



# OWASP

Open Web Application  
Security Project

## Android / iPhone Risks and Solutions



Part of the *Reverse Engineering  
and Code Modification  
Prevention* OWASP Project

# Agenda

- Prevalence of App Cracking
- Current Threats / Risks
- Where to Go For Guidance?



CONNECT.

LEARN.

GROW.

Just How Prevalent is Code Modification?

# A RECENT STUDY OF APPS



**OWASP**  
Open Web Application  
Security Project

# Changes to the Threat Landscape

- ◆ Most web application vulnerabilities stem from malicious input provided by a user;
- ◆ To mitigate these risks, we:
  - ◆ Consider anything coming from the user's browser to be unsafe
  - ◆ Apply various secure-coding techniques to sanitize incoming data from any untrustworthy source (e.g., user browser);
- ◆ Threats posed by other sources are generally considered less likely to occur



# Changes to the Threat Landscape

- ◆ We still need to worry about traditional sources of untrustworthy input (user browsers);
- ◆ However, we're introducing new sources of untrustworthiness...

## Examples of distributed or untrusted environments

### Mobile Applications



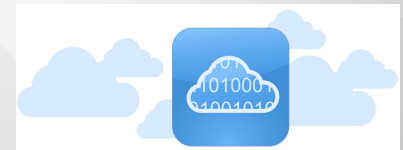
### Packaged Software



### Embedded Software



### Software in other at-risk / untrusted environments



# Application Integrity Problem



1. Software in untrusted environments is exposed to reverse-engineering, analysis, modification, and exploitation by attackers
2. Attackers can directly access the binary and compromise its integrity with various tools and techniques
3. Attackers may cause brand, revenue, or IP loss through reverse-engineering

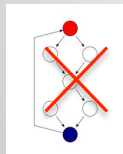
# What Do Binary Attacks Result In?



Compromise (disable, circumvent) of **security controls**, e.g., authentication, encryption, license management / checking, DRM, root / jailbreak detection



Exposure of **sensitive application information**, e.g., keys, certificates, credentials, metadata



Tampering with **critical business logic, control flows, and program operations**





# What Do Binary Attacks Result In?



Insertion of **malware or exploits** in the application and repackaging



Exposure of **application internals** (logic, vulnerabilities) via reverse-engineering



**IP theft** (e.g., proprietary algorithms) via reverse-engineering

**Piracy** and unauthorized distribution



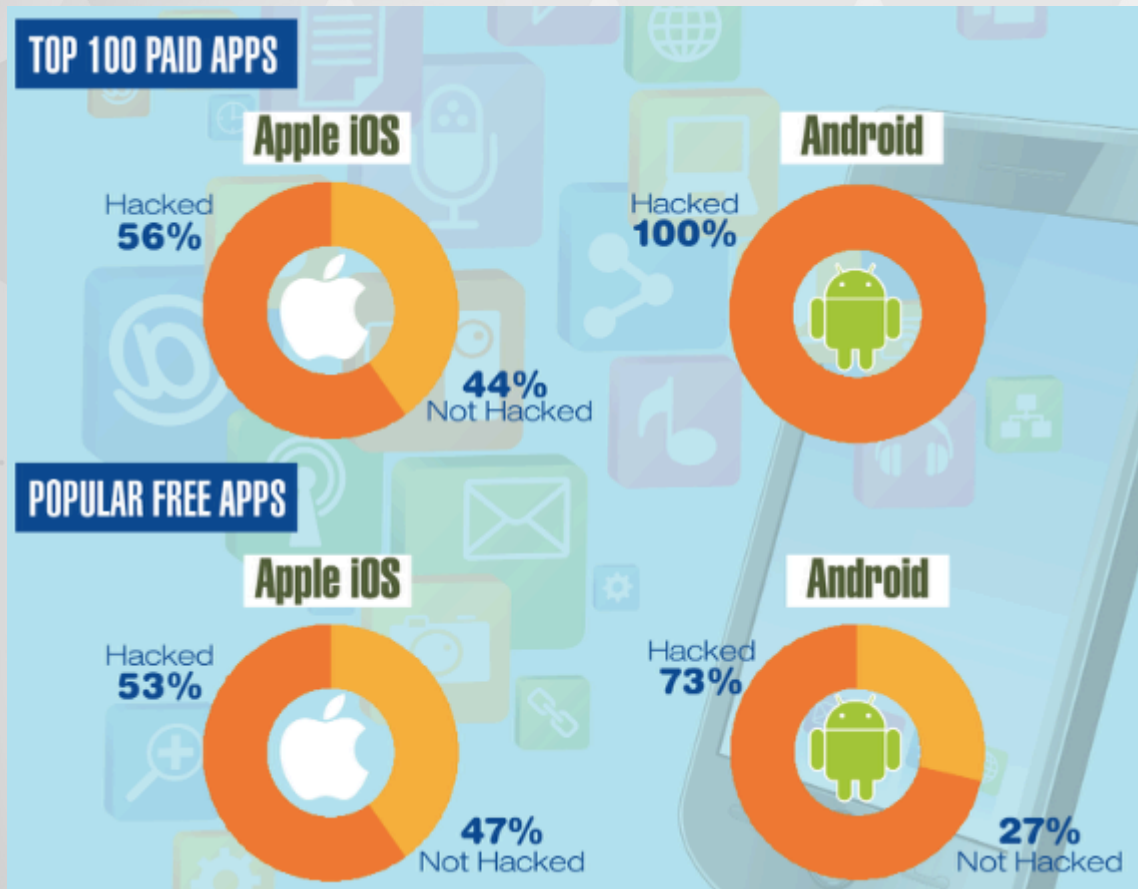


# Independent Surveys of the Wild...

- 86% of Android malware are repackaged versions of legitimate apps (IEEE Security & Privacy 2012)
- 90% of ethical hacking engagements were successful in gaining access to highly sensitive information (PwC mobile app report, 2012)
- Info Sec professionals say App Security and Mobile Security are #1 and #2 threat areas to their organizations (Frost & Sullivan, 2011)



# 2013 Arxan Study



- Analyzed Top 100 Apps for Android / iPhone for serious flaws
- Binary / HTML Modification extremely common

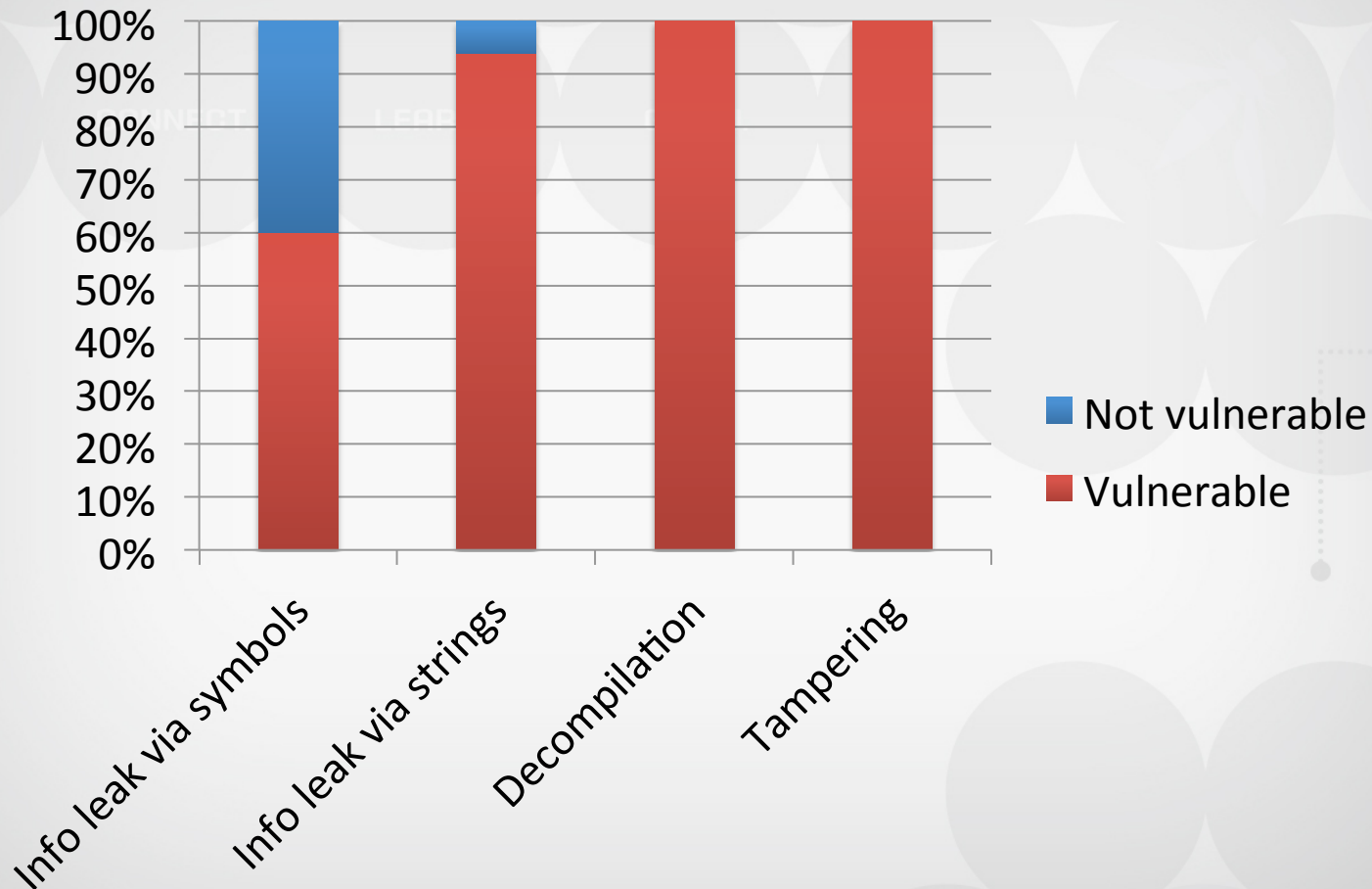


# Analysis

- What were the hackers interested in doing with these cracked apps?
  - Security Control Bypass
  - Adware / Spyware Code Injection
  - Repackaging (IP Theft)
  - Stealing Information About Users



# 2012 Study – Android Banking Vulnerabilities



CONNECT.

LEARN.

GROW.

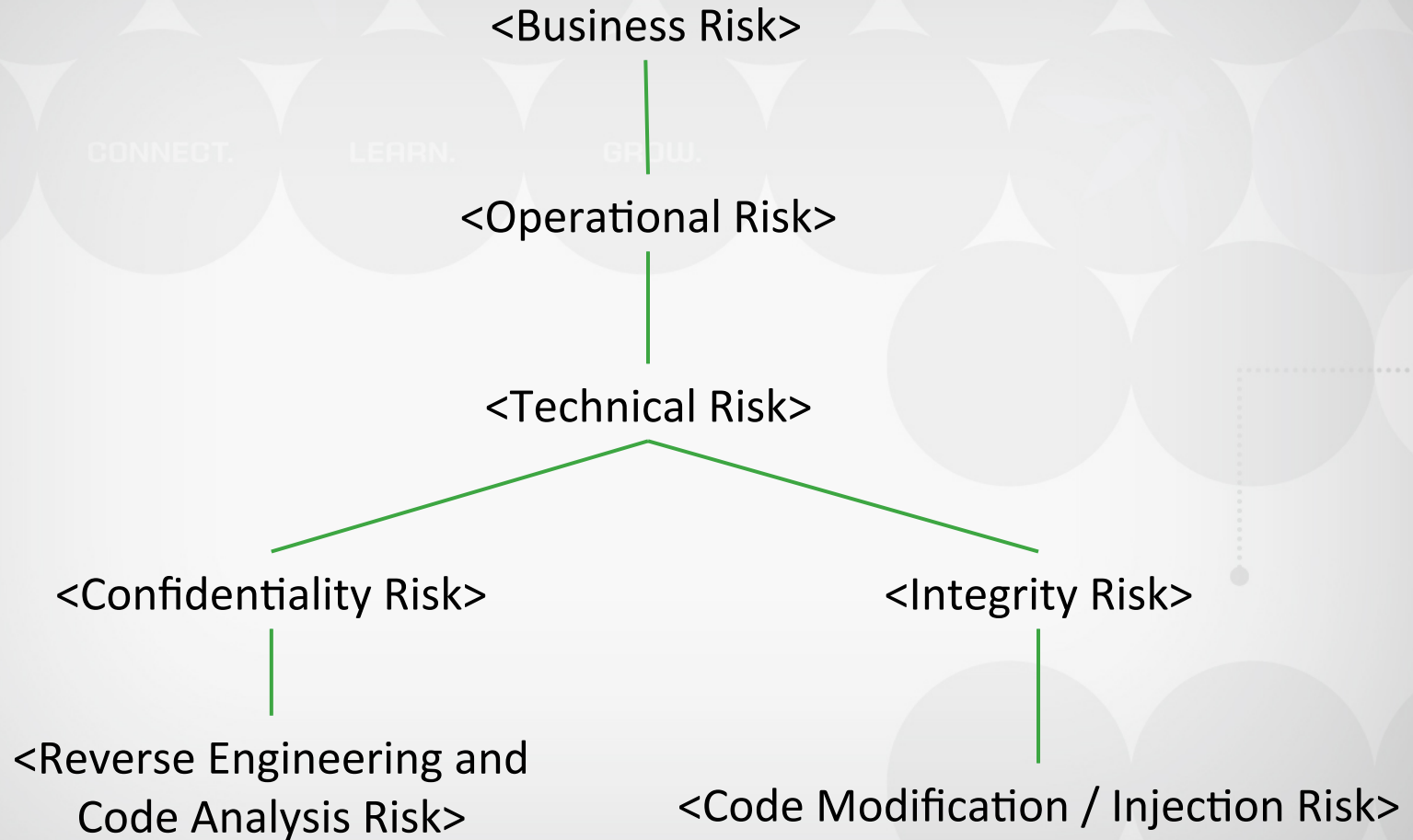
Things Organizations Should Think About

# **CURRENT RISKS & CODING FLAWS**



**OWASP**  
Open Web Application  
Security Project

# Android / iPhone Technical Risks



# Top Mobile Risks to Mitigate

- Code Modification Technical Risks
  - Repackaging
  - Method Swizzle With Behavioral Change
  - Security Control Bypass
  - Automated Jailbreak / Root Detection Disable
  - Presentation Layer Modification
  - Cryptographic Key Replacement



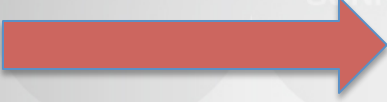


# Swizzling w/Behavioral Change

```
// Transaction-request delegate
- (IBAction)performTransaction:(id)sender
{
    if([self loginUserWithUsername:username
incomingPassword:password] != true)
    {
        UIAlertView *alert = [[UIAlertView
alloc] initWithTitle:@"Invalid User"
message:@"Authentication Failure" delegate:self
cancelButtonTitle:@"OK" otherButtonTitles:nil];

        [alert show];
        return;
    }

    // Perform sensitive operation here
}
```



This method  
will likely be  
swizzled and  
modified by an  
attacker



# Automated Jailbreak Bypass



```
-(BOOL) isJailbrokenEnvironment {  
    NSFileManager *filemgr = [NSFileManager defaultManager];  
  
    BOOL jailbrokenEnvironment =  
        [filemgr fileExistsAtPath:@"Applications/Cydia.app"];  
    return jailbrokenEnvironment;  
}
```

NOTE: Methods that appear to return a simple yes/no response and appear to be doing something sensitive are excellent candidates for simple code modification.



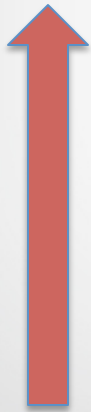
# Top Mobile Risks to Mitigate

- Reverse Engineering Risks
  - Exposed Method Signatures
  - API Monitoring
  - Exposed Data Symbols
  - Exposed String Tables
  - Algorithm Decompilation and Analysis
  - Application Decryption



# Cryptographic Key Theft

```
NSString* const szDecryptionKey =  
    @"32402394u2wewer90we90we09";  
  
NSString* const szEncryptionKey =  
    @"eroieuroiweruowieriw254234";
```



Flag hardcoded keys that could be easily found by an attacker through static or dynamic analysis.

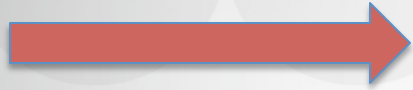


# AntiDebugger Checks

CONNECT.

LEARN.

GROW.



Common app  
entrypoints should  
check for the  
unauthorized  
presence of a  
debugger.

```
int main(int argc, char *argv[])  
{  
    @autoreleasepool {  
        return UIApplicationMain(  
    }  
}
```



**OWASP**  
Open Web Application  
Security Project

CONNECT.

LEARN.

GROW.

Useful OWASP Projects

# SOLUTIONS



**OWASP**  
Open Web Application  
Security Project

# Practical Solutions

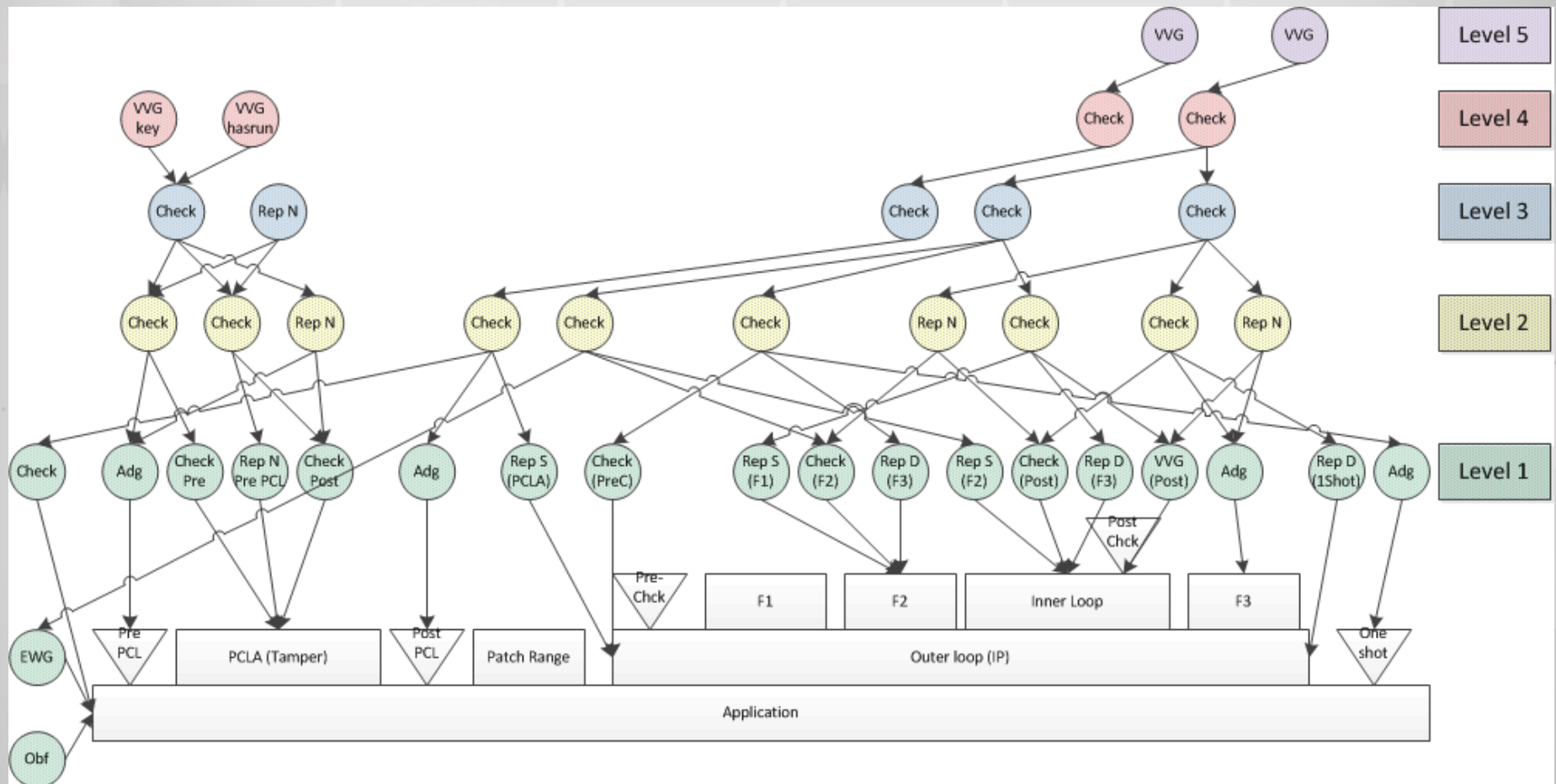
- Define and Implement Particular Security Controls In Your Mobile Apps:
  - Prevent Static / Dynamic Analysis
  - Detect Code Modification at Runtime
  - React to Modification at Runtime
  - Alert Intelligently

These controls must be written correctly and protected from reverse engineering / binary signatures!





# Defense-In-Depth Approach



# Practical Solutions

- Check out OWASP Project for more specific guidance / recommendations:



## *Reverse Engineering and Code Modification Prevention* OWASP Project

Subproject “Technical Risks of Reverse Engineering and  
Unauthorized Code Modification” for coding examples / solutions



**OWASP**  
Open Web Application  
Security Project

# Conclusions

- Unauthorized modification of Android / iPhone apps at the binary level is extremely common and overlooked by most security professionals
- There are lots of things you can do to mitigate this class of mobile vulnerabilities. You must code a correct solution and keep it correct via integrity controls.
- Check out the OWASP “Reverse Engineering” project for technical guidance
- Jonathan Carter – [jcarter@owasp.org](mailto:jcarter@owasp.org)

