Creating Secure Applications

Security lifecycle

Matt Murphy
Product Security Engineer

Overview

Why are you here?

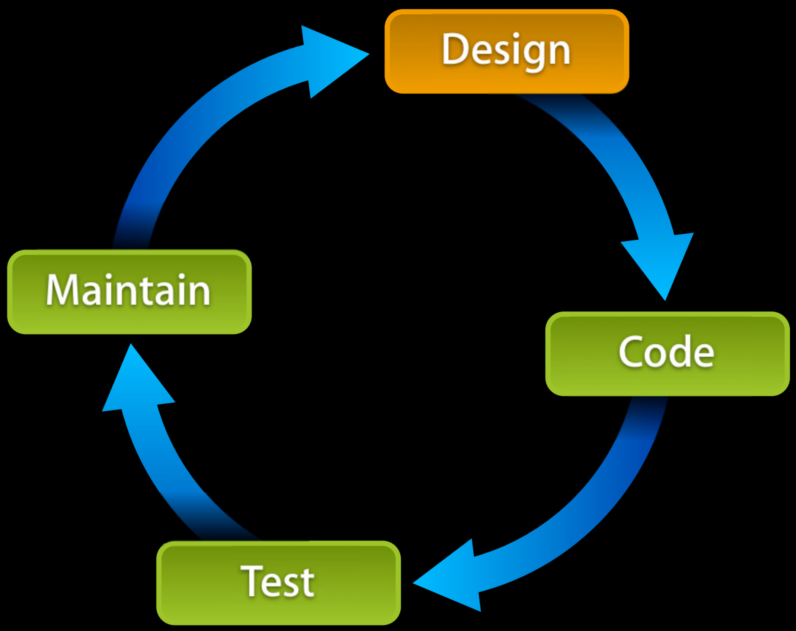
- Avoid the consequences of security issues
  - Negative press, lost revenue, etc.
- Realize that security is complicated
  - Trend toward highly connected environments
- Determine optimal ways to prevent security issues
  - Maximize benefits with available resources
  - Mistakes are expensive to fix later

Securing Your Application

In this part of the presentation

- Design for security
- Security tools
- Tips to avoid frequently seen security issues
- Later: Common Objective C / Cocoa security mistakes

Security Lifecycle



- Design
- Code
- Test
  - Automated tools
  - Manual testing/auditing
- Maintain
  - Fix bugs, deliver fixes
  - Not covered here

Design for Security

- Support privilege separation
- Run with reduced privilege
- Avoid setuid
- Protect data in transit
- More tips in the Secure Coding Guide

Design for Security

Support privilege separation

Only on Mac OS

- Don't use `AuthorizationExecuteWithPrivileges`
  - Factor privileged code into background daemon
- Use `launchd(8)` and service management APIs
  - `SMJobBless`, `SMJobSubmit`, etc.
  - See "SampleD" example

Designing for Security

- Support privilege separation
- Run with reduced privilege
- Avoid setuid
- Protect data in transit

Design for Security

Run with reduced privilege

Only on Mac OS

- Test as a standard user!
  - Your app should “just work”
  - If it doesn’t: you found a bug!
- Don’t rely on special capabilities of administrators
  - Don’t work for standard users
  - May break for administrators in the future

Run with Reduced Privilege

Avoid writing to…

Only on Mac OS

- /Applications
  - Including your app bundle
- /Applications/Utilities
- /Library and sub-directories
  - /Library/Application Support
  - /Library/Preferences
  - Etc.

Run with Reduced Privilege

Only on Mac OS

Common Problem	Solution(s)
Registration (serial number, license key, …)	Prompt at install time, while running with privilege
Global preferences, other privileged functionality	Use a `launchd(8)` job, protect with authorization as necessary
Custom installer	Use Installer.app if possible Use `installer(8)` command Install a `launchd(8)` job, remove when install completes

Designing for Security

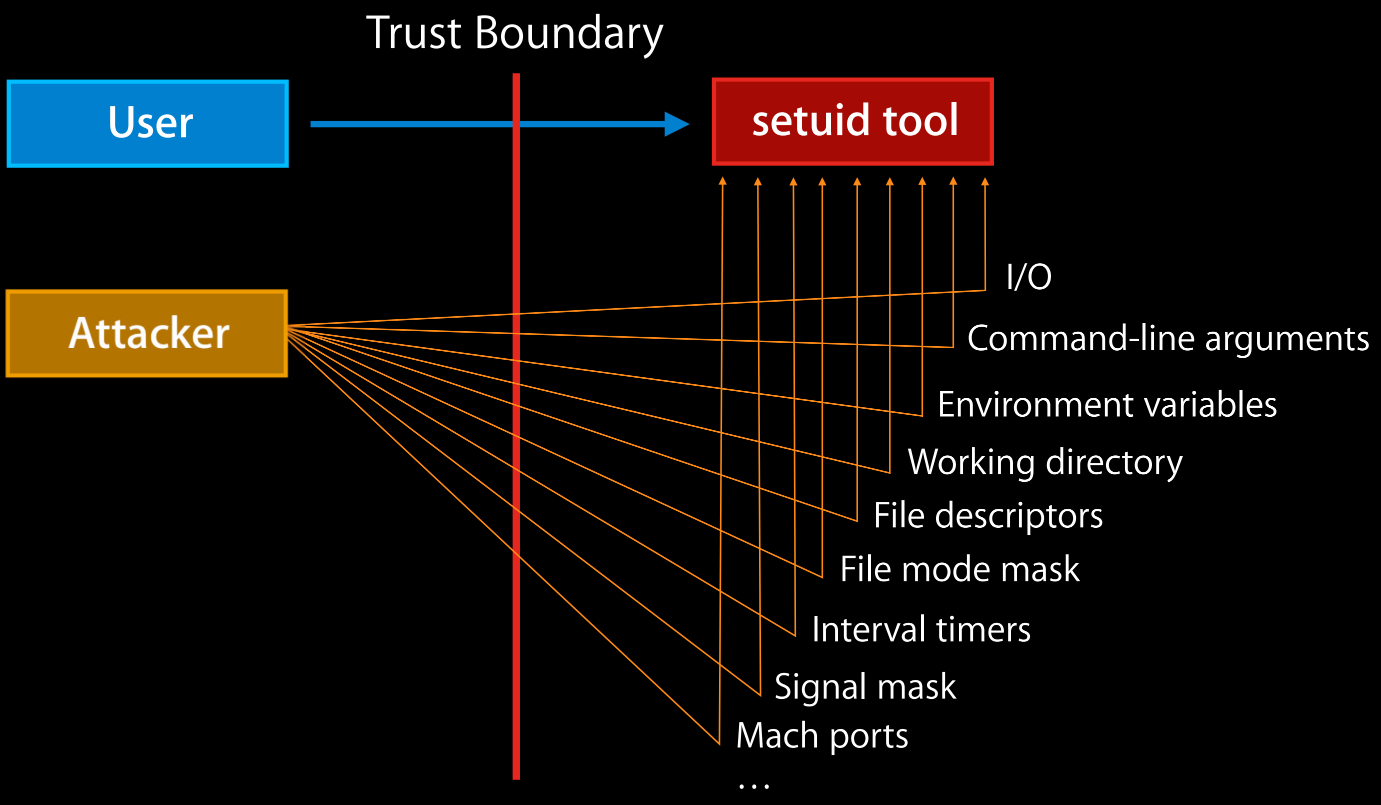
- Support privilege separation
- Run with reduced privilege
- Avoid setuid
- Protect data in transit

Avoid setuid

Only on Mac OS

- `setuid`/`setgid` is an attacker's dream
  - Control file descriptors, environment, etc.
  - Bugs in your own code, or lower-level APIs
- Don't write "self-repairing" privileged tools
  - Local user can alter binary
  - `setuid` bit may elevate malicious code to root!
    - Use installer packages and `RootAuthorization`

setuid Attack Surface



Designing for Security

- Support privilege separation
- Run with reduced privilege
- Avoid setuid
- Protect data in transit

Design for Security

Protect data in transit

- Assume users of your apps are mobile
  - MacBook, iPhone, iPod touch, iPad
  - Be suspicious of DNS, local network
- Protect sensitive data with SSL
  - `NSURLConnection` with https: URL
  - `CFReadStream` with SSL extensions

Design for Security

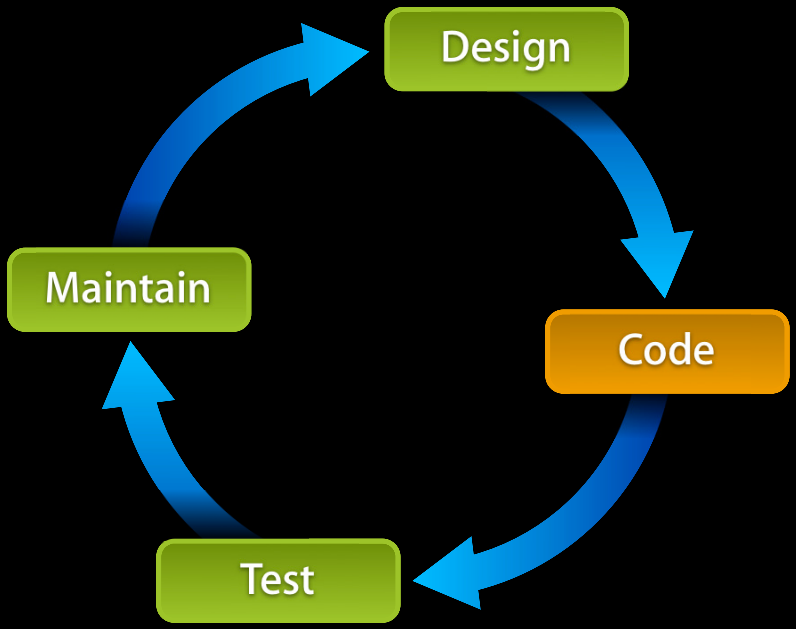
Protect data in transit

- Don’t disable chain validation!

```
CFDic      rySet     e(
  secur       ict
  kCFStr           idatesCertificateChain,
  kCFBoole         se);
CFReadStr          Property(
  stream
  kCFSt      Pro      SSLSettings,
  sec         DictR
```

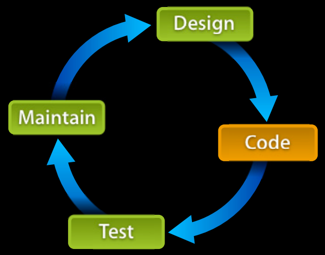
- Sign code, packages, etc.
  - Verify signing certificate

Security Lifecycle



- Design
- Code
- Test
  - Automated tools
  - Manual testing/auditing
- Maintain
  - Fix bugs, deliver fixes
  - Not covered here

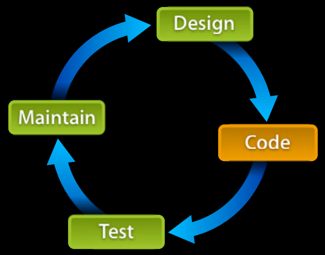
Secure Coding 101



- Safe file handling
- Permissions
- Bounds checking
- Integer overflows
- More in the Secure Coding Guide

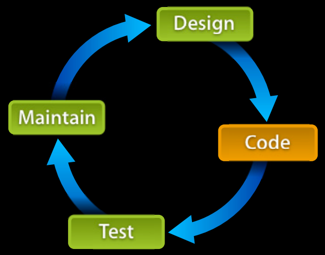
Secure Coding 101

Safe file handling



- Use safe temporary/cache directories
  - `confstr`
  - `NSTemporaryDirectory`
- Avoid world-writable directories
  - /tmp, /Library/Caches
- If you must use them, be careful
  - Higher level APIs (`writeToFile:`, `NSFileManager`, …) aren't safe
  - Only create files, always use `O_EXCL`

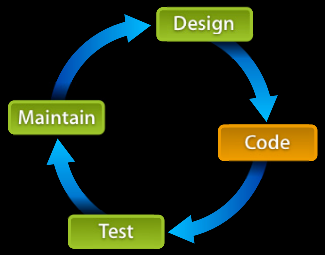
Secure Coding 101



- Safe file handling
- Permissions
- Bounds checking
- Integer overflows

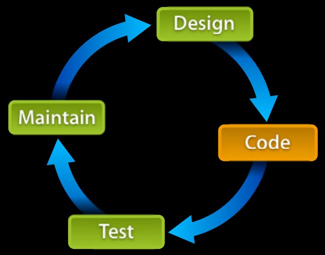
Secure Coding 101

Permissions



- Files are world-readable by default
  - Not appropriate for every file
  - Set tighter permissions where appropriate
- Avoid creating world-writable files
  - Subject to race conditions
  - Unprivileged user may damage file
  - Use a daemon to manage access

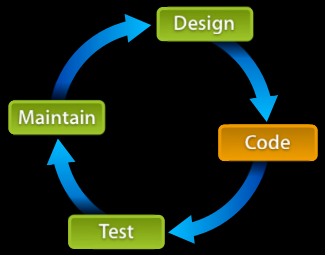
Secure Coding 101



- Safe file handling
- Permissions
- Bounds checking
- Integer overflows

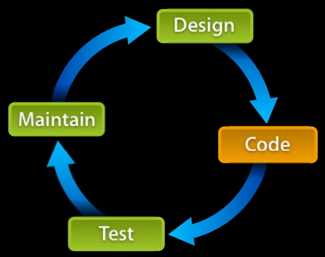
Secure Coding 101

Bounds checking



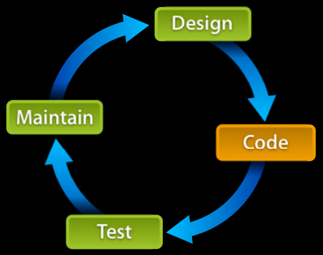
- Buffer overflows
  - Data too large for memory buffer allocated
  - Perform sanity checks
  - Use safe string functions

Use Safe String Functions

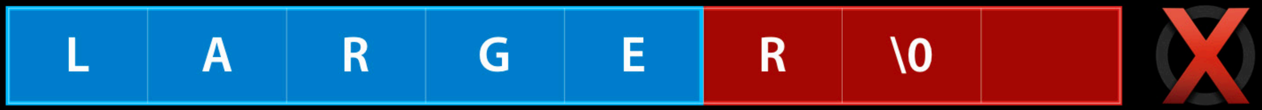


✗	✓
strcat, strcpy	strlcat, strlcpy
strncat, strncpy	strlcat, strlcpy
sprintf, vsprintf	snprintf, vsnprintf
gets	fgets

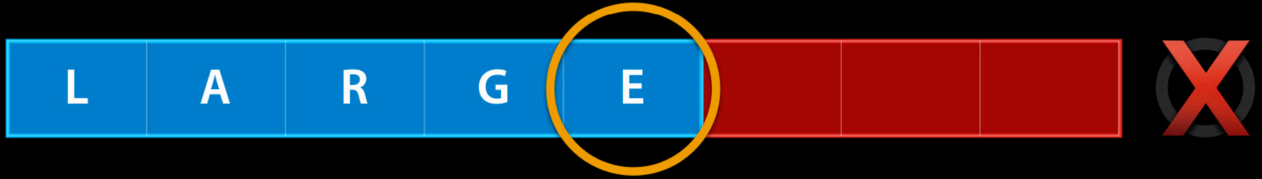
strcpy / strncpy / strlcpy



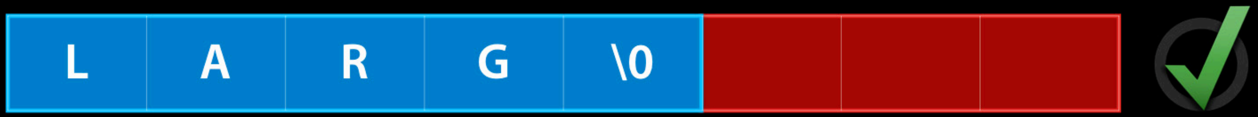
```
char destination[5]; char *source = “LARGER”;

strcpy(destination, source);
```

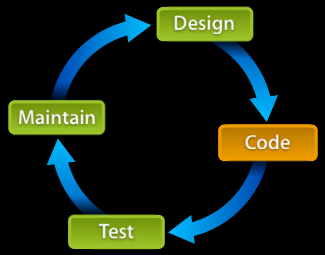


```
strncpy(destination, source, sizeof(destination));
```

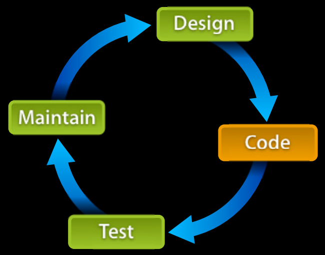


```
strlcpy(destination, source, sizeof(destination));
```





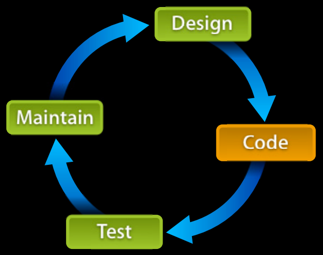
Secure Coding 101



- Safe file handling
- Permissions
- Bounds checking
- Integer overflows

Secure Coding 101

Integer overflows

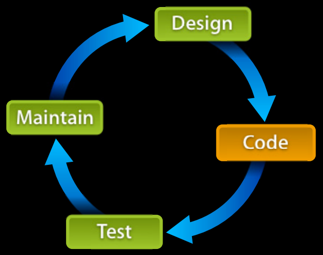


- Arithmetic operation produces value larger than integer type

```
struct binDataStruct {
  unsigned int nEntries;
  struct entry entryData[0];
};

struct binDataStruct *inputData = [someUntrustedData bytes];
NSData *copiedEntries = [NSMutableData dataWithLength:
    inputData->nEntries * sizeof(struct entry)];

for (i=0; i < inputData->nEntries; i++)
  memcpy([copiedWidgets mutableBytes] + i*sizeof(struct entry),
               &inputData->entryData[i],
               sizeof(struct entry));
```

Secure Coding 101

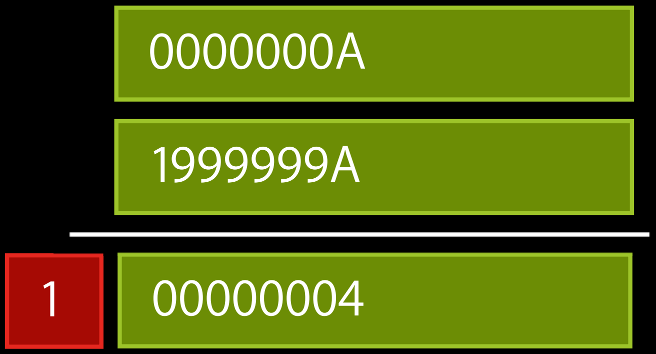
Integer overflows



- Use checkint API on untrusted integer operations

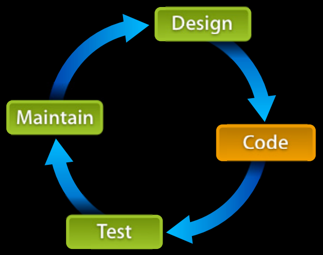
```
NSData *copiedEntries = [NSMutableData dataWithLength:
    inputData->nEntries * sizeof(struct entry)];
```

```
sizeof(struct entry)
inputData->nEntries
inputData->nEntries * sizeof(struct entry)
```



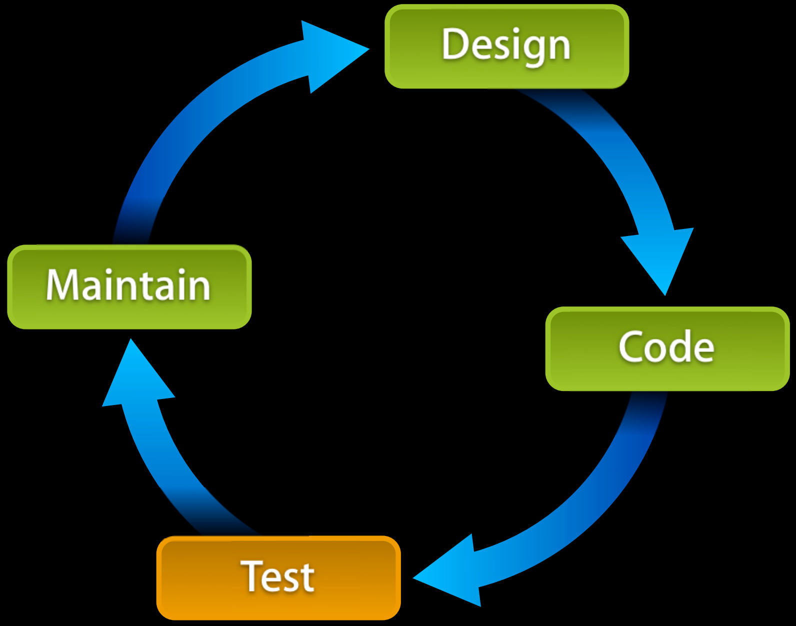
Secure Coding 101

Integer overflows: checkint



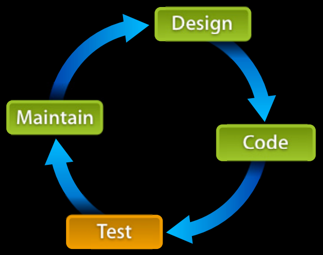
```
#include <checkint.h>
struct binDataStruct {
  unsigned int nEntries;
  struct entry entryData[0];
};
...
struct binDataStruct *inputData = [someUntrustedData bytes];
int intErr = CHECKINT_NO_ERROR;
unsigned int allocSize = check_uint32_mul(inputData->nWidgets,
    sizeof(struct widget), &intErr);
if (intErr != CHECKINT_NO_ERROR) goto fail;
NSData *copiedEntries = [NSMutableData dataWithLength:
    allocSize];

for (i=0; i < inputData->nEntries; i++)
  memcpy([copiedWidgets mutableBytes] + i*sizeof(struct entry),
             &inputData->entryData[i],
             sizeof(struct entry));
```

Security Lifecycle



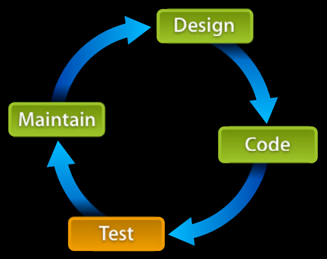
- Design
- Code
- Test
  - Automated tools
  - Manual testing/auditing
- Maintain
  - Fix bugs, deliver fixes
  - Not covered here

Test



- Static analysis
- Fuzzing

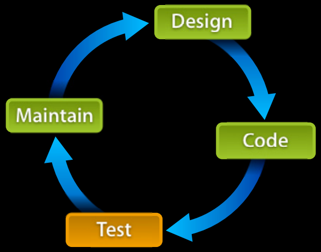
Static Analysis

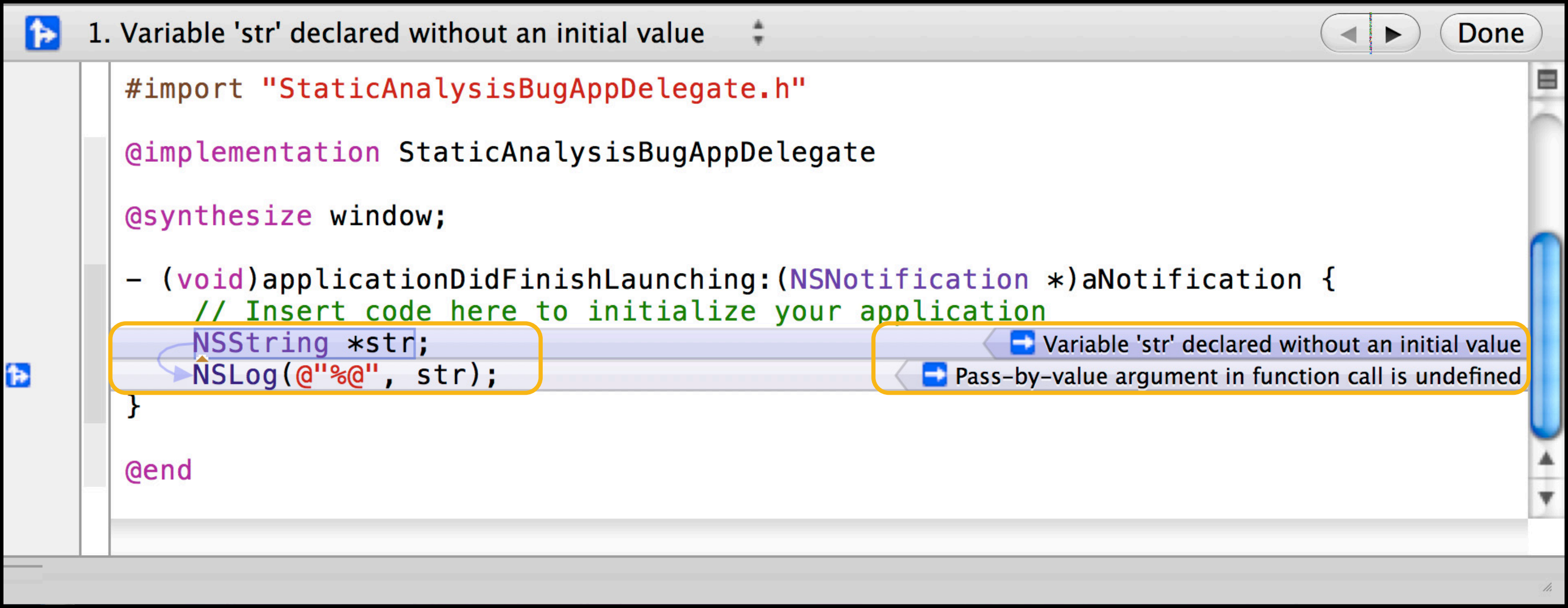


- Developer Tools now include a static analyzer
    - Run with the “Build and Analyze” menu item
- Checks code for common bugs:
    - Memory management issues
    - Small subset of buffer overflows
    - Some non-security bugs (dead store, etc.)
- Detailed warnings document data flow
- Rules aren’t very detailed, but improving

Static Analyzer Finds a Bug…

Example





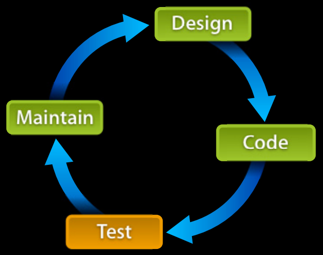
1. Variable 'str' declared without an initial value

```
#import "StaticAnalysisBugAppDelegate.h"

@implementation StaticAnalysisBugAppDelegate

@synthesize window;

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    // Insert code here to initialize your application
    NSString *str;
    NSLog(@"%@", str);
}

@end
```

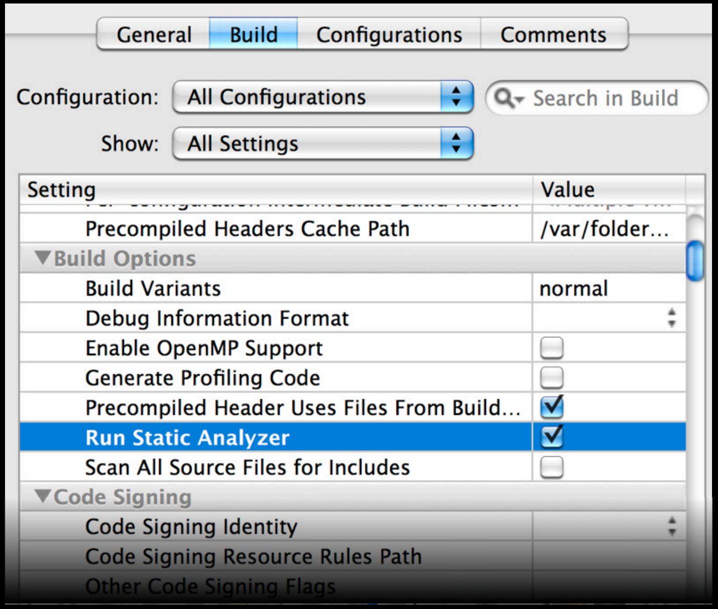
Variable 'str' declared without an initial value

Pass-by-value argument in function call is undefined

Static Analysis



- Use often for best results
  - Frequent runs catch regressions
  - New rules added in Developer Tools updates
- Project configuration option
  - Runs analyzer with every build

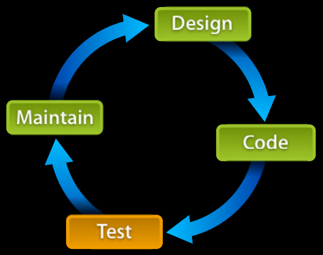


Configuration: All Configurations

Show: All Settings

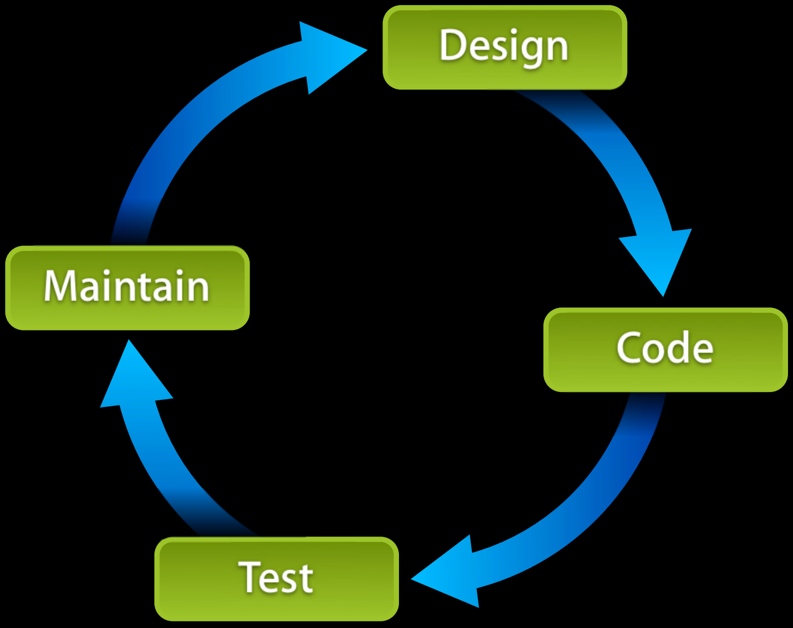
Setting	Value
Precompiled Headers Cache Path	/var/folder...
▼Build Options	
Build Variants	normal
Debug Information Format	
Enable OpenMP Support	☐
Generate Profiling Code	☐
Precompiled Header Uses Files From Build...	✓
Run Static Analyzer	✓
Scan All Source Files for Includes	☐
▼Code Signing	
Code Signing Identity	
Code Signing Resource Rules Path	
Other Code Signing Flags	

Fuzzing



- Subtly alter valid program inputs
  - File data
  - Network traffic
- Doesn't have to be complicated
- Program crash = bug
- CrashWrangler can help you prioritize
  - Run with crash logs, live targets
  - Heuristic for identifying exploitable bugs
  - Available as a download from connect.apple.com

Security Lifecycle



Securing a Cocoa Application

David Remahl
Product Security Engineer

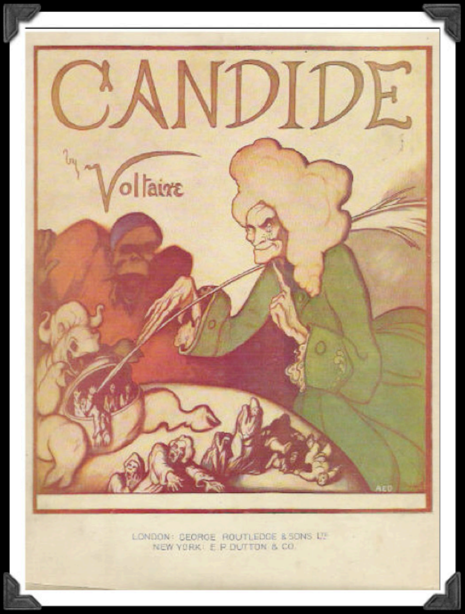
Naïveté

The magical and revolutionary feed reader

Naïveté

Both magical and revolutionary

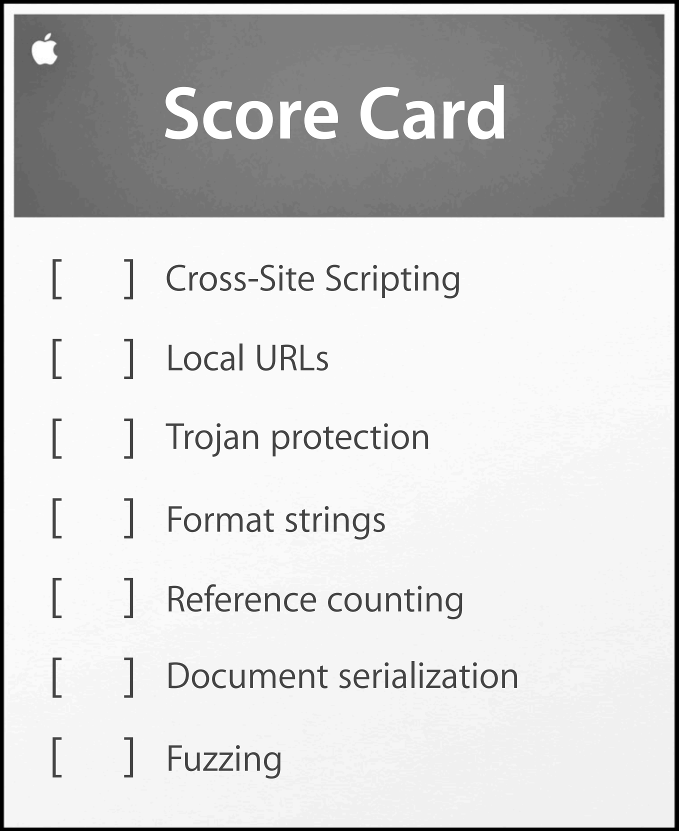
- Supports some well-formed Atom feeds
- Ground-breaking feature: Document based!
- Opens (emerging) industry-standard naive: URLs
- 512×512 icon
- Crashes: “It’s a feature, not a bug!”


CANDIDE
by Voltaire
LONDON: GEORGE ROUTLEDGE & SONS LTD
NEW YORK: E.P. DUTTON & CO.

Demo

Naïveté features

Naïveté


Score Card

[ ] Cross-Site Scripting

[ ] Local URLs

[ ] Trojan protection

[ ] Format strings

[ ] Reference counting

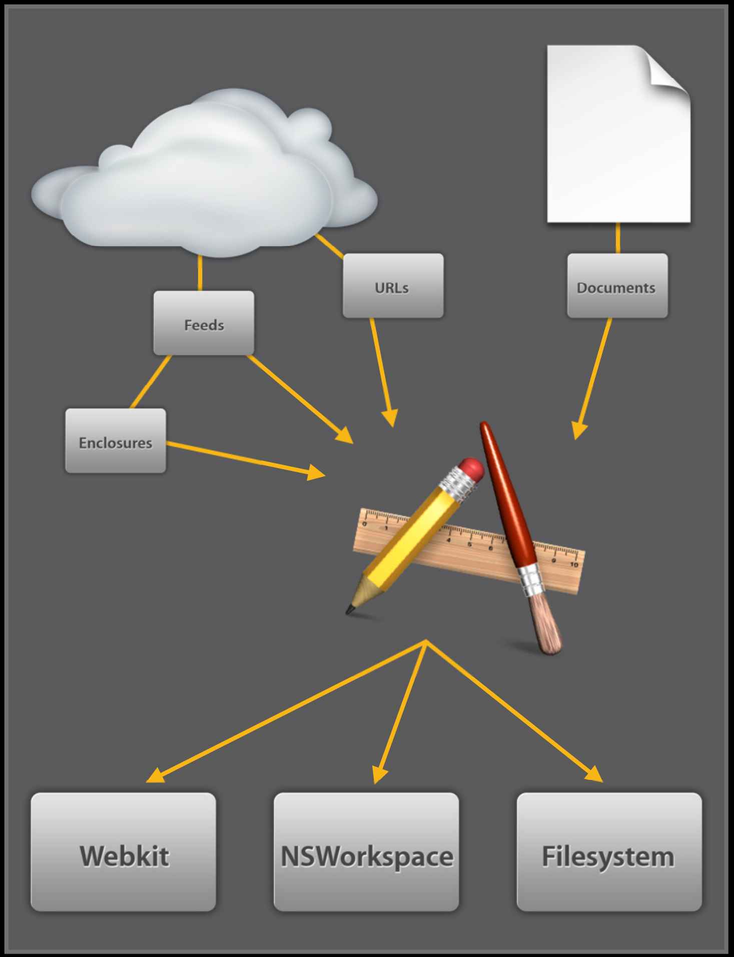
[ ] Document serialization

[ ] Fuzzing

Threat Model

Understanding the attack surface

- Entry points
  - naive: URLs (from Safari, etc)
  - Documents
  - Feeds
  - Enclosures



Threat Model

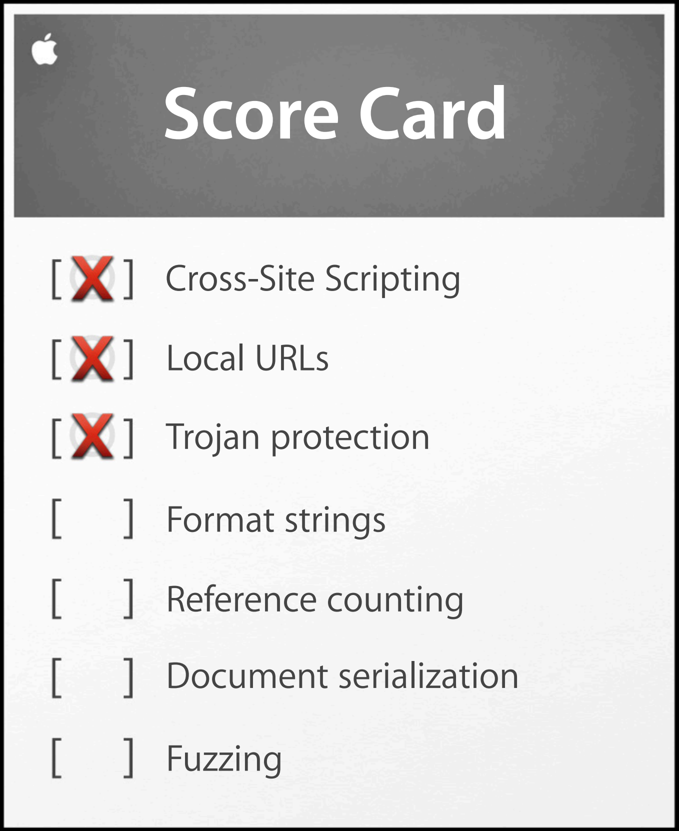
Understanding the attacks

- WebView
  - Document origin
  - Cross-site scripting (JavaScript injection)
  - External links
  - …
- URL handlers
  - Input validation
- Serialization format
  - …
- API documentation and Secure Coding Guide

Demo

Naïveté attacks

Naïveté



Score Card

[✗] Cross-Site Scripting

[✗] Local URLs

[✗] Trojan protection

[ ] Format strings

[ ] Reference counting

[ ] Document serialization

[ ] Fuzzing

Design Phase

Some lessons

- file: URLs are special
- Understand your APIs
- Applications that download files should use File Quarantine
  - Opt-in for all files created by the app
  - …or just for some, using `LSSetItemAttribute()`

File Formats

Playing safe

- Document formats have two layers
    - Semantic content (high-level)
    - Serialization format (low-level)
- What signifies a secure serialization format?
    - Simple and predictable
    - Small attack surface
    - Proper input validation

Demo

Naïveté’s document format

Naïveté

Score Card

- [✗] Cross-Site Scripting
- [✗] Local URLs
- [✗] Trojan protection
- [ ] Format strings
- [ ] Reference counting
- [✗] Document serialization
- [ ] Fuzzing

Archives and Serialization

✓

Safe for untrusted data

XML Property Lists

Binary Property Lists

NSXML

Core Data

Use for document formats,
network protocols,
shared data

✗

Unsafe for untrusted data

`NSArchiver`

`NSKeyedArchiver`

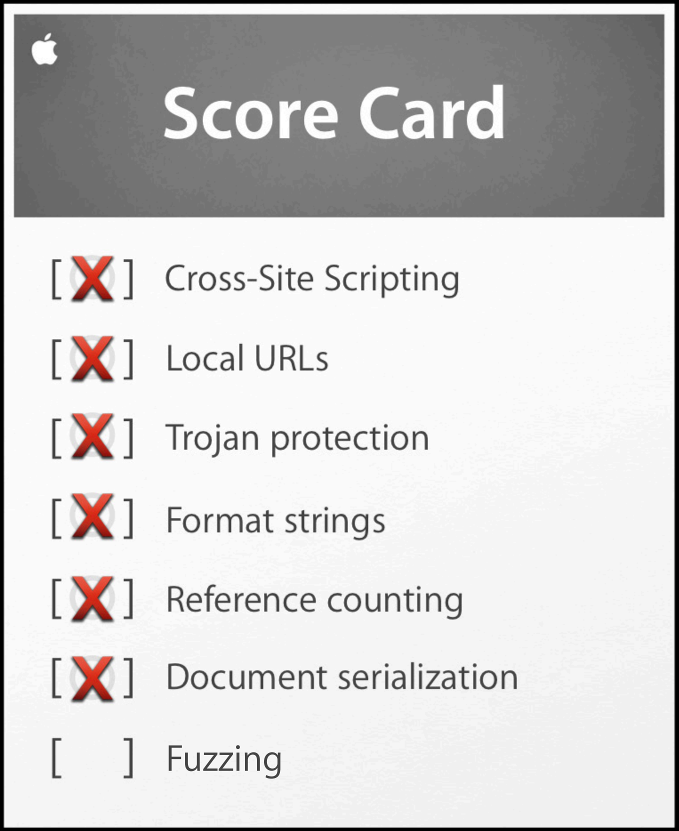
`NSSerialization`
(deprecated 10.2)

OK for preference files,
internal storage, frozen code,
trusted IPC

Demo

Static analysis and implementation issues

Naïveté



Score Card

[✗] Cross-Site Scripting

[✗] Local URLs

[✗] Trojan protection

[✗] Format strings

[✗] Reference counting

[✗] Document serialization

[ ] Fuzzing

Implementation Phase

More lessons

- Static analysis helps, but does not catch everything
- Be careful with format strings
- Reference counting and weak references are hard
  - Garbage Collection avoids some pitfalls

Testing Phase

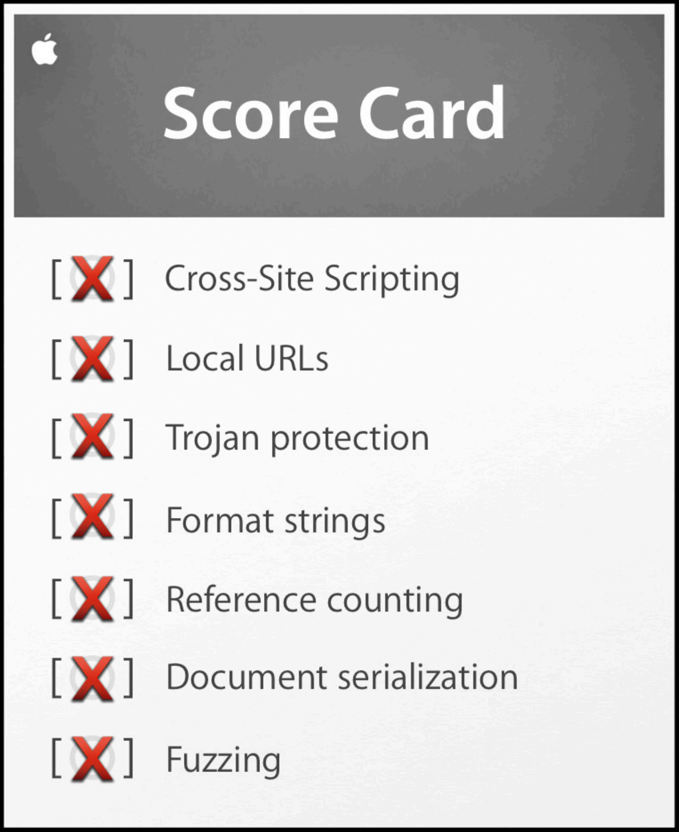
Fuzzing is easy and effective

- `pluzz.py`—A simple property list fuzzer in less than an hour
  - Enumerates the hierarchy of a plist
  - Replaces objects in plist with other types and boundary values
  - Writes a copy for each permutation into a numbered file
- Run with CrashWrangler

Demo

Property list fuzzing

Naïveté



Score Card

[✗] Cross-Site Scripting

[✗] Local URLs

[✗] Trojan protection

[✗] Format strings

[✗] Reference counting

[✗] Document serialization

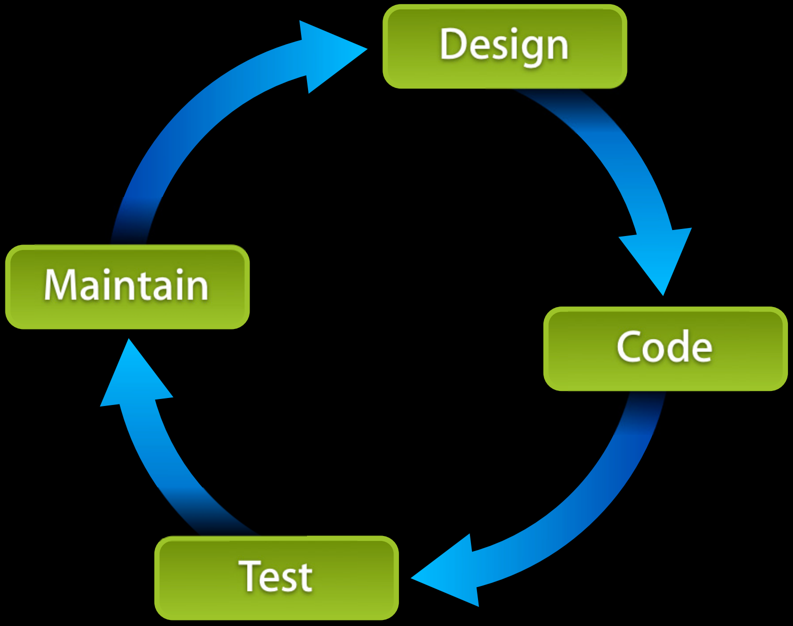
[✗] Fuzzing

Testing Phase

More on testing

- Fuzzing is an important part of the testing strategy
- Try multiple fuzzers
  - binary, random values, boundary values, dumb, guided, …
- Also use:
  - Unit testing (focus on edge cases)
  - Penetration testing (try to break it)

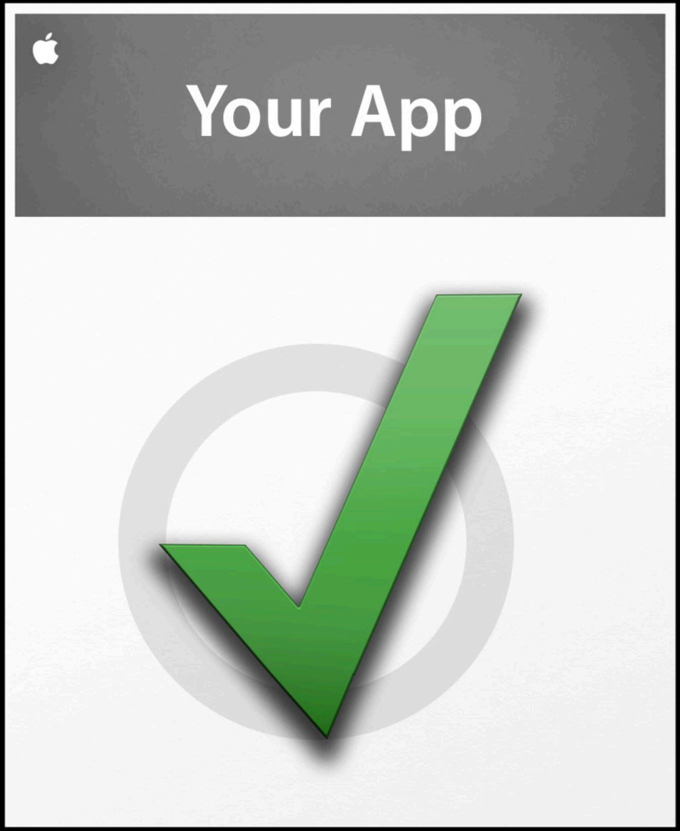
Summary



- Think about security throughout the development process
- Be aware of the security properties of the APIs you use
- Understand the attacks that affect your problem space
- Take advantage of hardening techniques and security APIs

Next Steps?

- Visit the Dev Forums Security section
- Read the Secure Coding Guide
- Run the Static Analyzer
- Fuzz your app



Related Sessions

Launch-on-Demand	Russian Hill Thursday 4:30PM
Network Apps for iPhone OS, Part 1	Pacific Heights Wednesday 2:00PM
Securing Application Data	Marina Thursday 11:30AM

Labs

iPhone OS and Mac OS X Security Lab	Core OS Lab A Thursday 2:00PM

WWDC10

[redacted]