnych paragrafach.



Rysunek 1.1: Główne komponenty systemu Android [1]

źródło: `http://developer.android.com/about/versions/index.html`

1.2.1. Jądro systemu

Podstawową warstwą zapewniającą interakcje ze sprzętem jest jądro systemu. Jądro systemu Android od wersji 4.0 (*Ice Cream Sandwich*), stanowi nieznacznie zmodyfikowane jądro Linuxa w wersji 3.0.x. Wcześniejsze wydania systemu opierały się na jądrach z linii 2.6.x. Najważniejsze zmiany w stosunku do głównej wersji to:

- dodatkowe mechanizmy komunikacji międzyprocesowej i zdalnego wołania metod (*Android Binder*),
- nowy podsystemem pamięci dzielonej `ashmem` i alokator pamięci `pmem`,
- `logger` - wsparcie jądra dla narzędzia `logcat`,
- dodatkowe mechanizmy ograniczające dostęp do wybranych funkcjonalności sieciowych (*paranoid network security*).

Sytuacja, w której jądro Androida jest rozgałęzieniem w stosunku do głównej linii Linuxa jest bardzo istotna z punktu widzenia bezpieczeństwa systemu. Wszelkie zmiany wprowadzane w jądrze Linuxa, w tym niektóre poprawki bezpieczeństwa, pojawiają się w zmodyfikowanej wersji dla Androida ze sporym opóźnieniem. Dodatkowo opóźnienie to jest powiększone przez system aktualizacji systemu, co zostanie opisane w paragrafie 1.5. Z tego powodu istnieje

bardzo wiele powszechnie znanych luk w jądrze Androida, pozwalających m. in. na eskalację uprawnień procesu. Pozwala to na tymczasowe lub permanentne uzyskanie uprawnień administratora systemu (tzn. „rootowanie" systemu). Jest to bardzo często wykorzystywane przez zwykłych użytkowników do wykonania niektórych czynności administracyjnych, np. zmiany konfiguracji systemu, odinstalowanie wybranych aplikacji systemowych, a nawet wgranie zupełnie nowego obrazu systemu. Istnieje wiele narzędzi umożliwiających wykonanie tego procesu zwykłemu, niezaawansowanemu użytkownikowi.

1.2.2. Biblioteki

System Android posiada szereg popularnych bibliotek napisanych w C/C++, używanych przez różne komponenty poprzez „framework" aplikacji. Możliwe jest także skorzystanie z tych bibliotek w kodzie natywnym napisanym w C/C++. Przykładowe biblioteki to:

- `libc` - standardowa biblioteka C, zoptymalizowana dla urządzeń wbudowanych,
- `webcore` - silnik przeglądarki internetowej. Może być wykorzystany w innych aplikacjach, które potrzebują wyświetlić stronę HTML, mp. w kliencie poczty e-mail,
- `sqllite` - lekka relacyjna baza danych,
- `OpenGL` - używana podczas renderowania grafiki dwu- i trójwymiarowej,
- zestaw bibliotek multimedialnych dostarczających kodeki wybranych formatów plików.

1.2.3. Środowisko czasu wykonania

Środowisko czasu wykonania systemu Android składa się z maszyny wirtualnej Dalvik oraz podstawowych bibliotek Javy.

Dalvik jest maszyną wirtualną, która została stworzona specjalnie dla systemu Android, w celu zapewnienia odpowiedniej wydajności na urządzeniach z mniejszymi zasobami, jak np. telefony komórkowe. Nie jest maszyną wirtualną Javy i używa własnego kodu bajtowego w formacie `.dex` (Dalvik executable). Możliwa jest jednak konwersja kodu bajtowego Javy do kodu Dalvika. W przeciwieństwie do wirtualnej maszyny Javy, która jest maszyną stosową, Dalvik jest maszyną rejetrową. Dalvik umożliwia uruchomienie wielu aplikacji jednocześnie tworząc klika instancji maszyny wirtualnej. Zapewnia izolację, zarządzanie pamięcią oraz wielowątkowość.

1.2.4. Aplikacje

System Android posiada kilka podstawowych aplikacji zapewniających podstawowe funkcjonalności nowoczesnego telefonu komórkowego, np. klient SMS, aplikacja do wykonywania połączeń głosowych, klient poczty, przeglądarka internetowa, zarządca aplikacji. Bardzo wiele funkcjonalności jest wydzielonych do osobnych komponentów tworząc tzn. „framework". Jest on zaprojektowany w sposób, który umożliwia zastąpienie jego dowolnego komponentu na inny zapewniający taką samą funkcjonalność.

1.3. Model bezpieczeństwa Androida

Platforma Android została zaprojektowana w taki sposób, aby możliwe było także zainstalowanie aplikacji z potencjalnie niezaufanych źródeł. Jest to odmienny model niż w przypadku

telefonów iPhone (z systemem operacyjnym iOS), gdzie wszystkie aplikacje mogą być jedynie zainstalowane z jednego źródła (*Apple store*) i mogą być zweryfikowane przed upublicznieniem. Z tego powodu system Android posiada mechanizmy bezpieczeństwa działające na wielu poziomach. Izolowanie aplikacji wykorzystuje mechanizmy kontroli dostępu do systemu plików, jakie udostępnia jądro Linuxa oraz mechanizm nadawania wybranych uprawnień zatwierdzanych przez użytkownika w trakcie instalacji aplikacji. Aplikacje muszą być także podpisane przy użyciu kryptografii klucza publicznego, jednak ten mechanizm umożliwia jedynie wiarygodne zidentyfikowanie autora aplikacji.

System Android zbudowany jest na bazie standardowego jądra Linuxa z nieznacznymi modyfikacjami, dlatego też posiada uznaniową kontrolę dostępu (*Discretionary Access Control - DAC*) na poziomie systemu plików, która opiera się na identyfikatorach użytkowników (*uid*) i grup (*gid*). Nad jądrem Android używa własnego zbioru bibliotek i usług. Aplikacje mogą być tworzone w Javie i są kompilowane do kodu bajtowego maszyny wirtualnej Dalvik. Aplikacje lub fragmenty mogą być również tworzone w C/C++ a następnie wywoływane z Javy poprzez interfejs JNI (*Java Native Interface*).

1.3.1. Uruchamianie aplikacji w *„piaskownicy”*

Aplikacje zainstalowane w systemie są ograniczone w *„piaskownicy”*, która jest zdefiniowana poprzez unikalny *uid* i odpowiadający *gid*. Identyfikatory te są tworzone dynamicznie podczas instalowania aplikacji. Nazwy użytkownika i grupy są identyczne i składają się z prefiksu `app_` oraz identyfikatora. Każda aplikacja używa innego identyfikatora użytkownika i grupy, co gwarantuje pełną izolację na poziomie systemu plików. Dostęp do plików i usług systemowych jest także znacząco ograniczony, zazwyczaj tylko do odczytu. Wywołanie funkcji lub usługi spoza *„piaskownicy”* jest możliwe poprzez odpowiednie API, które wymaga odpowiednich uprawnień aplikacji, co zostanie opisanie w punkcie 1.3.3.

Ograniczenia *„piaskownicy”* są wymuszane przed jądro systemu i poprzez odpowiednie usługi przestrzeni użytkownika, dlatego też dotyczą wszystkich aplikacji, włącznie z kodem natywnym wywoływanym bezpośrednio poprzez interfejs JNI lub przy użyciu wywołania systemowego `exec`. Jeżeli aplikacje potrzebują współdzielić dane, na przykład na poziomie systemu plików, powinny zadeklarować wspólny identyfikator użytkownika (*uid*) w manifeście. Jest to możliwe tylko wtedy, gdy aplikację są podpisane przy użyciu tego samego klucza prywatnego. Identyfikatory użytkowników i odpowiadające im uprawnienia są przechowywane w pliku `data/system/packages.xml` i są do odczytu przez wszystkie aplikacje zainstalowane na urządzeniu.

1.3.2. Kontrola dostępu do systemu plików

Uznaniowa kontrola dostępu do systemu plików w Androidzie jest zrealizowana przy użyciu tradycyjnych Unixowych uprawnień. Pliki tworzone przez aplikacje domyślnie mają ustawione uprawnienia na `rw-rw----` (0660 w notacji ósemkowej). Z tego powodu aplikacje zainstalowane z różnymi identyfikatorami użytkownika i grupy nie mogą czytać, modyfikować ani wykonywać wzajemnie swoich plików. Pliki mogą być jednak jawnie udostępniane poprzez użycie flag `MODE_WORLD_READABLE` i `MODE_WORLD_WRITABLE` podczas tworzenia ich w API Javowym lub poprzez wywołanie systemowe `chmod` w natywnym kodzie C/C++. Tworzenie plików z odpowiednimi uprawnieniami leży w gestii aplikacji.

Katalog z danymi aplikacji domyślnie znajduje się w `/data/data/<nazwa pakietu>/` i posiada następującą strukturę:

- `databases` - domyślny katalog do przechowywania baz danych sqlite

- `lib` - zawiera wszystkie natywne biblioteki używane przez aplikacje, skopiowane podczas instalacji
- `files` - jest to katalog, gdzie domyślnie są tworzone pliki przez aplikację
- `shared_prefs` - zawiera XML-owe pliki konfiguracyjne aplikacji

Standardowe, pre-instalowane aplikacje (np. `com.android.camera`) zazwyczaj także używają powyższej struktury katalogów, jednak ich lokalizacja może być odmienna na niektórych urządzeniach, np. telefony marki *Samsung* używają ścieżki `/dbdata/databases/<nazwa pakietu>/` dla pre-instalowanych aplikacji. Jest to istotnie z punktu widzenia twórcy exploitów, aby złośliwy kod potrafił się dostosować do struktury katalogów używanej przez dane urządzenie.

W trakcie rozruchu systemu różne części systemu plików są montowane z odmiennymi opcjami. Katalog `/data` używa opcji `'rw,nosuid,nodev,relatime'`, co m. in. oznacza, że flaga plików `setuid` nie będzie respektowana. Pliki wykonywalne będą zawsze uruchamiane z uprawnieniami użytkownika wykonującego program, a nie właściciela pliku.

Katalog `/system` jest montowany z opcjami `ro,relatime`, co powoduje, że cała partycja jest jedynie do odczytu. Warto zwrócić uwagę, że w tym przypadku opcja `nosuid` nie jest używana, ponieważ system Android korzysta z plików z flagą `setuid` w tej lokalizacji.

1.3.3. Uprawnienia aplikacji

Aplikacje w celu wyjścia z *„piaskownicy”* (zarówno na poziomie systemu pików, jak i wywoływania chronionych funkcji API systemu), muszą mieć wcześniej nadane odpowiednie uprawnienia. Wymagane przez aplikację funkcjonalności są wyświetlane w trakcie instalacji i muszą być zatwierdzone przez użytkownika. Nie jest jednak możliwe selektywne wybranie uprawnień.

Aplikacja, aby uzyskać dostęp do chronionych funkcji API, musi zawierać uprawnienia, takie jak na przykład:

```
<uses-permission android:name="android.permission.CALL_PHONE" />
```

w pliku `AndroidManifest.xml`. Każda próba wykonania chronionej funkcji API bez odpowiednich uprawnień spowoduje podniesienie wyjątku `SecurityException`.

1.3.4. Podpisywanie aplikacji

Każda instalowana aplikacja w systemie Android musi być podpisana przez twórcę przy użyciu klucza prywatnego wraz z odpowiadającym certyfikatem. Możliwe jest jednak używanie kluczy prywatnych potwierdzonych przez dowolne centrum certyfikacji, a także certyfikatów samopodpisanych. Mechanizm podpisywania aplikacji nie ma większego znaczenia z punktu widzenia bezpieczeństwa systemu. Jedyną korzyścią jaką wnosi jest potwierdzenie tożsamości autora instalowanej aplikacji. W przyszłości system podpisywania może zostać wykorzystany w celu ograniczenia dostawców aplikacji do jedynie uprzednio zweryfikowanych, podobnie jak to jest w przypadku konkurencyjnego iPhone'a firmy Apple. W obecnych wydaniach systemu Android funkcjonalność taka nie jest zaimplementowana.

Mechanizm podpisywanie aplikacji umożliwia programom współdzielenie zasobów, co zostało opisane w 1.3.1

1.4. Narzędzia programistyczne

Twórcy systemu Android udostępnili bardzo dobre narzędzia niezbędne do rozwijania i testowania aplikacji na ten system: SDK (Software Development Kit) i NDK (Native Development Kit).

Android SDK jest podstawowym zestawem narzędzi przydatnym dla bardziej zaawansowanych użytkowników systemu oraz programistów . Najważniejsze komponenty tego pakietu to:

- emulator wraz z zestawem obrazów kolejnych wersji systemu pozwala uruchomić wirtualny obraz systemu. Możliwe jest zdefiniowanie sprzętowych parametrów emulowanego obrazu, m. in. rozmiar pamięci RAM, rozdzielczość i typ ekranu. Dzięki temu można wykonywać testy na wielu konfiguracjach. Wirtualizacja opiera się na emulatorze QEMU.
- `adb` (Android Debug Bridge) - narzędzie umożliwiające komunikację z urządzeniem i debugowanie aplikacji. Pozwala wykonywać szereg czynności diagnostycznych, np. przesłanie pliku z/do urządzenia, wypisanie logów systemowych, uruchomienie konsoli, zainstalowanie/odinstalowanie aplikacji. Narzędzie to może być także wykorzystywane do pracy z uruchomionym wirtualnym obrazem.
- biblioteki dla kolejnych wydań systemu umożliwiają tworzenie aplikacji na daną platformę

Android NDK jest zestawem narzędzi dedykowanych dla systemu Android, który został stworzony, aby umożliwić tworzenie aplikacji lub ich części w C/C++. W jego skład wchodzą między innymi:

- różne wersje kompilatora gcc dla wybranych architektur procesora (ARM, x86, ...).
- standardowe narzędzia do debugowania i optymalizacji kodu, np. `gdb`, `gdbserver`, `gcov`, `ar`, `ld`, `objdump`.
- pliki nagłówkowe i skompilowane statyczne i dynamiczne biblioteki, np. libc, libstdc++, libz, libm.

Obydwa pakiety posiadają bogatą dokumentację oraz zawierają liczne przykłady użycia. Przykłady w dalszej części pracy będą używały powyższych narzędzi.

1.5. Aktualizacje systemu

System Android posiada wbudowany mechanizm aktualizacji całego systemu. Jednak nowe wersje systemu są dostarczane przez producenta ze sporym opóźnieniem. Wielu producentów urządzeń wraz ze standardową dystrybucją systemu dostarcza dodatkowe oprogramowanie, np. *HTC Sense* firmy HTC, nakładka graficzna *TouchWiz* firmy Samsung. Źródła systemu zazwyczaj są nieznacznie modyfikowane przez producenta dla każdego modelu telefonu, gdyż wymagają, np. dodatkowych sterowników urządzeń. Niektóre telefony posiadają także dodatkowe oprogramowanie operatora telefonii komórkowej. Nowe wersje systemu w pierwszej kolejności pojawiają się na urządzeniach z linii *Google Nexus*, ponieważ nie zawierają one żadnych dodatkowych rozszerzeń producenta, ani firm telekomunikacyjnych. Bardzo wiele modeli nie otrzymuje dalszego wsparcia ze strony producenta. Jest to głównie spowodowane niewystarczającą platformą sprzedową danego modelu. Bywa także, że decyzja o zaprzestaniu

Wersja	Nazwa kodowa	Udział
2.1	Eclair	1.7%
2.2	Froyo	4.0%
2.3.3 - 2.3.7	Gingerbread	39.7%
4.0.3 - 4.0.4	Ice Cream Sandwich	29.3%
4.1.x	Jelly Bean	23.0%
4.2.x	Jelly Bean	2.0%

Tabela 1.1: Udział wersji systemu Android[2](stan na 02-04-2013)

źródło: `http://developer.android.com/about/dashboards/index.html`

wydawania aktualizacji ma przyczyny biznesowe i productent nie ponosi nakladów, na mało popularne urządzenia.

Tabela 1.1 przedstawia udział najpopularniejszych wersji systemu. Zauważyć można, że największy udział mają wersję, których data premiery miała miejsce ponad dwa lata temu.

Rozdział 2

Techniki ataków i sposoby na przeciwdziałanie

2.1. Klasyczny błąd przepełnienie bufora

2.2. Technika „heap spray”

2.2.1. NX bit

W celu ochrony przed tego typu takimi w wersji 6 architektury ARM wprowadzona została technologia NX bit (No Execute). Umożliwia ona systemowi operacyjnemu oznaczyć wybrane strony pamięci jako niewykonywalne. Gdy bit NX dla danej strony jest ustawiony, próba wykonania zawartości tej strony jako kodu kończy się wygenerowaniem wyjątku, zgłaszanego systemowi operacyjnemu, co powoduje przerwanie wykonywania programu. Bit NX powinien być ustawiony dla wszystkich stron procesu, z wyjątkiem programu i bibliotek oraz świadomie dozwolonych przez program wyjątków.

Technologia „NX bit” jest wspierana przez Androida od wersji 2.3.

wyko dzięki któremu system operacyjny

2.3. Technika „return to library” (Ret2Libc)

Rozdział 3

Tworzenie payloadów

Rozdział 4

Przykłady ataków i ich implementacja w narzędziu Metasploit

4.1. CVE-2010-1119

4.2. CVE-2010-1807

Rozdział 5

Podsumowanie

Bibliografia

[1] Anthony Desnos, Geoffroy Gueguen *Android: From Reversing to Decompilation*, Black Hat, Abu Dhabi, 2011

[2] S. Höbarth, R. Mayrhofer, *A framework for on-device privilege escalation exploit execution on android*, IWSSI/SPMU 2011: 3rd International Workshop on Security and Privacy in Spontaneous Interaction and Mobile Phone Use, colocated with Pervasive 2011, czerwiec 2011. dostępne na `http://www.medien.ifi.lmu.de/iwssi2011/`

[3] Gaurav Kumar, Aditya Gupta, *A Short Guide on ARM Exploitation*, `http://www.exploit-db.com/wp-content/themes/exploit/docs/24493.pdf`

[4] Yves Younan, Pieter Philippaerts, *Alphanumeric RISC ARM shellcode*, Phrack, 66, czerwiec 2009

[5] Joshua Hulse, *Buffer Overflows: Anatomy of an Exploit*, `http://packetstormsecurity.com/files/108549/Buffer-Overflows-Anatomy-Of-An-Exploit.html`

[6] Emanuele Acri, *Exploiting Arm Linux Systems*, `http://packetstormsecurity.com/files/98376/Exploiting-ARM-Linux-Systems.html`

[7] Collin Mulliner, Charlie Miller, *Fuzzing the Phone in your Phone*, Black Hat USA, 2009

[8] Jonathan Salwan, *How to Create a Shellcode on ARM Architecture*, `http://www.exploit-db.com/papers/15652/`

[9] Dustin „I)ruid" Trammel, *Metasploit Framework Telephony*, Black Hat USA, 2009

[10] Itzhak Avraham, *Non-Executable Stack ARM Exploitation*, Black Hat DC, 2011

[11] jip@soldierx.com, *Stack Smashing On A Modern Linux System*, `http://www.soldierx.com/tutorials/Stack-Smashing-Modern-Linux-System`

[12] ARM Ltd. *Arm architecture reference manual.*

[13] ARM Ltd. *Procedure call standard for the arm architecture.*

[14] Metasploit framework, `http://www.metasploit.com`

[15] Android project, `http://developer.android.com`

[16] The WebKit Open Source Project, `http://www.webkit.org`