Static Analysis of Mobile Apps for Security and Privacy

Manuel Egele
megele@cmu.edu
Carnegie Mellon University

Mobile Devices are Ubiquitous

400 million iOS devices in total
(June 2012)

400 million Android devices in total
(June 2012)

Today, about 1 *billion* smart devices!

Mobile Apps – A Success Story

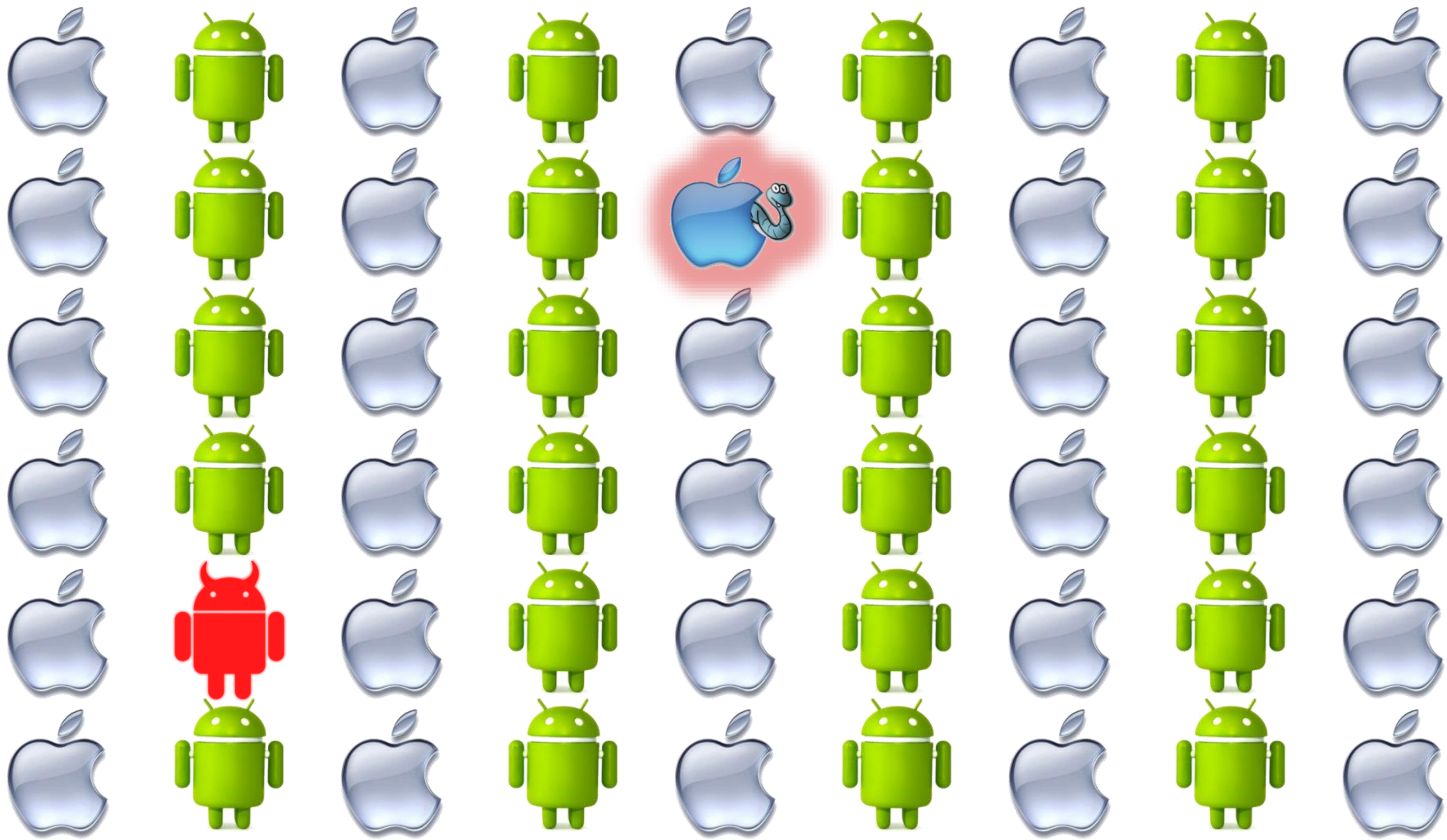
Apple App Store

- 775,000 apps
- 40 billion downloads
- $5 billion to developers

Google Play

- 490,000 apps
- ~ $247 million / year

Are All Apps Good?



Detecting Bad Apps

Bad apps available on App Stores

- Find & Call—leak address book from iOS and Android, contacts receive spam SMS
- Path—circumvent denied location access
- MogoRoad—leaked phone numbers lead to marketing calls

My system identified more than 200 bad apps

My Vision:
Automatically assess the security of mobile applications.

Security Properties

1. **Define a security property**
   - Privacy of sensitive data
   - Integrity of control-flow
   - Correct application of crypto primitives
2. **Build system to evaluate security property**
   - PiOS (Privacy)
   - MoCFI (Control-flow integrity)
   - Cryptolint (Crypto primitives)
3. **Evaluate the property on real-world data**
   - 1,407 iOS apps
   - 16,943 Android apps

Challenge – Software

- UI driven and interactive
- Complex runtime environments
  - Objective-C runtime
  - Android framework
- Apps mix type-safe and unsafe code

Novel analysis techniques necessary

Overview

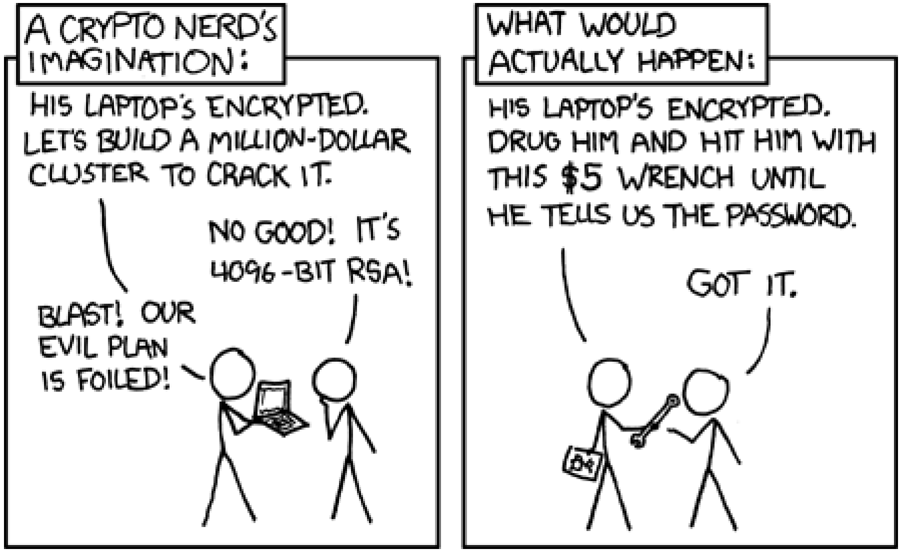
- Mobile security challenges
- Analysis of mobile apps
  - Statically detect privacy leaks
  - Retrofit apps with CFI
  - Misused crypto

Mobile Applications on iOS [NDSS'11]

- Third party developers build applications
- Binaries vetted by Apple during application review process
- Users expect sensitive data to be protected from misbehaving 3rd party applications

Research Goals

1. Analyze if user's expectation of privacy holds

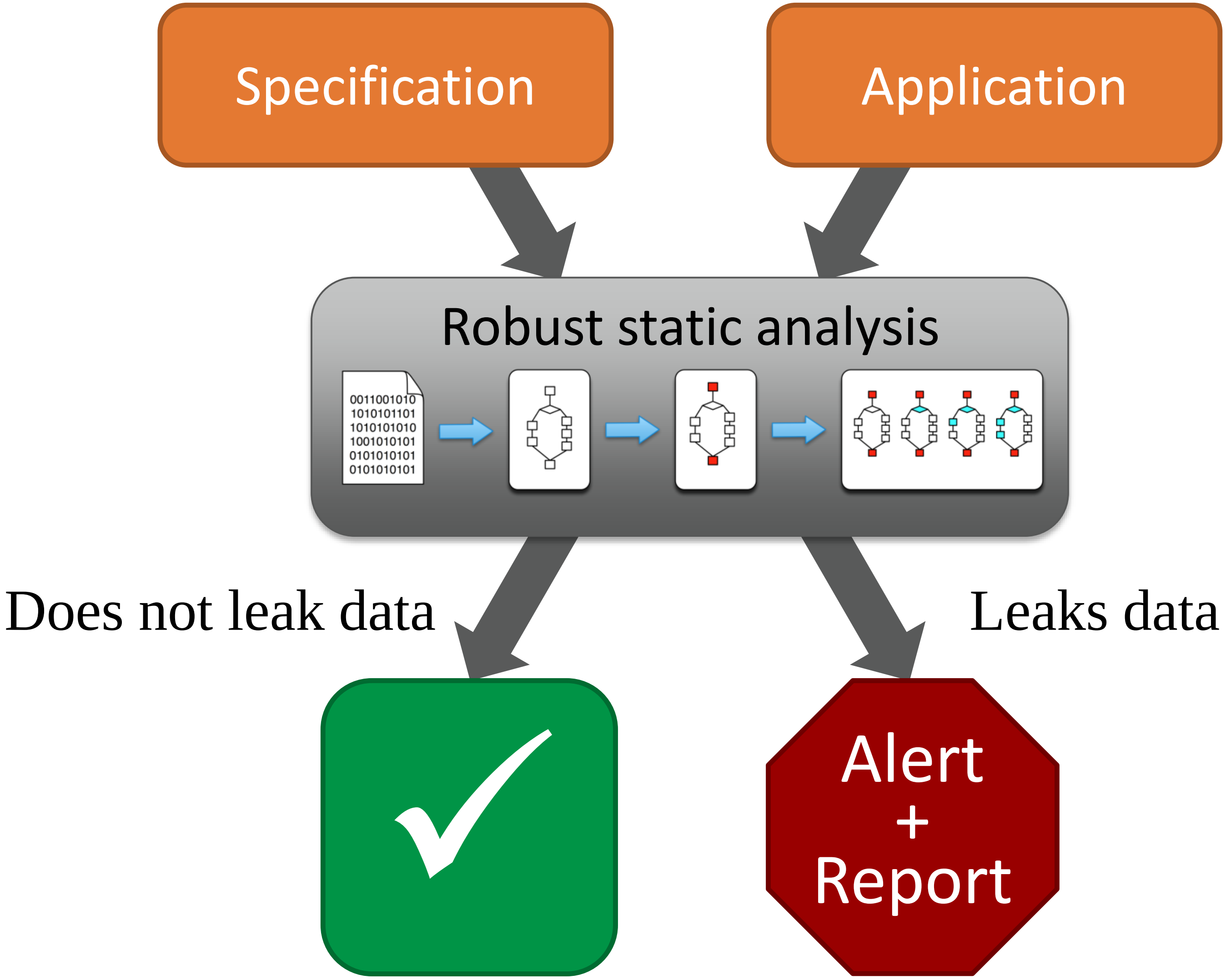


2. Perform analysis on a large number of apps

Plan of Action

1. Security property: "Apps should not *access privacy sensitive information* and *transmit this information over the Internet* without user intervention or consent"
2. System to evaluate the property – PiOS
3. Evaluate on 1,407 real-world apps

PiOS – System Overview

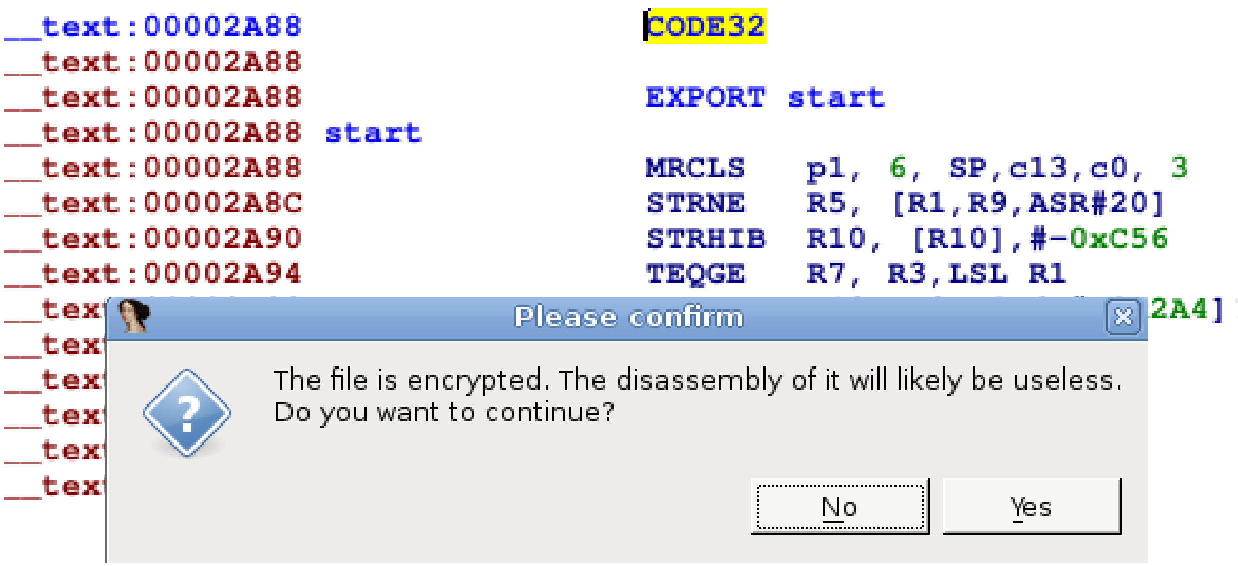


How To …

detect apps that *access privacy sensitive information* and *transmit this information over the Internet* without user intervention or consent?

1. Identify whether app accesses privacy sensitive information – *a source* (e.g., address book API)
2. Identify whether app communicates with the Internet – *a sink* (e.g., networking API)
3. Analyze whether data accessed in 1. is transmitted in 2.

Static Analysis of iOS Apps



Background – iOS & DRM

- Apps are encrypted and signed by Apple
  - Individual key for each user
- iOS loader verifies signature and performs decryption in memory
- Decrypt App Store apps
  - Attach with debugger while app is running
  - Dump decrypted memory regions
  - Reassemble binary

Background – Static Analysis

- Reason about program without executing it
- Terminology and concepts:
  - Basic Block
  - Control Flow Graph (CFG)
  - Call Graph (CG)
  - Super Control Flow Graph (sCFG)

Basic Block

A maximal sequence of instructions that always execute in the same order together.

```
1. x = y + z
2. z = t + i
```

```
3. x = y + z
4. z = t + i
5. jmp 1
```

```
6. jmp 3
```

3 basic blocks

Control Flow Graph (CFG)

A static ***Control Flow Graph*** is a graph where

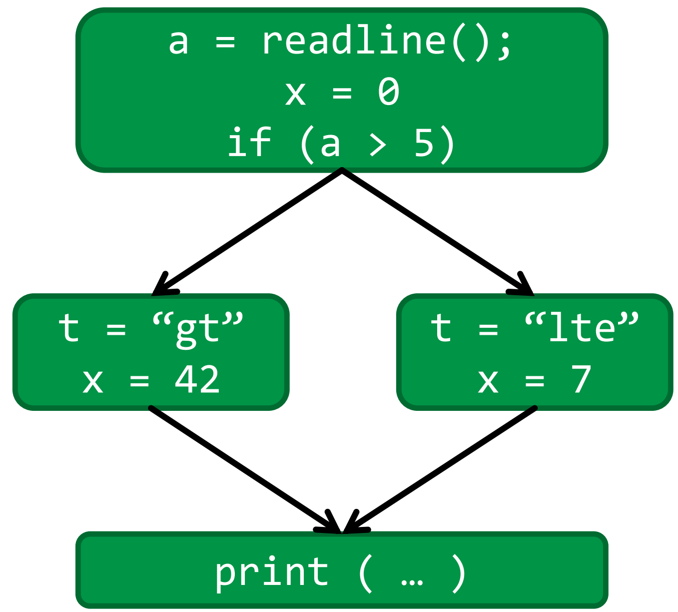
- each vertex v_i is a basic block, and
- there is an edge (v_i, v_j) if there ***may*** be a transfer of control from block v_i to block v_j.

Historically, the scope of a CFG is limited to a function or procedure, i.e., intra-procedural.

CFG – Example

- each vertex v_i is a basic block, and
- there is an edge (v_i, v_j) if there ***may*** be a transfer of control from block v_i to block v_j.

```
a = readline()
x = 0
if (a > 5) {
   t = “gt”
   x = 42
} else {
   t = “lte”
   x = 7
}
print(“input was “ +
   t + “ 5”)
```



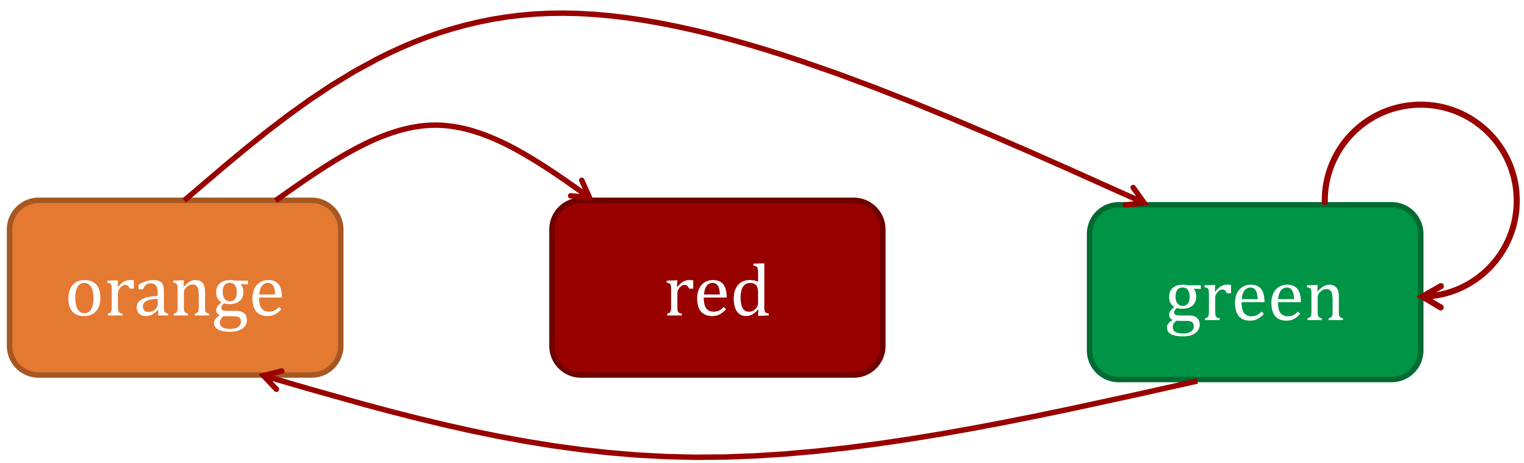
Call Graph

Nodes are functions. There is an edge (v_i, v_j) if function v_i calls function v_j.

```
void orange()
{
  green();
  red();
}
```

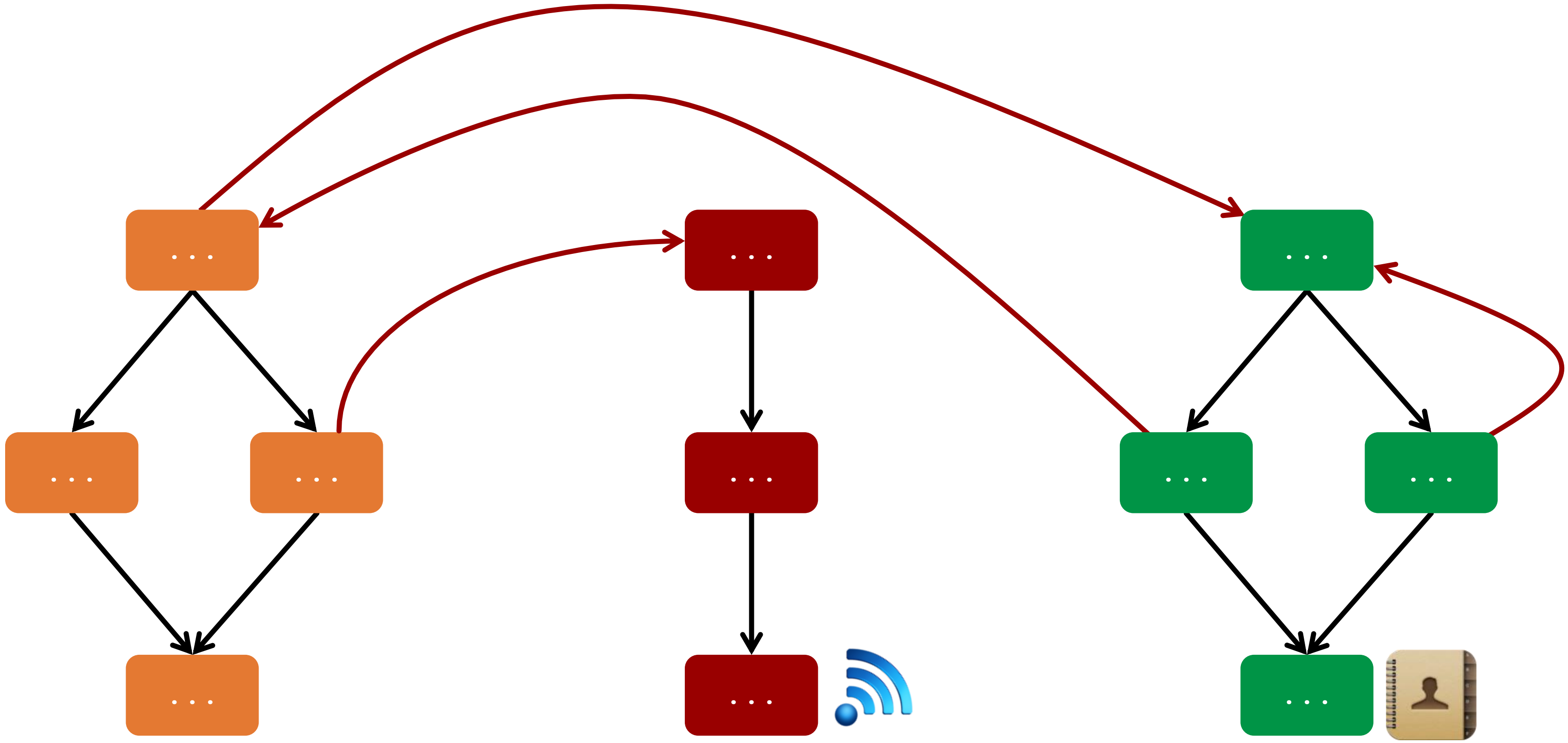
```
void red()
{
...
}
```

```
void green()
{
  green();
  orange();
}
```

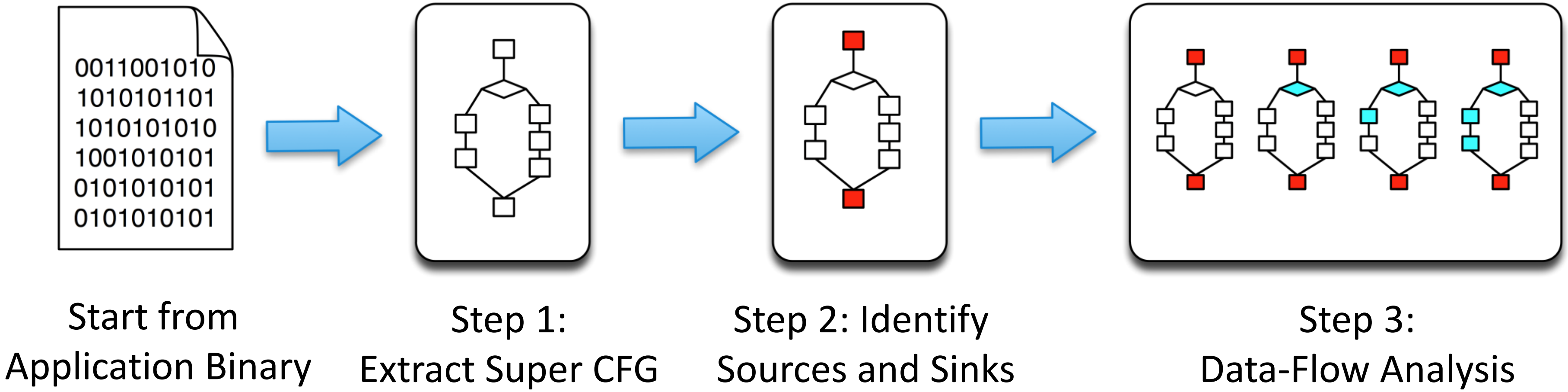


Super Control Flow Graph

Superimpose CFGs of all procedures over the call graph



PiOS – Static Analysis

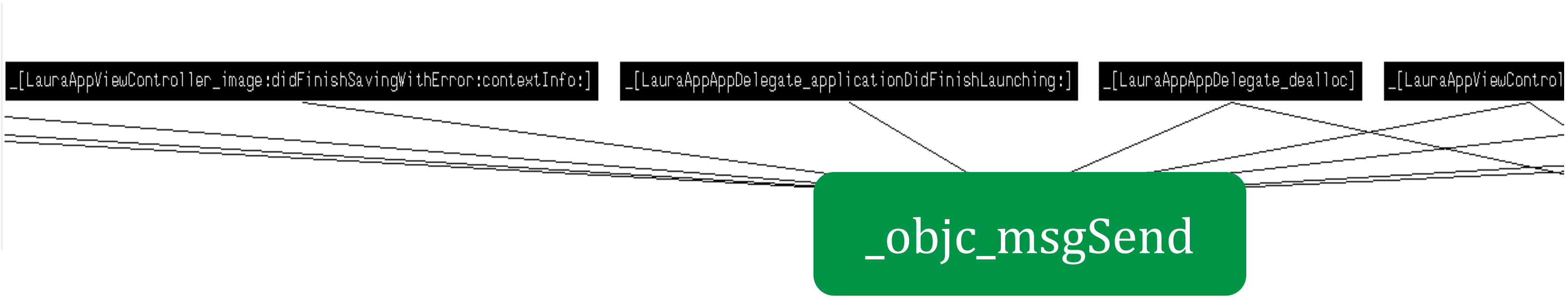


1. Extract super control flow graph from binary application
2. Identify *sources* of sensitive information and network communication *sinks*
3. Data flow analysis between sources & sinks

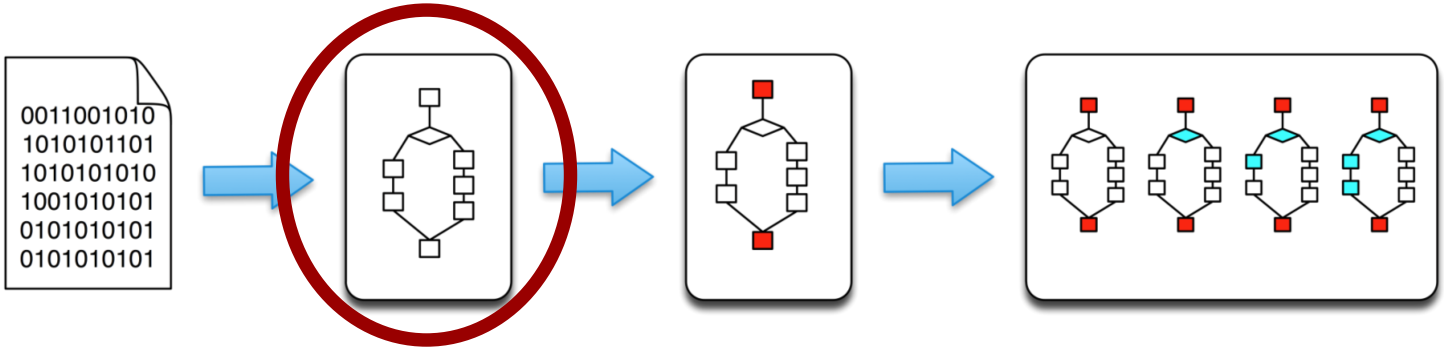
Running Example (Tank Wars)

Static Analysis of iOS Apps

IDA Pro: Call-graph for “Tank Wars”



Extract Super CFG



PiOS – Analysis

- Most iOS apps are written in Objective-C
- Cornerstone: objc_msgSend dispatch function
- Task: Resolve type of receiver and value of selector for objc_msgSend calls
  - Backwards slicing
  - Forward propagation of constants and types

`objc_msgSend`
Dynamic Dispatch Function

Arguments

- Receiver (Object)
- Selector (Name of method, string)
- Arguments (vararg)

Method look-up at runtime

- Traverses class hierarchy
- Calls method denoted by selector
- Information available at runtime, challenging to extract statically

Similar to reflection in Java

- Objective-C only uses reflection

PiOS – Analysis (Super CFG)

Novel analysis approach for object-oriented binaries written in Objective-C based on two key techniques:

1) Resolve *type* of receiver and *value* of selector for objc_msgSend calls
   a) Backwards slicing [Weiser ‘81]
   b) Forward propagation of constants and types
2) Multiple candidate types for receiver
   ⇒ class hierarchy

objc_msgSend Example

```
1 LDR R0, =off_24C58 → UIDevice
2 LDR R1, =off_247F4 → currentDevice
3 LDR R0, [R0]
4 LDR R1, [R1]
5 BLX _objc_msgSend
...
1
```

Type of R0: Value of R1

Q: What method is invoked here?

NSString:initWithFormat

(fmt: “uniqueid=%@&scores=%d”)

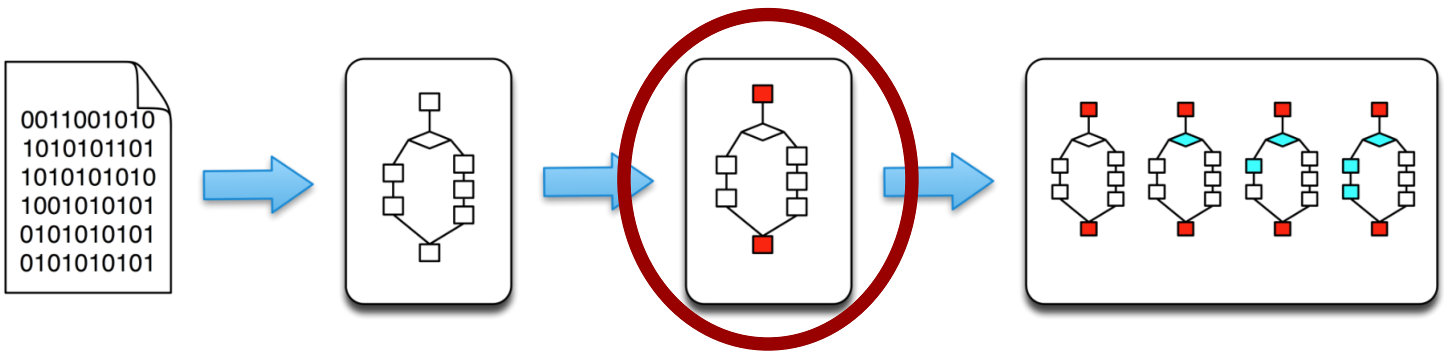
PiOS – Analysis (Super CFG)

Novel analysis approach for object-oriented binaries written in Objective-C based on two key techniques:

1) Resolve *type* of receiver and *value* of selector for objc_msgSend calls
   a) Backwards slicing [Weiser '81]
   b) Forward propagation of constants and types
2) Multiple candidate types for receiver
   ⇒ class hierarchy

Result: Super-CFG constructed from successfully resolved calls to objc_msgSend

Identify Sources and Sinks

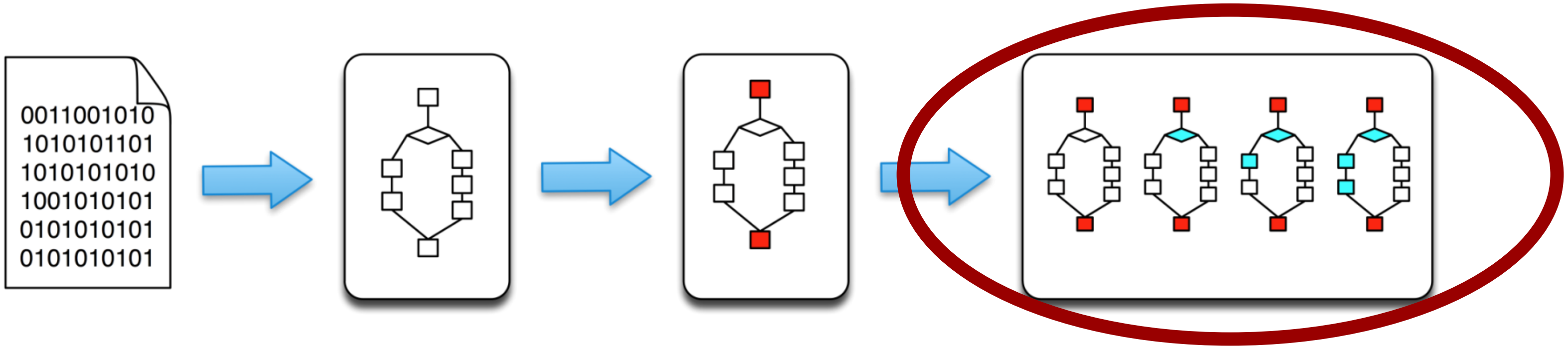


PiOS – Finding Privacy Leaks

- Based on super-CFG
- Reachability Analysis (find paths)
  - From interesting *sources*
  - To network *sinks*

- Sources and sinks identified by API calls

Dataflow Analysis



Data-Flow to Model Security Properties

- Tracks *how* information is propagated through an application or system
- Data-flow captures confidentiality problems well (e.g., how is sensitive information used)

Now we can detect apps that *access privacy sensitive information* and *transmit this information over the Internet* without user intervention or consent.

PiOS – Evaluation

- 1,407 Applications
  (825 from App Store, 582 from Cydia)
- Pervasive ad and app-telemetry libraries
  - 772 apps (55%) contain at least one such library
  - Leak UDIDs, GPS coordinates, etc.
- Apple requires that libraries are statically linked

Advertisement Libraries

- 82% of apps that use Ads use AdMob (Google)
- Send UDID and AppID on start and ad-request
- Ad company can build detailed usage profiles
- Problem: Location-aware apps
  - Access to GPS is granted per app/binary
  - Libraries linked into location-aware apps have access to GPS

PiOS – Evaluation: Leaked Data

Source	#App Store 825	#Cydia 582	Total 1407
DeviceID	170 (21%)	25(4%)	195(14%)
Location	35(4%)	1(0.2%)	36(3%)
Address book	4(0.5%)	1(0.2%)	5(0.4%)
Phone number	1(0.1%)	0(0%)	1(0.1%)
Safari history	0(0%)	1(0.2%)	1(0.1%)
Photos	0(0%)	1(0.2%)	1(0.1%)

PiOS – Evaluation: Case Studies

From: iTunes Store <msmac@apple.com>
To: iphonedev@iseclab.org
Subject: Re: App Store; Follow-up: 104124416
Date: Sun, 9 May 2010 16:00:45 -0700 (PDT) (*05/09/2010 07:00:45 PM*)

Dear Manuel,

Thank you for your patience while this matter was under review.

I understand you have privacy concerns with the "Gowalla" application.

For information on Apple's Privacy Policy, you can review the Terms of Sale for the App Store:

I located a section for you that applies to your concern:

"Apple is not responsible for Third Party Products, the content therein, or any warranties or claims that you or any other party may have relating to that Third Party Product or your use of the Third Party Product."

You may also consider contacting the developer regarding your concerns about this application.

I was able to locate this email address for you, from their website (http://gowalla.com/) in which you may contact the developer about their Privacy Policy:

live@gowalla.com

Impact in Popular Media

Overview

- Mobile security challenges
- Analysis of mobile apps
  - Statically detect privacy leaks
  - Retrofit apps with CFI
  - Misused crypto

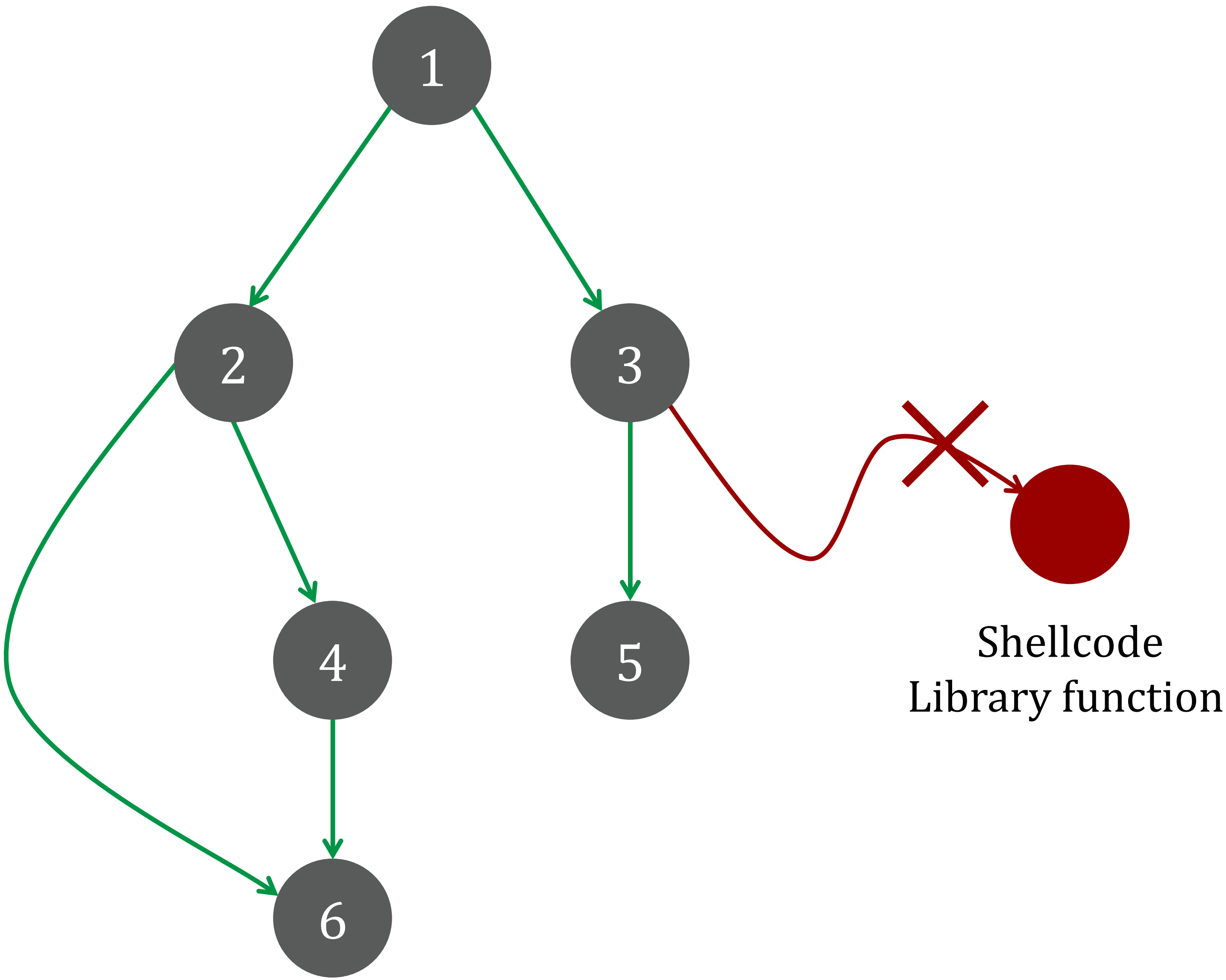
Attacks on Mobile Software

- Developers make mistakes (bugs)
- A bug becomes a security vulnerability if it can be exploited through an attack
- Attackers can compromise a device through such attacks

Control Flow Attacks

- Many attacks rely on hijacking of control flow
  - Buffer overflows
  - Function pointer overwrites
- iOS has powerful defenses
  - W $\oplus$ X
  - Stack canaries
  - Mandatory code signing
  - ASLR
- Attacks leverage return-oriented-programming
  - pwn2own contest

Control Flow Integrity [Abadi'05]



MoCFI – Static Analysis [NDSS'12]

- sCFG recovery using PiOS
- Identify branch instructions
- Identify instructions implementing "return"
  - `ldr PC,[R12]`
  - `pop {R4-R7,PC}`
- Bundle meta information with the app

MoCFI – Dynamic Enforcement

- Enforcement code in dynamic library
- Library parses the metadata and modifies application in memory
  - Rewrite control-flow instructions to enforce CFI (i.e., only perform the original control-flow instruction if validation succeeds)

Attackers can no longer hijack control flow

Overview

- Mobile security challenges
- Analysis of mobile apps
  - Statically detect privacy leaks
  - Retrofit apps with CFI
  - Misused crypto

Security Properties

Approach is to evaluate security properties

- Privacy of sensitive data
- Integrity of control-flow
- What about programming errors?

Do developers apply crypto correctly?

Detecting Crypto Misuse

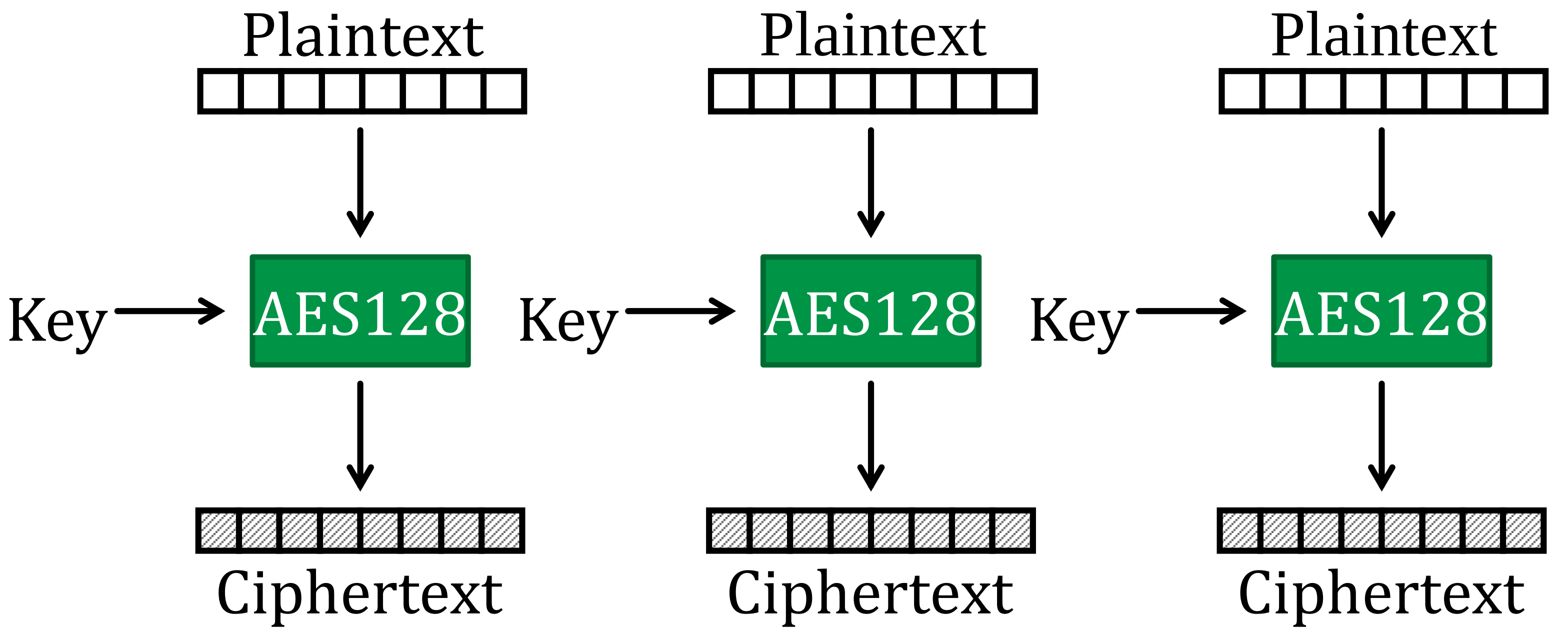
- App developers handle sensitive data
- They realize encryption is good
- App developers are no security experts

Plaintext

Block Cipher Modes (ECB)

Blockcipher

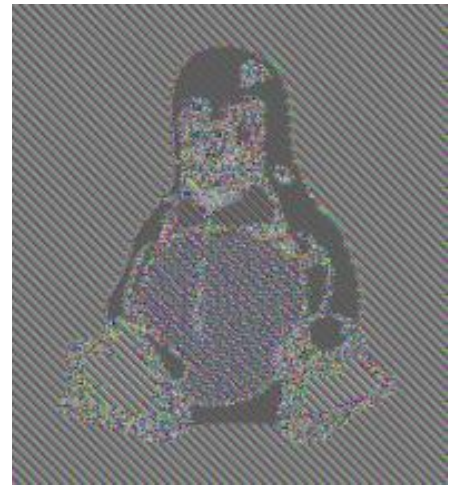
Encrypt one block of n-bit length plaintext into one block of n-bits of cipher text (For AES128, n = 128)



Electronic Code Book (ECB) Mode

Block Cipher Modes (ECB)

Plaintext

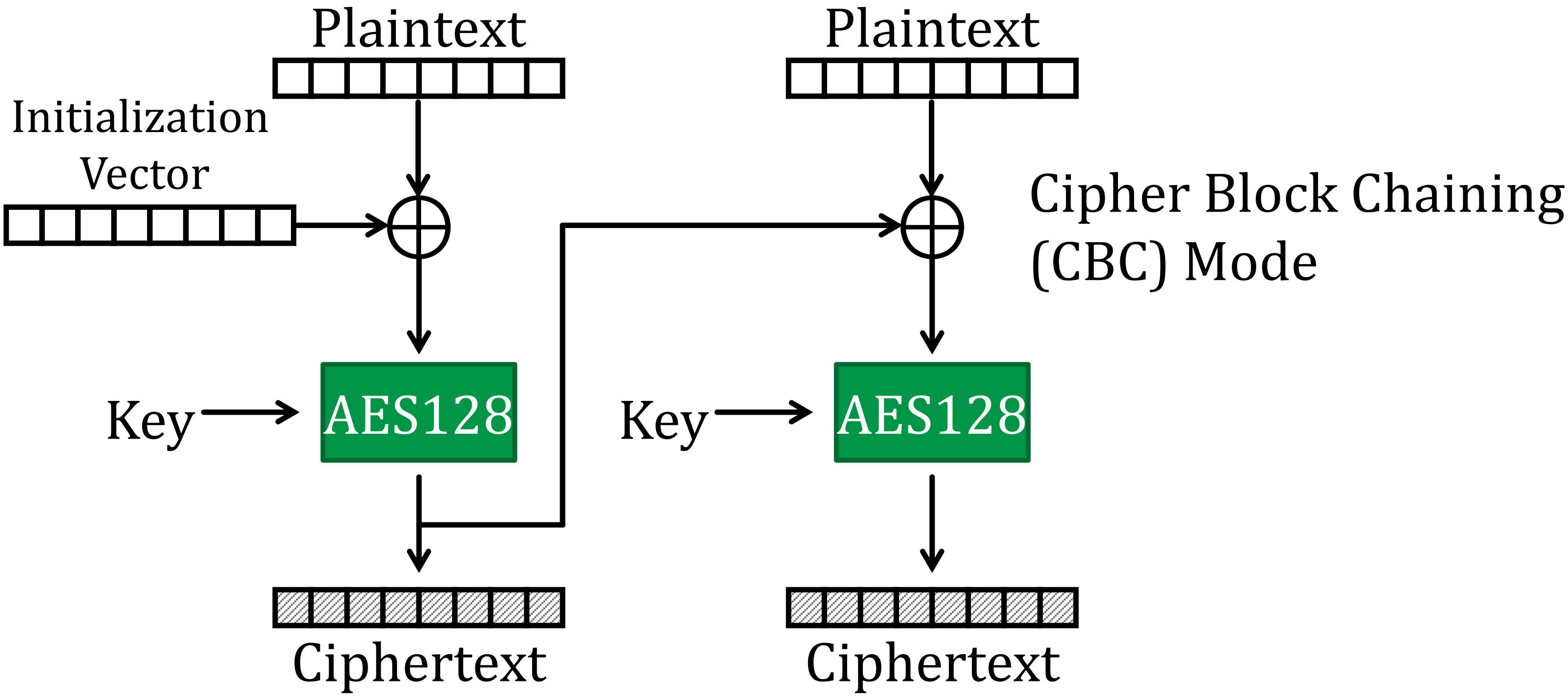


AES128/ECB

Block Cipher Modes (CBC)

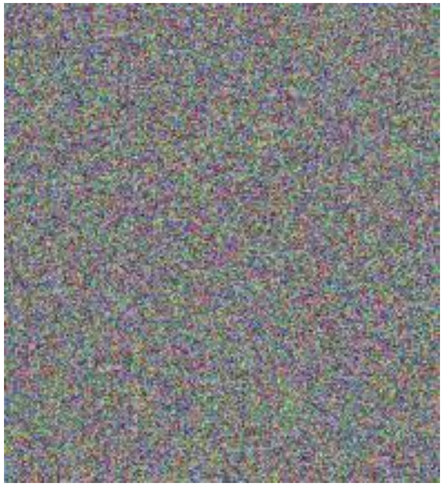
Blockcipher

Encrypt one block of n-bit length plaintext into one block of n-bits of cipher text (For AES128, n = 128)



Block Cipher Modes (CBC)

Plaintext



AES128/CBC

Crypto APIs in Android

Cryptographic service providers (CSP) are interfaces to:

- (A-) symmetric crypto
- MAC algorithms
- Key generation
- TLS, OpenPGP, etc.

Android uses BouncyCastle as CSP

BouncyCastle is compatible to Java Sun JCP

Commonly Used Crypto Primitives

Symmetric encryption schemes IND-CPA

Block ciphers: AES/[3]DES

Encryption modes: ECB/CBC/CTR

Password-based encryption Cracking resistance

Deriving key material from user passwords

Pseudo random number generators Secure seed

Random seed

Common Rules

1) Do not use ECB mode for encryption
2) Do not use a static IV for CBC mode
3) Do not use constant symmetric encryption keys
4) Do not use constant salts for PBE
5) Do not use fewer than 1,000 iterations for PBE
6) Do not use static seeds to seed `SecureRandom()`

Cryptolint

Static program analysis techniques

1. Extract a super control flow graph from app
2. Identify calls to cryptographic APIs
3. Static backward slicing to evaluate security rules

Automatically detect if developers do not use crypto correctly!

Static Program Slicing [Weiser '81]

Slicing criterion:

Program point **p** and a variable **x**

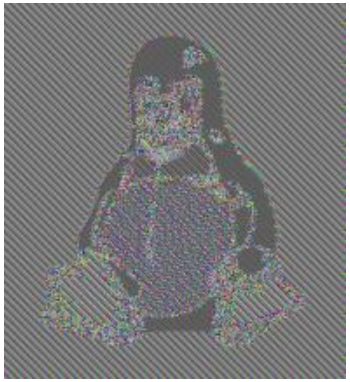
Slice:

All program instructions that might affect the value of **x** at point **p**

Rule 1: Thou Shalt Not Use ECB

Transformation string specifies:

- Algorithm
- Block Cipher Mode (optional)
- Padding (optional)



```
Cipher.getInstance("AES/ECB/PKCS7Padding", "BC");
```

Default for block ciphers: ECB (undocumented)

Problem: Bad defaults

Rule 2: Thou Shall Use Random IVs

CBC$ algorithm specifies random IV

```
c = Cipher.getInstance("AES/CBC/PKCS7Padding");
c.getIV();
```

Developer can specify IV herself

```
public final void init (int opmode, Key key,
   AlgorithmParameterSpec params)
IvParameterSpec(byte[] iv)
```

Problem: Insufficient Documentation

Rule 3: Thou Shalt Not Use Static Symmetric Encryption Keys

Key embedded in application ⇒ not secret

Symmetric encryption schemes often specify a randomized key generation function

To instantiate a key object:

```
SecretKeySpec(byte[] key, String algorithm)
```

Problem: Developer Understanding

Rule 4: Thou Shalt Not Use Constant Salts for Password Based Encryption

RFC2898 (PKCS#5):

"4.1 Salt … producing a large set of keys … one is selected at random according to the salt."

```
PBEParameterSpec(byte[] salt,
   int iterationCount)
```

Problem: Poor Documentation

Rule 5: Thou Shalt Not Use Small Iteration Counts for PBE

RFC2898 (PKCS#5):

"4.2 Iteration Count: For the methods in this document, a minimum of 1,000 iterations is recommended."

```
PBEParameterSpec(byte[] salt,
  int iterationCount)
```

Problem: Poor Documentation

Rule 6: Thou Shalt not Seed `SecureRandom()` With Static Values

Android documentation for `SecureRandom()` PRNG:

"This class generates cryptographically secure pseudo-random numbers. It is best to invoke `SecureRandom` using the default constructor. "

...

"Seeding `SecureRandom` may be insecure"

`SecureRandom()` vs. `SecureRandom(byte[] seed)`

Problem: Developer Understanding

Evaluation

- 145,095 Apps downloaded from Google Play
- Only Apps that use
  - `javax/crypto`
  - `java/security`
  - Filter popular libraries (advertising, statistics, etc.)
- 11,748 Apps analyzed

Evaluation

11,748 apps use crypto

65% use ECB

13% use static salt for passwords

31% use static symmetric key

13% use small iteration counts

16% use known IV for CBC

14% misuse `SecureRandom()`

88% have *major* crypto problem

Password Manager (2010)

```
private String encrypt(byte [] key, String clear) {
  byte [] encrypted;
  byte [] salt = new byte[2];
  ...
  Random rnd = new Random();
  //Cipher cipher = Cipher.getInstance("AES");
  Cipher cipher =
    Cipher.getInstance("AES/ECB/PKCS7Padding", "BC");
  cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
  rnd.nextBytes(salt);
  cipher.update(salt);
  encrypted = cipher.doFinal(clear.getBytes());
```

Password Manager (+6 days)

```
private String encrypt(byte [] key, String clear) {
  byte [] encrypted;
  byte [] salt = new byte[2];
  ...
  Random rnd = new Random();
  Cipher cipher =
    Cipher.getInstance("AES/CBC/PKCS7Padding", "BC");
  byte [] iv = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
  IvParameterSpec ivSpec = new IvParameterSpec(iv);
  cipher.init(Cipher.ENCRYPT_MODE, skeySpec, ivSpec);
  rnd.nextBytes(salt);
  cipher.update(salt);
  encrypted = cipher.doFinal(clear.getBytes());
```

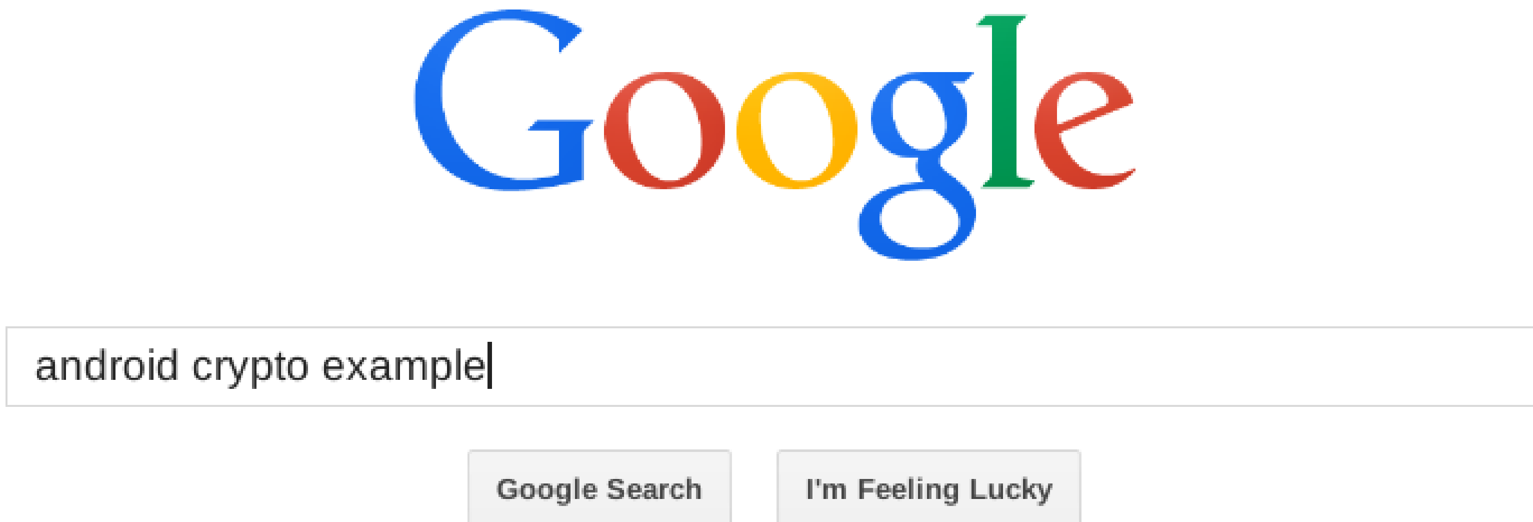
Password Manager (+2yrs, 5mo)

```
private String encrypt(byte [] key, String clear) {
  ...
  Random rnd = new Random();
  Cipher cipher =
    Cipher.getInstance("AES/CBC/PKCS7Padding", "BC");
    byte [] iv = new byte[16];
    rnd.nextBytes(iv);
    IvParameterSpec ivSpec = new IvParameterSpec(iv);
    cipher.init(Cipher.ENCRYPT_MODE,skeySpec,ivSpec);
    encrypted = cipher.doFinal(clear.getBytes());
    ...
```

Password Manager (key)

```
public static byte []
hmacFromPassword(String password) {
  byte [] key = null;
  ...
  Mac hmac = Mac.getInstance("HmacSHA256");
  hmac.init (new SecretKeySpec
    ("notverysecretiv".getBytes("UTF-8"), "RAW"));
  hmac.update(password.getBytes("UTF-8"));
  key = hmac.doFinal();
  ...
  return key;
```

How Do Developers Learn Crypto?



Google android crypto example

www.androidsnippets.com/encryptdecrypt-strings

```
private static byte[] encrypt(byte[] raw, byte[] clear) throws Exception {
    SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
        Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
```

www.example8.com/category/view/id/15562

Example8.com

```
skeySpec = new SecretKeySpec(keyraw, "AES");
cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
```

stackoverflow.com/questions/6788018/android-encryption-decryption-with-aes

2 Answers

active oldest votes

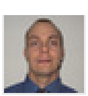
34

You could use functions like these:

```
private static byte[] encrypt(byte[] raw, byte[] clear) throws Exception {
        SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
        byte[] encrypted = cipher.doFinal(clear);
        return encrypted;
    }
```

www.java2s.com/Code/Android/Security/AESEncryption.htm

```
skeySpec = new SecretKeySpec(keyraw, "AES");
cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
```



kvandroidapp.blogspot.com/.../example-for-encrypt-and-decrypt...
by Klaus Villaca - in 44 Google+ circles
Nov 17, 2012 - **Example** for Encrypt and Decrypt using AES with **Android** 4.2

keyCzar

"Developers should not be able to inadvertently expose key material, use weak key lengths or deprecated algorithms, or improperly use cryptographic modes."

```
Crypter crypter = new Crypter("/path/to/your/keys");
String ciphertext = crypter.encrypt("Secret message");
```

Supported Operations

Encrypt	Decrypt	Authenticated Encryption, used to send messages

Crypto in Apple iOS

- Apple provides ECB and CBC
- Better default (CBC)
  - But: man CCCryptor (IV … initialization vector) "If CBC mode is selected and no IV is provided, an IV of all zeros will be used."
  - Constant IV: $m[0] == m'[0] \Rightarrow c[0] == c'[0]$

Automatically assess the security of mobile applications.

- [x] Privacy
- [x] Control-flow
- [x] Crypto misuse
- [ ] Many others

> 1 billion

> 1 million

Let’s make mobile secure!


Questions?