内部構造から探る Android への脆弱性攻撃とマルウェアの脅威

Inside Android Security

Fourteenforty Research Institute, Inc.

株式会社 フォティーンフォティ技術研究所

http://www.fourteenforty.jp

新技術開発室　大居　司

背景：Android とマルウェア

- Android が国内のスマートフォン向け OS のシェアにおいて
  iOS を抑え首位を獲得したことが報道される（2011年5月，7月）
- 一方で、Android におけるマルウェアの問題は深刻になりつつある
  - AdaptiveMobile (1) によれば、2010年の Android マルウェア
    感染件数は前年比 4 倍以上となった（2010年12月）
  - 通常の方法では削除できない Android マルウェアが
    相次いで発見される（2011年3月，6月）
- 複数のベンダから、ウイルス対策を含む
  モバイルセキュリティ製品が発売される（主要なものは 2011年3月～）
  - Android は、PC と同じくセキュリティ製品で
    守る必要のある端末になりつつある

(1) http://www.adaptivemobile.com/

背景：Android と脆弱性攻撃

- 一般的な OS においては対抗機能の整備が進んでいる（2003年～）
- iOS の場合: JailbreakMe（2007年～）
  - ブラウザから iPhone などの端末の管理者権限を取得し、場合によっては悪用できることを証明した
- Android の場合: DroidDream（2011年3月）
  - システム内部の脆弱性を使用し、通常の方法では削除できない領域にバックドアアプリをインストールする
- 脆弱性攻撃によって悪意あるコードを実行する余地がある

現状の課題

- 脆弱性攻撃
  - 外部から攻撃できるのか
  - 攻撃を防ぐような措置は取られているのか
- マルウェア
  - 現状どのようにシステムに入り込んでいるか
  - 悪意ある行動を起こせる範囲は?
- ウイルス対策ソフト
  - どのように動作するか
  - これで Android 端末を守れるのか

- これらについて、主に技術的な側面から見ていく

アジェンダ

- 低レイヤーのセキュリティ
  - Linux カーネル / Android ベースシステム
- Android アプリケーション レイヤーの仕組み
  - パッケージ
  - パーミッション
  - インテント / アクティビティ / ブロードキャスト
- 脅威とそれへの対抗策
  - マルウェア
  - root 化
  - ウイルス対策ソフトとその問題

Linux カーネルのメモリ保護機構と Android における現状

低レイヤーのセキュリティ

低レイヤーにおけるセキュリティの意義

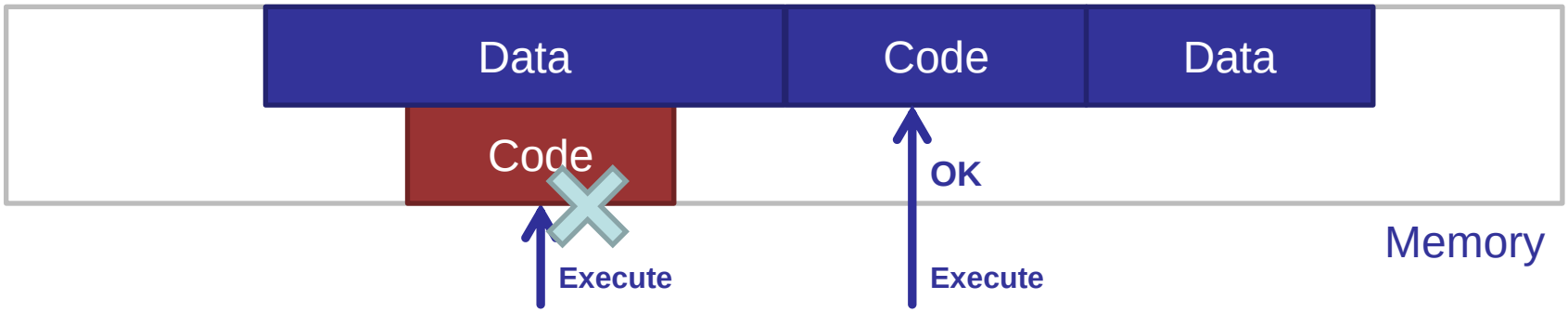
- Android アプリは Java で書かれているが、
  それでも多くのネイティブコードに依存している
  - Linux カーネル
  - Dalvik VM
  - WebKit
- Android アプリは自らネイティブコードを実行できる
  - 安全でないコードによる危険性
  - マルウェア実行や exploit の危険性
- Android においては、低レイヤーの部分である Linux カーネルや
  フレームワークのセキュリティが極めて重要である。

低レイヤーのセキュリティ機能

	Android -2.2	Android 2.3-,3.0-	備考
DEP (スタック)	× (1)	○(1)	対応: iOS 2.0-
DEP (その他)	× (2)	○	
W^X の強制	×	×	
ASLR (スタック)	○	○	対応: iOS 4.3
ASLR (ヒープ)	×	×	
ASLR (モジュール)	×	×	部分的対応: iOS 4.3

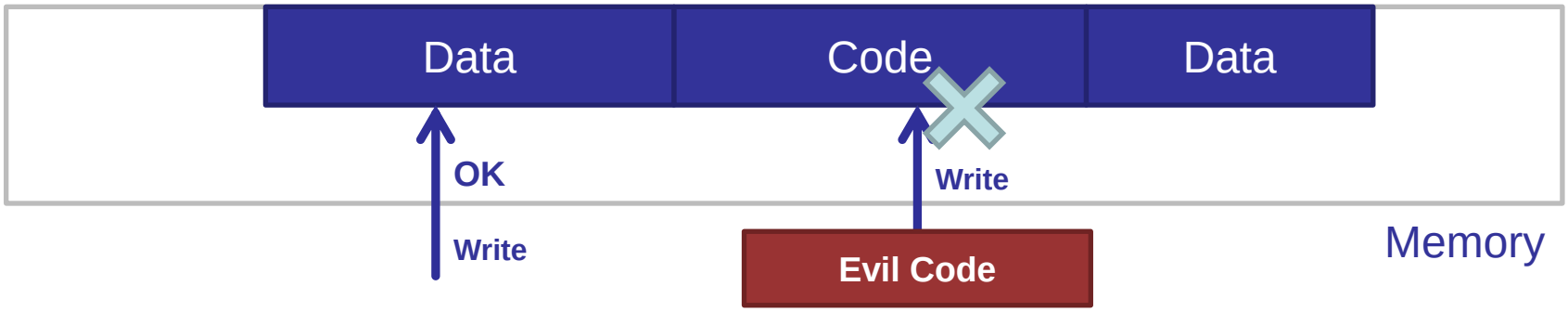
(1) ネイティブアプリケーションの場合コンパイルフラグによって異なるが、
ここでは対応するビルド環境におけるデフォルトの設定でコンパイルした場合の結果を示す。
(2) システムコールの引数に依存する。ここでは移植性のある一般的な使用法 (引数) における結果を示す。

セキュリティ機能：DEP



- ハードウェア的にコード（実行可能）とデータ（実行不能）領域を区別し、攻撃者によって送り込まれたコードが実行されることを阻止する
- Linux カーネルの機能だが、互換性のために無効にされる場合がある
  - プログラムビルド時のオプションに -Wl,-z,noexecstack がない場合
  - このときプロセスの互換性フラグ（read-implies-exec）により、DEP が実質的に無効化されてしまう
- Android 2.2 以前のフレームワーク（指定されるビルド時のオプション）に互換性フラグの問題があったが、2.3 で解決

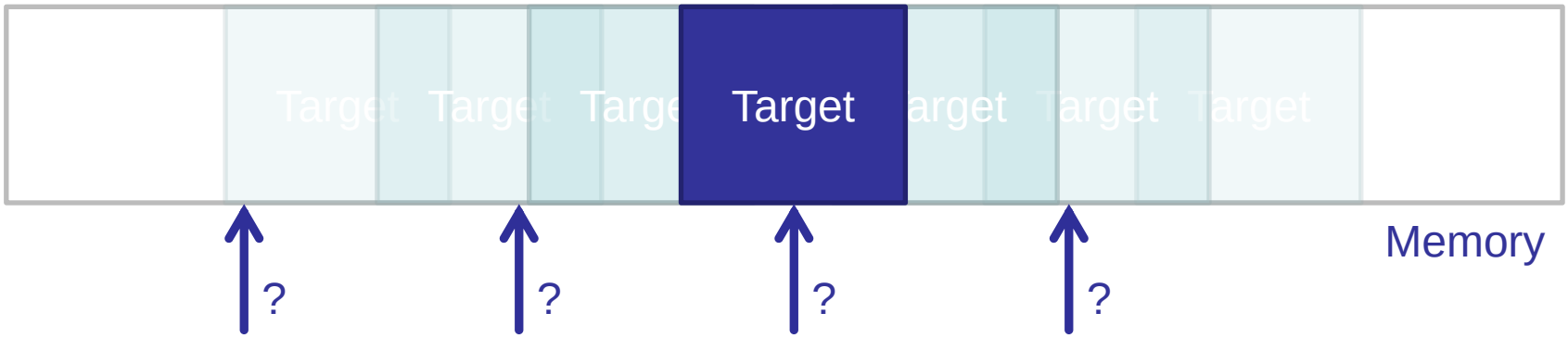
セキュリティ機能：W^X の強制



- 新たなコードを上書きできる、書き換え可能なコード領域を作成しない
  - コード領域が書き換え可能だと、そこを攻撃者に突かれてしまう
- 標準の Linux カーネルには、W^X を厳密に
  強制する機能が存在しない（PaX (1) パッチなどを必要とする）
  - Linux をベースとする Android も同様
  - Android には JIT があるが、その場合であっても
    W^X を強制することは十分可能だと考えられる（例: iOS の WebKit）

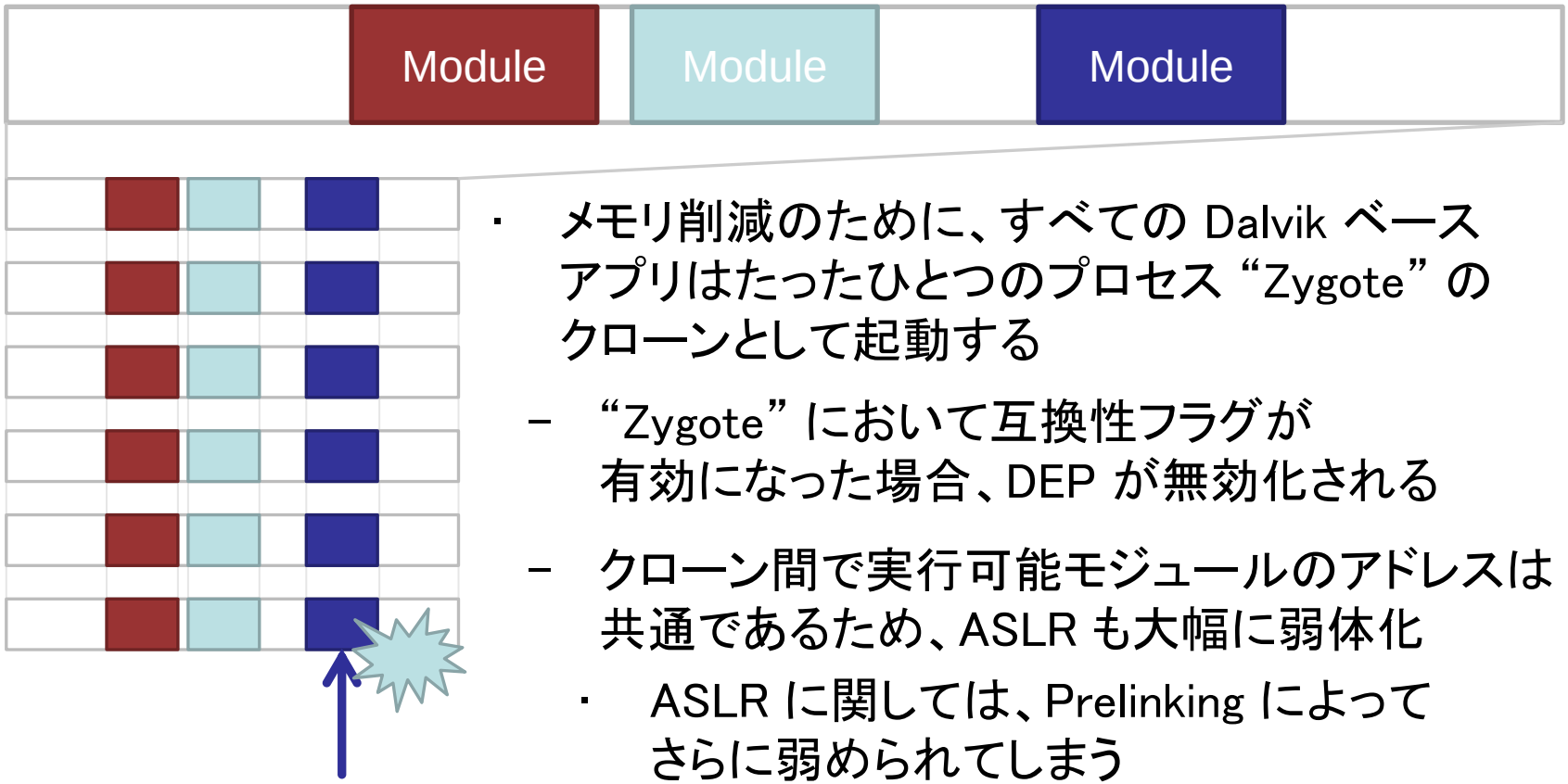
(1) http://pax.grsecurity.net/

セキュリティ機能：ASLR



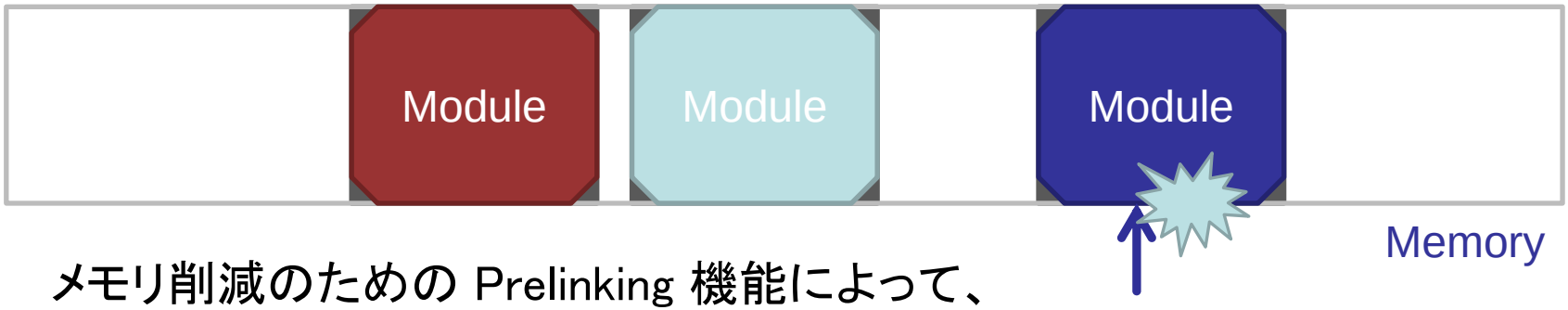
- メモリの配置をランダム化することで、
  特定のメモリアドレスを狙った攻撃を当たりにくくする
- Android で使用する Linux カーネルのデフォルト設定では、
  ヒープ以外のメモリをランダム化するようになっている
  - スタックはこれにより、比較的安全になる
  - しかし実際には、実行可能モジュールはカーネルの
    セキュリティ機能を無視してランダム化されない
    （後述の Prelinking による）

セキュリティ上の懸念：Zygote



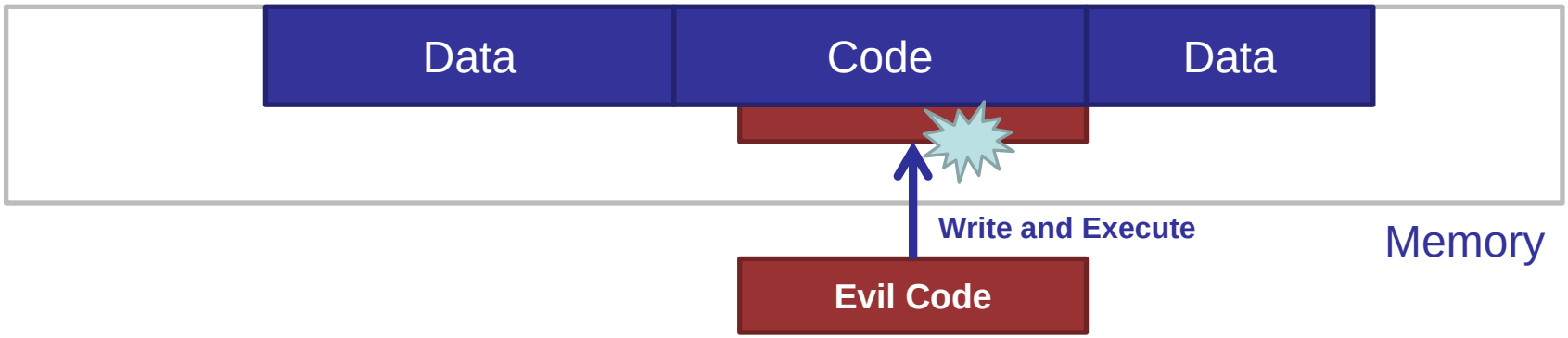
- メモリ削減のために、すべての Dalvik ベースアプリはたったひとつのプロセス“Zygote”のクローンとして起動する
  - “Zygote”において互換性フラグが有効になった場合、DEP が無効化される
  - クローン間で実行可能モジュールのアドレスは共通であるため、ASLR も大幅に弱体化
    - ASLR に関しては、Prelinking によってさらに弱められてしまう

セキュリティ上の懸念：Prelinking



- メモリ削減のための Prelinking 機能によって、
  本来ランダム化されるはずの実行可能モジュールのアドレスは
  フレームワークビルド時に設定されたものに固定されてしまう
  - 同じ機種、同じバージョンなら、必ず同じアドレスに配置される
  - ライブラリなどに対する ASLR の事実上の無効化
  - return-into-libc や ROP などの、洗練された、
    特定アドレスを狙う攻撃が極めて容易になってしまう
- メモリ削減のための Zygote と Prelinking は、
  本来 Linux カーネルが持つセキュリティ機能を半減してしまう。

セキュリティ不備の実例：V8 Engine



- ブラウザ (WebKit) に結びついた V8 Engine は、特定のメモリ領域を読み書きおよび実行可能 (rwx) で確保する (JavaScript 実行のため)
  - 実装において W^X の原則が徹底されていない
  - 悪意のあるコードを実行するのに利用されてしまう可能性がある
- ブラウザ (WebKit) を内部利用するすべてのアプリが同じリスクを背負うことになる

まとめ

- Linux カーネルが持つセキュリティ機能は、
  最新の Android においても十分に機能しているとはいえない
  - ネイティブコード部分が狙われた場合、
    高い確率で攻撃に成功してしまう危険性がある
- 不適切な実装やビルド設定は解決の余地がある
- Android 独自のメモリ削減の仕組みは
  メモリ保護の仕組みを大幅に弱体化する
  - モバイル CPU の性能向上によって改善していく余地がある

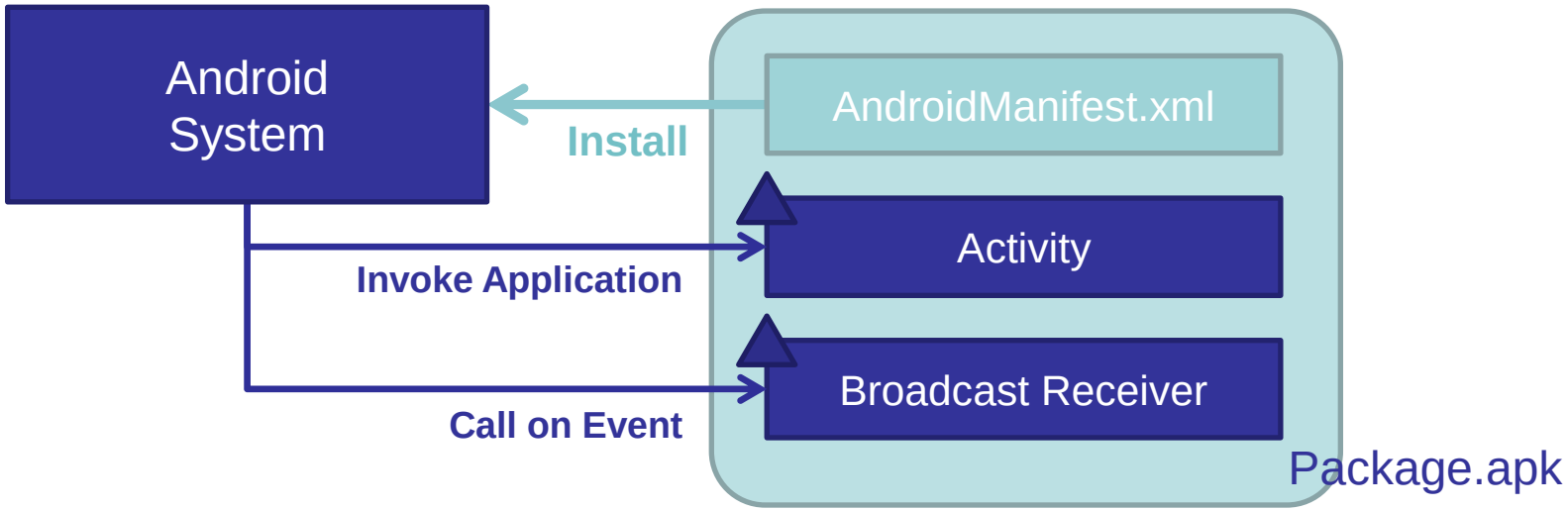
Android 独自の機構はどのように動作するか

アプリケーションレイヤーの独自の仕組み

Android アプリケーションの仕組みとは?

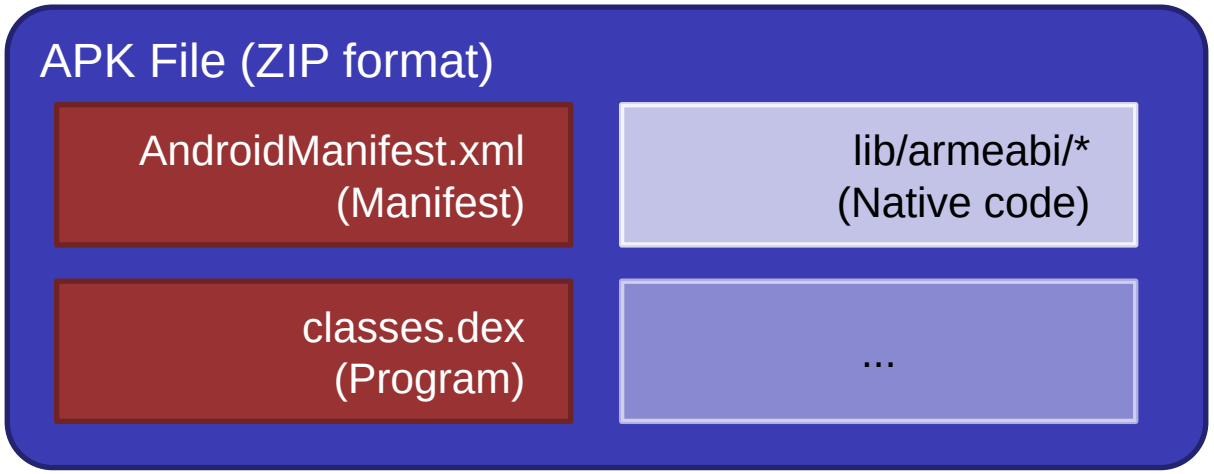
- 一般の OS におけるアプリケーションの位置づけとは全く異なる
  - インテントを利用したアプリケーション（アプリ）同士の連携が可能
- Android セキュリティを知るには、
  これら独自の仕組みをある程度理解しておく必要がある
  - パッケージとマニフェスト
  - パーミッション
  - インテント機構
    - アクティビティ（Activity）
    - ブロードキャスト（Broadcast）
    - …
- ここでは Android における機構の大まかな仕組みを解説する。

Android : アプリの仕組み



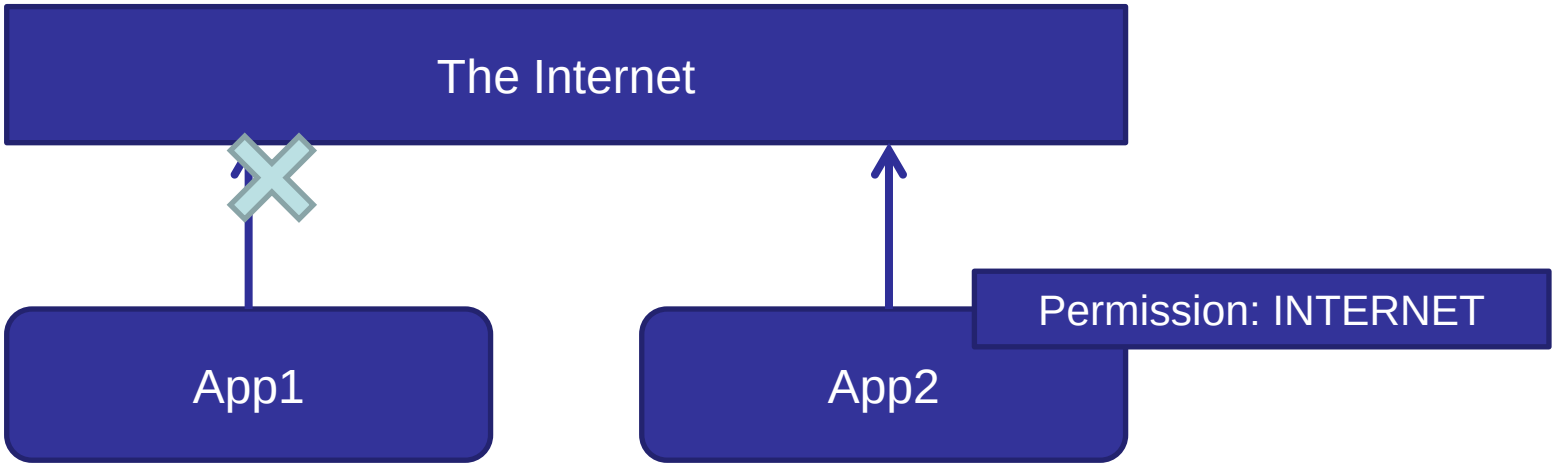
- アプリは “パッケージ” に格納される
- パッケージ内のクラスが “どのように呼び出されるべきか” を
  あらかじめ登録しておき、一定のルールでシステムから呼び出される
  - アクティビティ (Activity)
  - ブロードキャスト (Broadcast)
  - …

Android : パッケージ



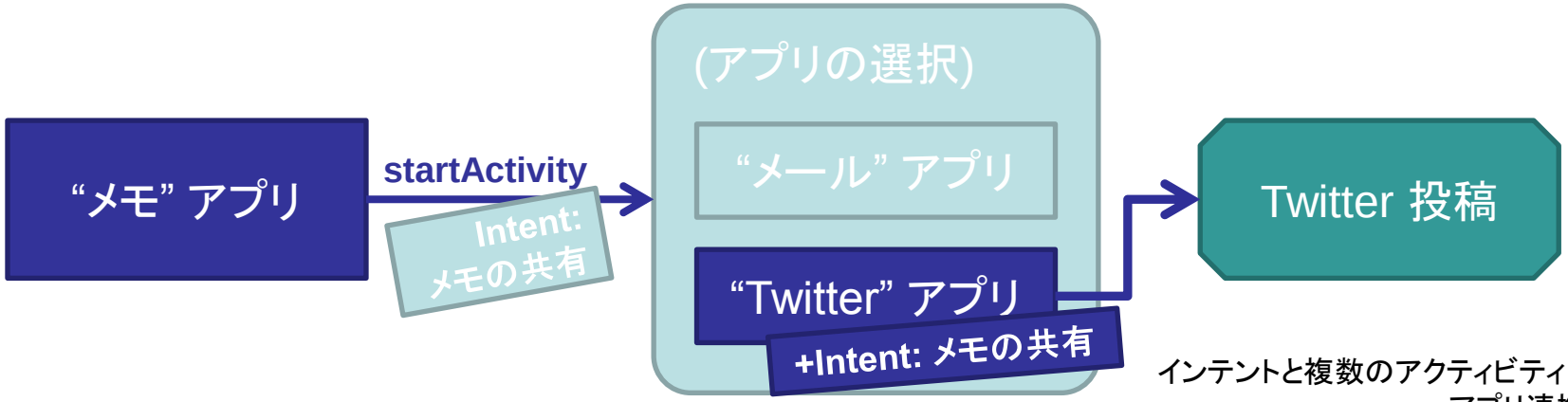
- パッケージ自体は単なる ZIP アーカイブ
- AndroidManifest.xml (マニフェスト)
  - Android アプリの情報やシステムからどのように呼び出されるか、必要とする権限などの情報を格納する
  - インストール後には変更できない
  - 検証が容易

Android : パッケージ/パーミッション



- アプリの動作に必要な「権限」をまとめたもの
  - INTERNET　　（インターネットへの接続）
  - READ_PHONE_STATE　（電話番号や状態を取得）
  - Android 2.3.3 の場合、合わせて 100 個以上が存在する
- 必要なパーミッションがない操作は拒否される
- パーミッションさえあれば、ほとんど何でもできる

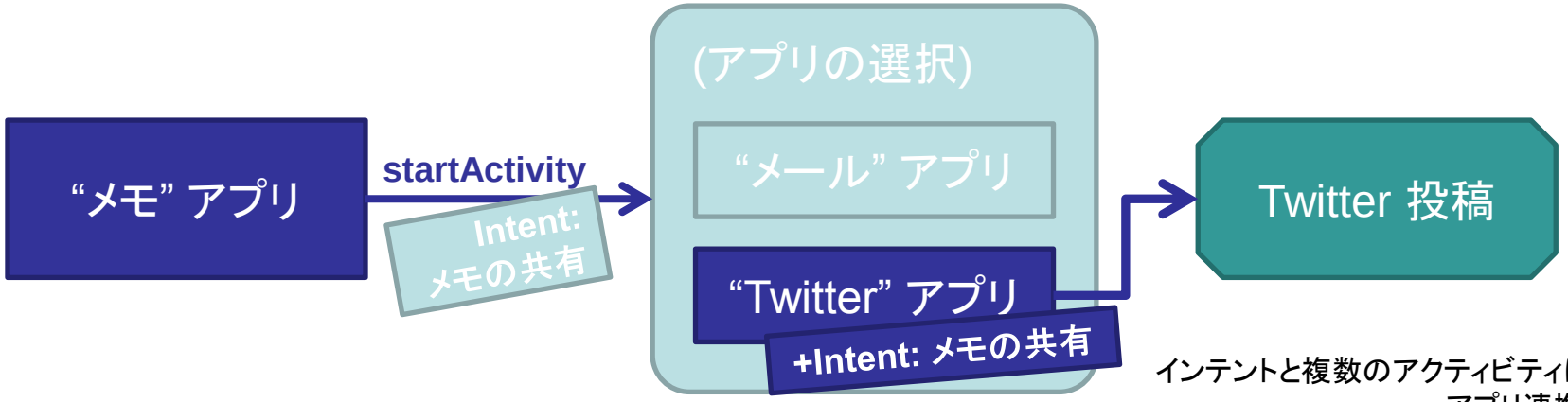
Android : インテント



インテントと複数のアクティビティによる
アプリ連携の例

- インテントと呼ばれる機構を用いてアプリ間での連携を行う
  - 操作内容、対象を含んだメッセージを送受信
- インテント機構は様々な用途に用いられ、極めて重要な位置を占める
  - アプリの相互呼び出し
  - システムイベントの通知、受信

Android : インテント（アクティビティ）



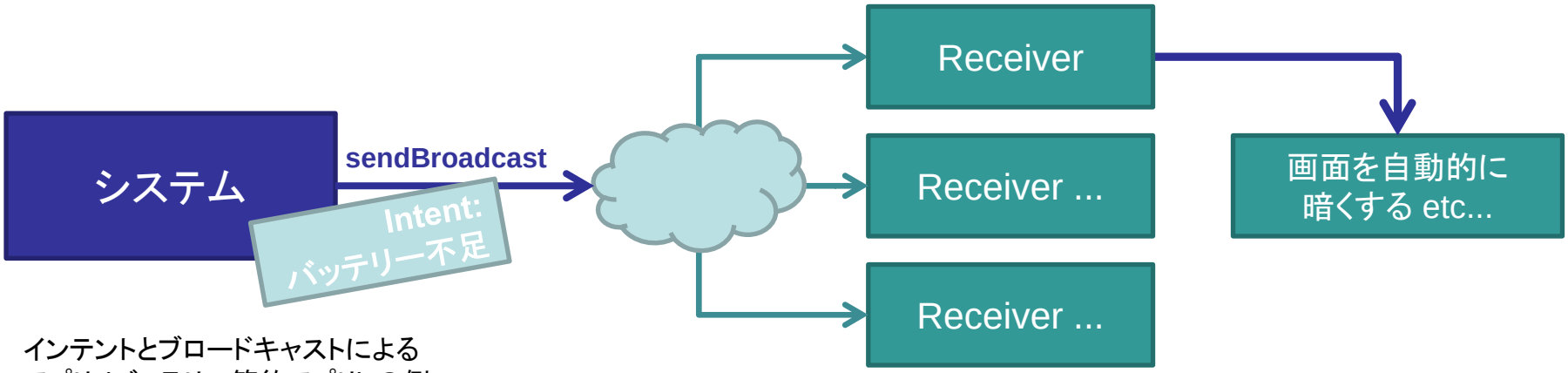
インテントと複数のアクティビティによる
アプリ連携の例

- アクティビティ（Activity）は小さなアプリケーションのように動く、
  ユーザーが “何かを行う” ひとかたまりの単位をモジュール化したもの
  - ここでは、“メモを作成する” “Twitter に投稿する” ための
    インタフェースがアクティビティに相当する
  - 行う操作と対象をマニフェストに記述しておけば、
    自動的にアプリ同士の連携を取ることができる
    （後述の インテント フィルタ による）

Android : 優先アプリケーション

- 対象となり得る複数のアクティビティがある場合、メニューから起動するアプリを選択する
  - “常にこの操作で使用する” をチェックすると、選択されたアクティビティが優先アプリケーションとして登録される
  - 以降、同じ状況においては優先アプリケーションが自動的に起動される

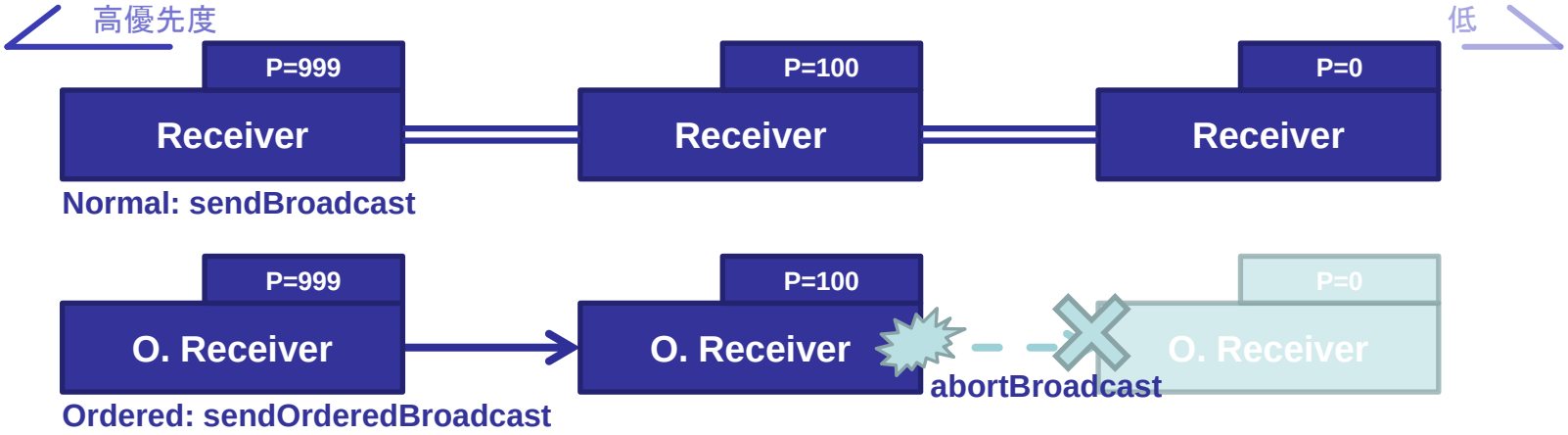
Android : インテント (ブロードキャスト)



インテントとブロードキャストによる
アプリ (バッテリー節約アプリ) の例

- ブロードキャスト (Broadcast) は、システムやアプリが
  発生させたイベントを受信するための仕組み
  - 登録された (原則として) すべての ブロードキャスト レシーバ
    (BroadcastReceiver) が起動され、そこでイベントが処理される

Android : 順序付きのブロードキャスト



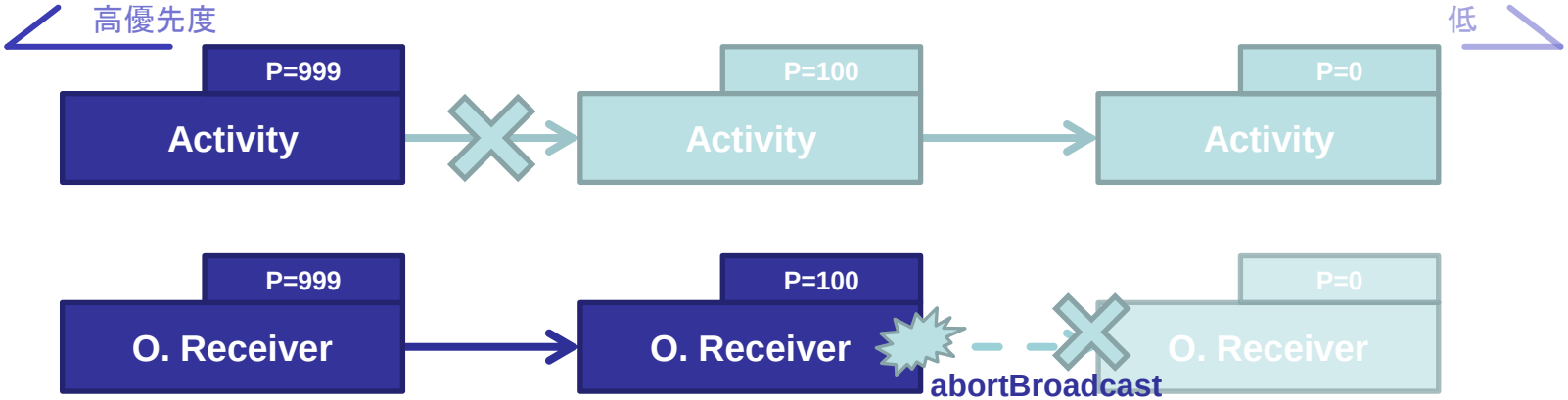
- ブロードキャストには、“順序なし” と “順序付き” のものがある
  - システム中では、一部のイベントだけが “順序付き”
- 順序付きのブロードキャストには、次の性質がある
  - 後述する優先度の順番に呼び出される（同じ優先度なら順不同）
  - abortBroadcast という操作で、ブロードキャストが
    優先度の低い他のレシーバに渡らないようにすることができる

Android : インテント フィルタ

例: テキストをアップロードするアプリ　例: 特定サイト閲覧専用アプリ　例: バッテリー節約アプリ

- Windows に例えると関連付けの仕組みに近いが、
  Android のインテント フィルタ (Intent Filter) はさらに柔軟である
  - アクション (何をする/受け取るか), カテゴリ (どう実行するか)
  - ファイルの種類 (MIME タイプ), 場所…
- AndroidManifest.xml に記述しておく
  - システムがすべてのインテント フィルタを管理し、
    必要なアクティビティやブロードキャストレシーバを起動する

Android : インテント フィルタの優先度



- インテント フィルタには、その優先度を設定できる
  - 高優先度のインテント フィルタに対応するアクティビティや ブロードキャスト レシーバが優先される
  - ブロードキャストの場合には、“順序付き” のものだけが 優先度順に呼び出される (そうでない場合には順不同)

まとめ：解説した項目

- Android アプリ
  - パッケージ / マニフェスト
  - パーミッション
- インテントを使う Android 独自の方式
  - アクティビティ (Activity)
  - ブロードキャスト (Broadcast)
    - 順序付きのもの
    - 順序付きでないもの
- インテントを補助するインテント フィルタ
  - その自由度と、優先度の仕組み

Android マルウェアや保護の現状と問題

脅威の現状、対抗策の限界

Android セキュリティと脅威の現状

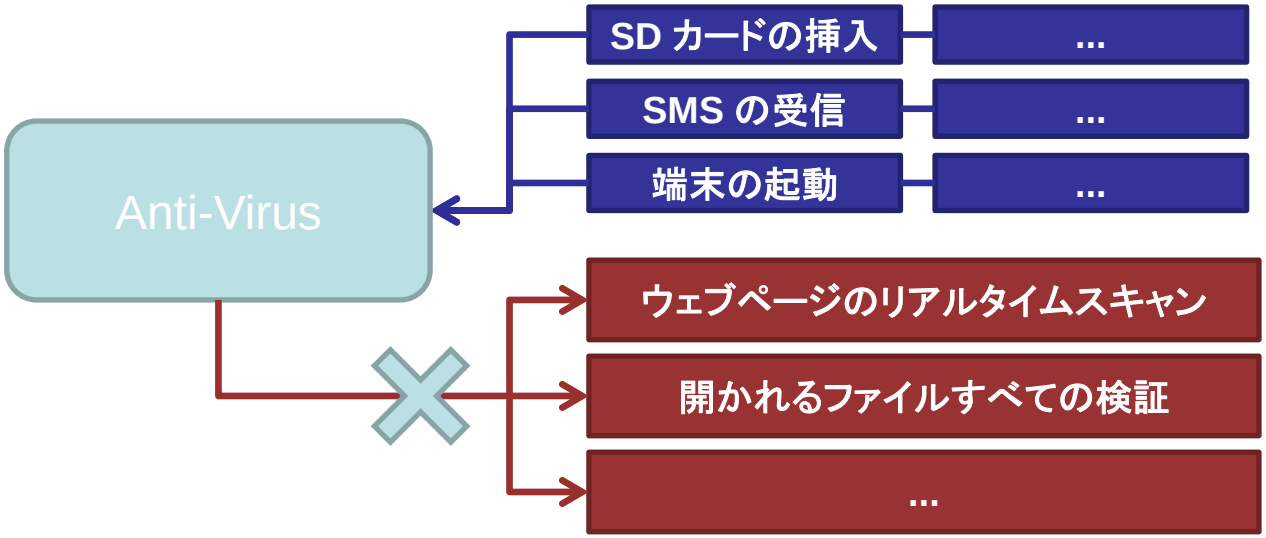
- 既に複数のマルウェアと、幾つかのウイルス対策ソフトが存在する
  - マルウェアの脅威はどのようなものか
  - ウイルス対策ソフトで守れるのか

- ウイルス対策ソフトとその動作
  - 権限不足という致命的な問題
- マルウェア
  - 現状のトレンドと具体的な動作
- root 化の問題
  - 破られる、Android のセキュリティ機能
  - 対抗策と、それでも解決しない問題

ウイルス対策ソフト：現状

- 現在の Android ウイルス対策ソフトは、大量のインテント フィルタを利用した部分的なリアルタイム保護を実現している
  - ダウンロードしたファイルやアプリのスキャン
  - SMS や E-メールのスキャン

ウイルス対策ソフト：現状と問題



- ウイルス対策ソフトは、一般権限で動作する
  - PC 用のものは、一般的に管理者権限を使用する
- 一般権限で動作する問題
  - 保護できない部分が多数存在する
    （ファイルアクセスのチェックや異常動作の検出など）
  - マルウェアを排除できない可能性がある（後述）

マルウェア：現状のトレンド

- 現状ほぼ全てが、次のどちらかに当てはまる
  - スパイウェア　　：重要な情報を搾取する
  - バックドア / ボット　：コマンドを受け取って悪意ある活動を行う、
    定期的に新たなマルウェアをダウンロードする…
- 中国国内を対象にした / 中国を強く示唆するマルウェアが多い
  - 中国の携帯ネットワークを示す APN や 電話番号
  - 盗み出した情報を SMS で送信する際に、
    中国語のメッセージを組み立てる
- 外部への / 外部からの情報のやりとり
  - インターネットを経由した HTTP 通信と
    SMS を使用したメッセージ通信が非常に多い

マルウェア：インテント フィルタの使用

- マルウェアも情報を盗み出したり、あるいはシステムに
  留まり続けるためにブロードキャストを使用する
  - 解析したマルウェア 9 種のうち一種 (FakePlayer) を除き、
    高い優先度をブロードキャストのインテント フィルタに設定している
- SMS の受信は、“順序付き” のブロードキャストである
  - abortBroadcast を使用すると、標準の SMS アプリでは
    受信したメッセージが見えなくなる
  - これを悪用し、一部のマルウェアは SMS として受信したコマンドが
    他のアプリからは見えないようにする

マルウェア：進化の現状

- 本格的な難読化が施されている Android マルウェアは
  現在のところ確認されていない
  - 現状のマルウェアはそれほど洗練されていない（比較的解析が容易）
- ただし、技術的には急激に進歩しつつある
  - DroidDream
    root 化を行った上で定期的にパッケージ（APK ファイル）を
    ダウンロード、自動でインストールする
  - Plankton
    外部から DEX ファイル（Dalvik バイトコード）をダウンロードし、
    クラスローダーという Java の仕組みを悪用して動的に実行する
- root 化を利用するマルウェアをはじめ、
  洗練されたマルウェアが問題になっていくと思われる

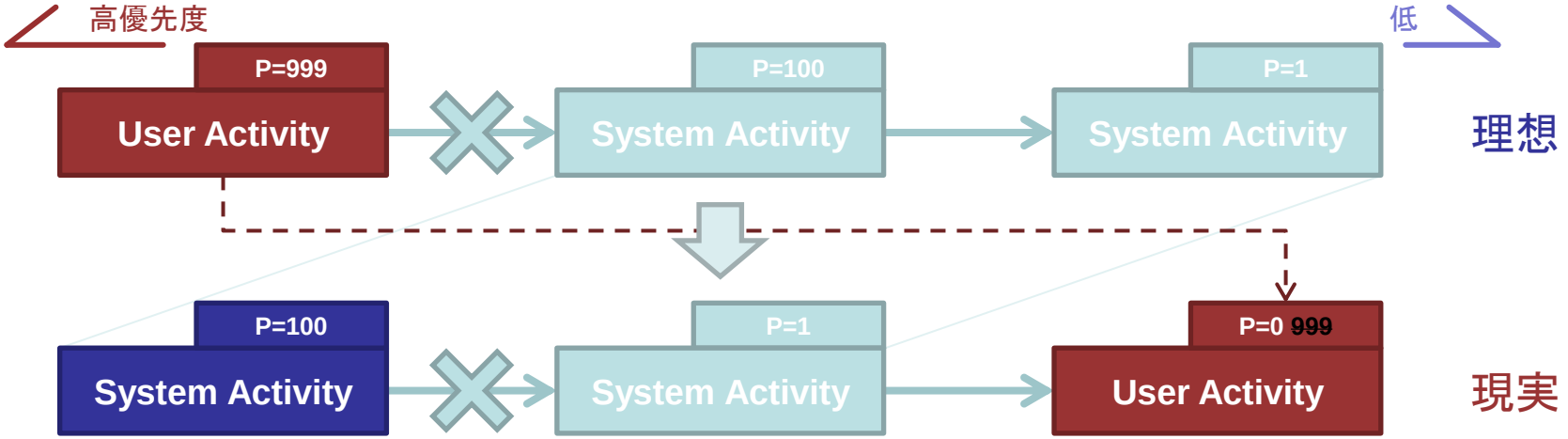
root 化

- ユーザー自身が Android 端末の制限を解除するために
  自ら脆弱性を突いて root（管理者）権限を取得することがある
- root 化によく使用される脆弱性
  - CVE-2009-1185 (exploid)
    - udev（デバイス管理）における不適切な netlink メッセージのチェック
  - CVE-2010-EASY (rage against the cage)
    - 正式な CVE 番号ではない（exploit 製作者による名称）
    - adb における、リソース不足時の不適切な setuid の使用
  - CVE-2011-1149 (psneuter)
    - ashmem（Android 共有メモリ）におけるアクセスチェックの不備
  - CVE-2011-1823 (Gingerbreak)
    - vold（ボリューム マネージャ）における不適切な netlink メッセージのチェック

root 化：問題

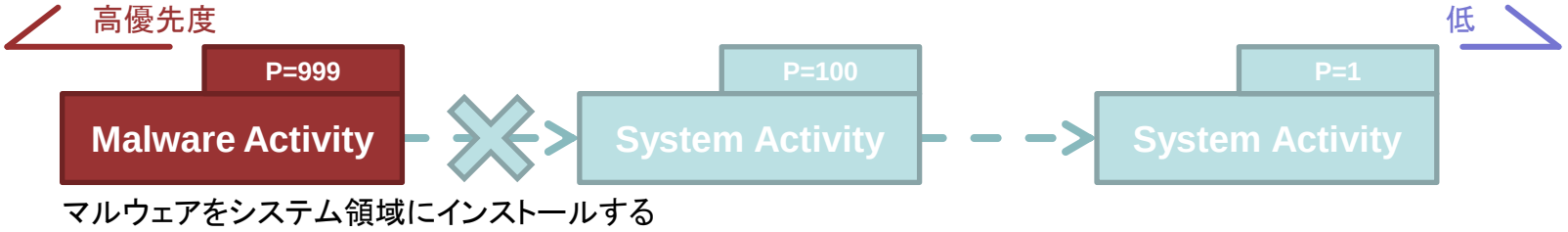
- マルウェアも全く同じ脆弱性を突くことができてしまう
  - マルウェアがシステム内の特権を取得することで、ウイルス対策ソフトが手出しできないようにしてしまう
- また root 化によって、Android が持つ保護の仕組みが破れてしまう
  - アクティビティに対応するインテント フィルタの優先度
  - パーミッションの仕組みの保護
- これらがマルウェアによって実行されると、ウイルス対策ソフトが対抗できなくなってしまう可能性がある

root 化による問題：優先度の保護 (1)



- 最も高い優先度を持つアクティビティが最優先で使用される
  - 例: ウェブサイトへのアクセスをフックする
  - 低い優先度を持つアクティビティが呼び出されない (メニューなし)
- マルウェアなどによる乗っ取りを防ぐための優先度保護の仕組み
  - システム領域 “以外” にインストールされたアプリは
    デフォルト (0) より上の優先度を持つことができない

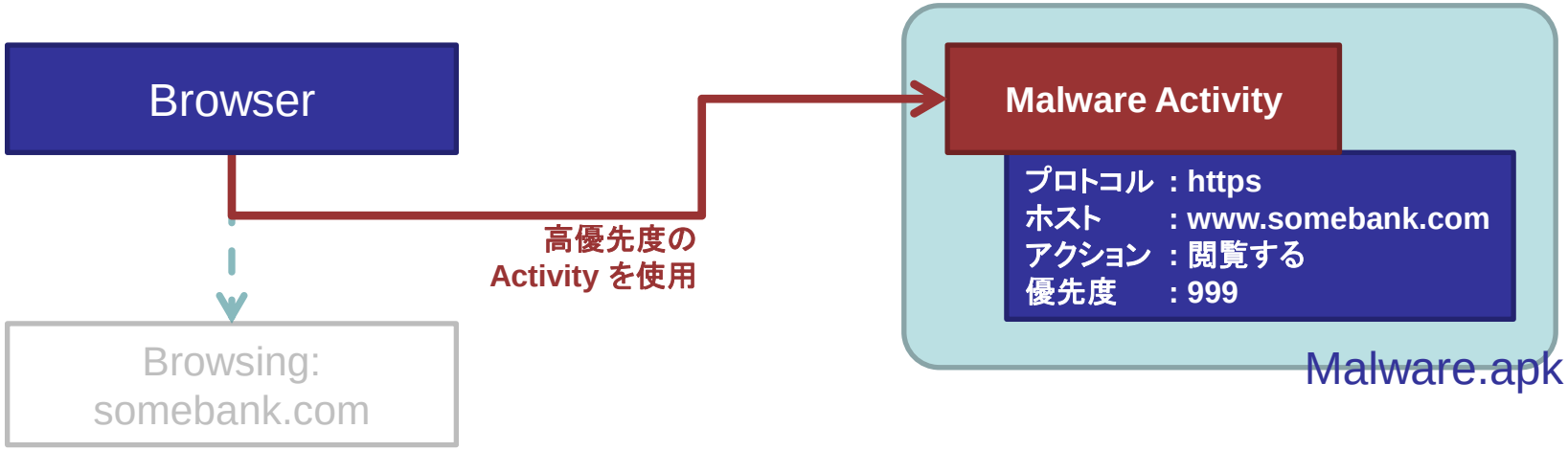
root 化による問題：優先度の保護 (2)



- root 化によって、マルウェアをシステム領域※ に
  インストールすることが可能になってしまう
  - システム領域にインストールされたアプリは Android システムの
    一部として信頼され、優先度の保護が無効になる
  - 特定のファイルを開くことを防止したり、
    ある種のフィッシングを実行できるようになる
- 条件：root 化の上でシステム領域に書き込みやマウントができること

※ カスタマイズされていない Android においては、/system/app, /system/framework, /vendor/app の 3 箇所

root 化の悪用：フィッシング

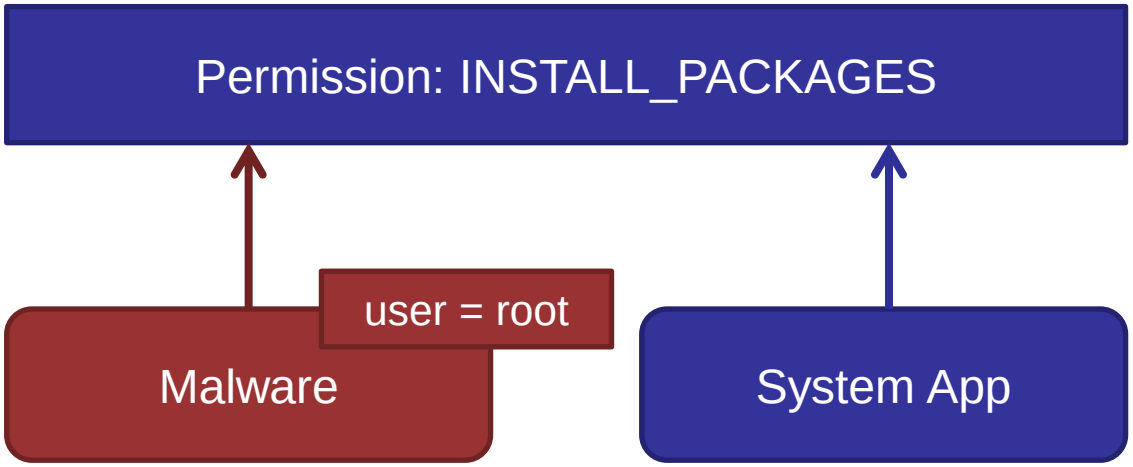


- ウェブサイトへのアクセスをマルウェアにリダイレクトすることができる
  - ユーザーがほとんど気づけない
  - アクティビティの UI をブラウザと同じにすれば、
    ほとんど本物のウェブサイト、ブラウザと区別できなくなる
- ウイルス対策ソフトはこの手法への対抗策を持たない

root 化による問題：パーミッションの保護 (1)

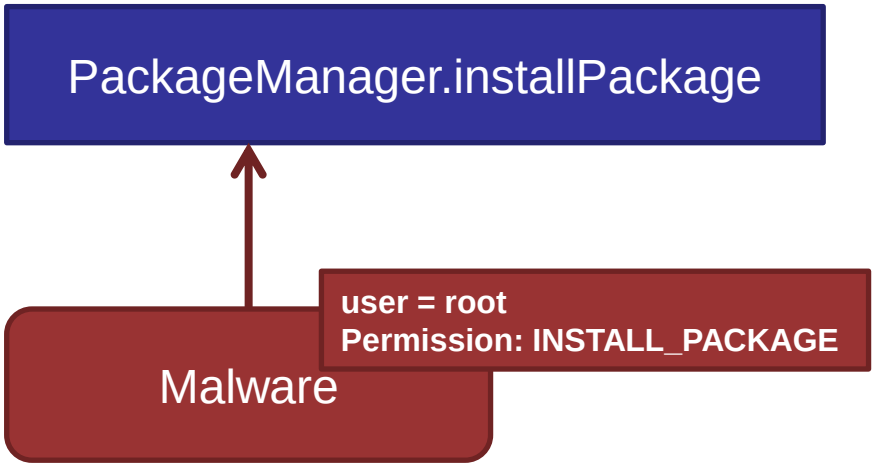
- 一部のパーミッションは、一般権限のアプリからは要求できない
  - 例えば上記のパッケージを強制インストールするための権限
  - 悪用された場合の危険性が大きいため、
    一部のパーミッションはシステムアプリからのみ使用できる
  - マニフェスト (AndroidManifest.xml) にパーミッションを
    定義しても、インストール時に拒否される

root 化による問題：パーミッションの保護 (2)



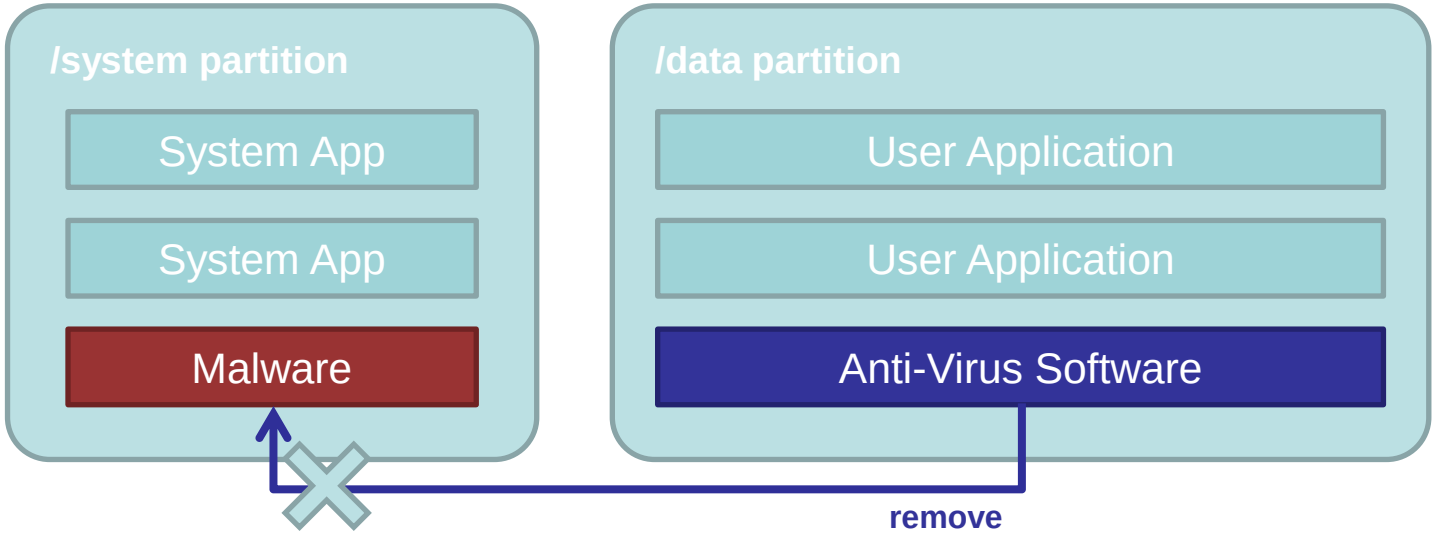
- root 化されたアプリに対しては、パーミッションの仕組みが働かない
  - 保護されたパーミッションであっても、その権限を得ることができる
- これによってできることは…
  - パッケージの強制インストール, アンインストール
  - 優先アプリケーションの再設定など
- 条件：root 化のみ

root 化の悪用：サイレントインストール



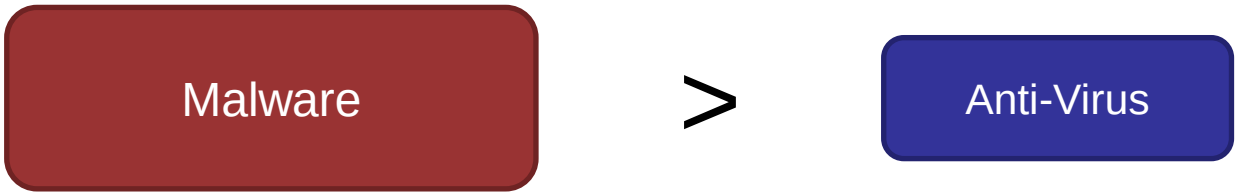
- INSTALL_PACKAGE パーミッションをマルウェアが獲得した場合、ユーザーの許可なく任意のパッケージがインストールされてしまう
  - 通常はインストールする前に警告を受けるが、この権限を持つ場合には一切警告なしにインストールできる
- root 権限をパッケージファイルに対して即座に与えることで、本来ウイルス対策ソフトが行える対策を妨害することができる

root 化の悪用：システム領域の改変



- root 権限を悪用して、本来読み取り専用の
  システムパーティションにマルウェアをインストールする
- ウイルス対策ソフトはマルウェアを削除できない
  - 読み取り専用パーティションにある上に、Android システムからは
    システムアプリのひとつとして認識されてしまう

root 化の悪用：ウイルス対策ソフトの排除



- マルウェアは root 化さえしてしまえば、ウイルス対策ソフトよりもはるかに強力な権限を行使することができる
  - ウイルス対策ソフトが自ら root 化の脆弱性を利用することには倫理上大きな問題がある
  - ウイルス対策ソフトからはマルウェアを攻撃できない
- 逆にマルウェアはウイルス対策ソフトを攻撃することができる
  - 強制的なアンインストール
  - プロセスの停止
  - プロセスやパッケージの改変による無力化など

root 化の悪用：対策とその限界 (1)

- 脆弱性を積極的に発見、除去する
  - 当たり前に見えるが、しかし脆弱性が発見されて
    10 ヶ月以上経過してもそれが修正されていない機種が存在する
    (http://www.ipa.go.jp/about/technicalwatch/pdf/110622report.pdf)
- root ユーザーを含めて制限するセキュリティ機構
  - SELinux, AppArmor, TOMOYO Linux...
  - SHARP 製 Android 端末 : Deckard / Miyabi LSM
    - システムパーティションが不正にマウントされることを阻止
    - ptrace (強制的なデバッグ) の禁止
    - chroot (仮想ファイルシステムの作成) の禁止など
  - root 権限を確保した攻撃者が必要以上に
    権限を拡大することを防止できる

root 化の悪用：対策とその限界 (2)

- root ユーザーの制限だけでは完全な対策にならない
  - パーミッションによる保護は root に対しては無効化されている
  - Dalvik ベースのアプリはすべて単一プロセスのクローンであるため、既存のセキュア OS においてポリシーで保護することが難しい
  - ウイルス対策ソフトの権限は弱い一般権限のまま
- Android 特有の事情を考慮した上で権限の保護を行うことが必要
- 正当なウイルス対策ソフトの権限を、少なくとも root 化したマルウェアと同等まで引き上げられることが望ましい

まとめ

- Android 向けのマルウェアもウイルス対策ソフトも、
  初期と比べると飛躍的に進化している
  - しかし現状ウイルス対策ソフトは低い権限で動作しており、
    限定的な保護しかできていない
- root 化は Android のセキュリティ機構を破り、
  ウイルス対策ソフトが手出しできないマルウェアを可能にしてしまう
  - 仮にウイルス対策ソフトがマルウェアを発見できても、
    それを除去、無害化することができない可能性がある
  - 対処には権限面での見直しや、Android 特有の事情を考慮した
    新しい保護の仕組みが必要とされる

Android は “守れる” か?

総括

Android は守られているか? (1)

- 脆弱性攻撃
  - Android は WebKit など多数のネイティブライブラリに依存しており、ネイティブコード部分は従来と同様に攻撃可能である
  - 本来 Android が基盤とする Linux カーネルの保護機能は、様々な理由によって効果が半減している
    - フレームワークビルド時のミス (DEP)
    - メモリ削減のための機能 (Zygote, Prelinking)
  - よって脆弱性が存在した場合、それを攻撃するのは現状それほど難しくない

Android は守られているか? (2)

- マルウェア / ウイルス対策ソフト
  - マルウェアは現状トロイの木馬としてインストールされ、
    インテント フィルタなどの仕組みを用いることで情報を盗んだり、
    あるいは悪意あるコマンドを受け取る
  - root 化によってシステムの保護をバイパスし、通常出来ない
    悪意ある行動を起こすことが可能になる (サイレントインストールなど)
  - ウイルス対策ソフトは一般ユーザーの権限で動作するため、
    保護できない部分があるばかりか、root 化されたマルウェアに対して
    手を出すことができない

- 現状、Android 端末を十分に守りきることはかなり難しい

これからのためにすべきこと (1)

- 技術的責任 : Android プロジェクト
  - セキュリティ機構を厳密なものに変える
    (root を特別扱いしないようにする)
    - システムコールレベル (LSM)
    - Android フレームワークのセキュア化
  - 正当なウイルス対策ソフトの保護を支援する
    - 例: 特別な署名が成されたアプリに一定の特権を与える
  - メモリ保護機能を強化する
    - Zygote や Prelinking の見直しやセキュリティ強化

これからのためにすべきこと (2)

- 技術的責任：端末メーカー
  - 既存の脆弱性を素早く修正し、アップデートを行う
    (端末ユーザーが攻撃されることをいち早く防ぐ)
  - フレームワークやネイティブプログラムをビルドする際の
    コンパイルフラグに注意を払う (-Wl,-z,noexecstack)
  - 独自のカスタマイズを十分に検証する
    - Android のセキュリティ機構が破れないよう
      (あるいは正当なプログラムの実行を阻害されないよう)
      十分に注意して実装を行う

結論

- 現状 Android の保護は十分でなく、また脅威も氾濫しているが、解決が不可能な問題ではない
  - ただし、利用者は当面注意を必要とする
- Android プロジェクト、端末メーカー、セキュリティベンダーなどが協力して Android のセキュリティを高めることに注力すべき

ありがとうございました

Fourteenforty Research Institute, Inc.
株式会社 フォティーンフォティ技術研究所
http://www.fourteenforty.jp

新技術開発室　大居　司
<oi@fourteenforty.jp>