Developing Secure Mobile Apps

SecAppDev 2012

Apple iOS (Xcode) edition

KRvW Associates, LLC

Copyright© 2012 KRvW Associates, LLC

Platform Architecture

What the iOS / hardware platform offers us in the way of protection

KRvW Associates, LLC

Copyright© 2012 KRvW Associates, LLC

iOS application architecture

The iOS platform is basically a subset of a regular Mac OS X system's

- -From user level (Cocoa) down through Darwin kernel
- Apps can reach down as they choose to
- -Only published APIs are permitted, however



Key security features

Application sandboxing App store protection Hardware encryption Keychains SSL and certificates



Application sandboxing

By policy, apps are only permitted to access resources in their sandbox

- Inter-app comms are by established APIs only
 - URLs, keychains (limited)
- -File i/o in ~/Documents only

Sounds pretty good, eh?



App store protection

Access is via digital signatures

- Only registered developers may introduce apps to store
- Only signed apps may be installed on devices
- Sounds good also, right?
- -But then there's jailbreaking...
- -Easy and free
- -Completely bypasses sigs



Hardware encryption

Each iOS device (as of 3g) has hardware crypto module

- -Unique AES-256 key for every iOS device
- -Sensitive data hardware encrypted
- Sounds brilliant, right? –Well...



Keychains

Keychain API provided for storage of small amounts of sensitive data

- -Login credentials, passwords, etc.
- -Encrypted using hardware AES
- Also sounds wonderful –Wait for it...



SSL and x.509 certificate handling

API provided for SSL and certificate verification

- Basic client to server SSL is easy
- Mutual verification of certificates is achievable, but API is complex
- Overall, pretty solid –Whew!



And a few glitches...

Keyboard data Screen snapshots Hardware encryption is flawed



Keyboard data

- All "keystrokes" are stored
- -Used for auto-correct feature
- -Nice spell checker
- Key data can be harvested using forensics procedures
- -Passwords, credit cards...
- -Needle in haystack?



Screen snapshots

Devices routinely grab screen snapshots and store in JPG

- -Used for minimizing app animation
- -It looks pretty
- WHAT?!
- -It's a problem
- Requires local access to device, but still...



But the clincher

Hardware module protects unique key via device PIN –PIN can trivially be disabled –Jailbreak software No more protection...



Discouraged?

If we build our apps using these protections only, we'll have problems

- -But consider risk
- -What is your app's "so what?" factor?
- -What data are you protecting?
- -From whom?
- Might be enough for some purposes



Copyright© 2012 KRvW Associates, LLC

But for a serious enterprise...

The protections provided are simply not adequate to protect serious data

- -Financial
- -Privacy
- -Credit cards
- We need to further lock down
- -But how much is enough?



Application Architecture

How do we build our apps securely?

KRvW Associates, LLC

Copyright© 2012 KRvW Associates, LLC

Common app types

Web app Web-client hybrid App – Stand alone – Client-server – Networked Decision time...



Web applications

- Don't laugh--you really can do a lot with them
- -Dashcode is pretty slick
- Can give a very solid UI to a web app
- Pros and cons
- -Data on server (mostly)
- -No app store to go through
- -Requires connectivity



Web-client hybrid

- Local app with web views –Still use Dashcode on web views
- Local resources available
 via Javascript
 - Location services, etc
- Best of both worlds?
- -Powerful, dynamic
- -Still requires connection



iOS app -- client-server

Most common app for enterprises

- Basically alternate web client for many
- -But with iOS UI on client side
- Server manages access, sessions, etc.
- Watch out for local storage
- -Avoid if possible
- -Encrypt if not





iOS app -- networked

- Other network architectures also
- -Internet-only
- -P2P apps
- Not common for enterprise purposes



Common Security Mechanisms

Now let's build security in

KRvW Associates, LLC

Copyright© 2012 KRvW Associates, LLC

Common mechanisms

Input validation Output escaping Authentication Session handling Protecting secrets -At rest -In transit SQL connections



Input validation

Positive vs negative validation

-Dangerous until proven safe

-Don't just block the bad

Consider the failures of desktop anti-virus tools –Signatures of known viruses



Input validation architecture

We have several choices –Some good, some bad Positive validation is our aim

- Consider tiers of security in an enterprise app
- -Tier 1: block the bad
- -Tier 2: block and log
- -Tier 3: block, log, and take evasive action to protect



Input validation (in iOS)

// RFC 2822 email addres regex.

NSString *emailRegex =

// Create the predicate and evaluate.

NSPredicate *regExPredicate =

[NSPredicate predicateWithFormat:@"SELF MATCHES %@", emailRegEx]; BOOL validEmail = [regExPredicate evaluateWithObject:emailAddress];

```
if (validEmail) {
...
} else {
...
}
```

Input validation (server side Java)

```
protected final static String ALPHA_NUMERIC =
```

```
"^[a-zA-Z0-9\s.\-]+$";
```

```
// we only want case insensitive letters and numbers
```

```
public boolean validate(HttpServletRequest request, String
parameterName) {
```

```
boolean result = false;
Pattern pattern = null;
parameterValue = request.getParameter(parameterName);
if(parameterValue != null) {
    pattern = Pattern.compile(ALPHA_NUMERIC);
    result = pattern.matcher(parameterValue).matches();
    return result;
} else
```

```
{ // take alternate action }
```

Output encoding

Principle is to ensure data output does no harm in output context

- -Output escaping of control chars
 - How do you drop a "<" into an XML file?
- -Consider all the possible output contexts

nection ement.executable at next())

Output encoding

This is normally server side code

Intent is to take dangerous data and output harmlessly

Especially want to block Javascript (XSS)

In iOS, not as much control, but

-Never point UIWebView to untrusted content



Output encoding (server side)

Context <body> UNTRUSTED DATA HERE </body> <div> UNTRUSTED DATA HERE </div> other normal HTML elements

String safe =
ESAPI.encoder().encodeForHTML(request.getParameter("input"));

Authentication

This next example is for authenticating an app user to a server securely

-Server takes POST request, just like a web app



Authentication (forms-style)

// Initialize the request with the YouTube/Google ClientLogin URL (SSL).
NSString youTubeAuthURL = @"<u>https://www.google.com/accounts/ClientLogin</u>";
NSMutableRequest *request =
[NSMutableURLRequest requestWithURL:[NSURL URLWithString:youTubeAuthURL]];

[request setHTTPMethod:@"POST"];

// Build the request body (form submissions POST).
NSString *requestBody =
[NSString stringWithFormat:@"Email=%@&Passwd=%@&service=youtube&source=%@",
emailAddressField.text, passwordField.text, @"Test"];

[request setHTTPBody:[requestBody dataUsingEncoding:NSUTF8StringEncoding]];

// Submit the request.
[[NSURLConnection alloc] initWithRequest:request delegate:self];

// Implement the NSURLConnection delegate methods to handle response.

Copyright© 2012 KRvW Associates, LLC

Mutual authentication

We may also want to use x.509 certificates and SSL to do strong mutual authentication

More complicated, but stronger

Certificate framework in NSURL is complex and tough to use

(Example is long--see src)



Authentication (mutual)

```
/ Delegate method for NSURLConnection that determines whether client can handle
```

 $\ensuremath{\textit{//}}\xspace$ the requested form of authentication.

```
- (BOOL)connection:(NSURLConnection *)connection
canAuthenticateAgainstProtectionSpace:(NSURLProtectionSpace *)protectionSpace {
```

```
// Only handle mutual auth for the purpose of this example.
```

```
if ([[protectionSpace authenticationMethod] isEqual:NSURLAuthenticationMethodClientCertificate]) {
    return YES;
```

```
} else {
  return NO;
}
```

```
}
```

// Delegate method for NSURLConnection that presents the authentication

```
// credentials to the server.
```

- (void)connection:(NSURLConnection *)connection

```
didReceiveAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge {
```

```
id<NSURLAuthenticationChallengeSender> sender = [challenge sender];
NSURLCredential *credential;
NSMutableArray *certArray = [NSMutableArray array];
```

Session handling

Normally controlled on the server for client-server apps

Varies tremendously from one tech and app container to another

Basic session rules apply Testing does help, though



Testing

Checklist

- -Credentials encrypted in transit?
- -Username enumeration or harvesting?
- -Dictionary and brute force attacks
- -Bypassing
- -Password remember and reset

- -Password geometry
- -Logout and browser caching
- Dynamic validation is very helpful

POST http://www.example.com/AuthenticationServlet HTTP/1.1

Host: www.example.com

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko/20100404

Accept: text/xml,application/xml,application/xhtml+xml

Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3

Accept-Encoding: gzip,deflate

```
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

Keep-Alive: 300

Connection: keep-alive

Referer: http://www.example.com/index.jsp

Cookie:

JSESSIONID=LVrRRQQXgwyWpW7QMnS49vtW1yBdqn98CGlkP4jTvVCGdyPkmn3S !

Content-Type: application/x-www-form-urlencoded Content-length: 64

```
delegated_service=218&User=test&Pass=test&Submit=SUBMIT
```

POST https://www.example.com:443/login.do HTTP/1.1

Host: www.example.com

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko/ 20100404

Accept: text/xml,application/xml,application/xhtml+xml,text/html

Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3

Accept-Encoding: gzip,deflate

```
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

Keep-Alive: 300

Connection: keep-alive

Referer: https://www.example.com/home.do

Cookie: language=English;

Content-Type: application/x-www-form-urlencoded

Content-length: 50

Command=Login&User=test&Pass=test

POST https://www.example.com:443/login.do HTTP/1.1 Host: www.example.com User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14) Gecko/20100404 Accept: text/xml,application/xml,application/xhtml+xml,text/html Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3 Accept-Encoding: gzip,deflate Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7 Keep-Alive: 300 Connection: keep-alive Referer: http://www.example.com/homepage.do Cookie[.] SERVTIMSESSIONID=s2JyLkvDJ9ZhX3yr5BJ3DFLkdphH0QNSJ3VQB6pLhjkW6F Content-Type: application/x-www-form-urlencoded Content-length: 45

User=test&Pass=test&portal=ExamplePortal

GET https://www.example.com/success.html?user=test&pass=test HTTP/ 1.1

Host: www.example.com

```
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.14)
Gecko/20100404
```

Accept: text/xml,application/xml,application/xhtml+xml,text/html

```
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
```

```
Accept-Encoding: gzip,deflate
```

```
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

```
Keep-Alive: 300
```

```
Connection: keep-alive
```

Referer: https://www.example.com/form.html

If-Modified-Since: Mon, 30 Jun 2010 07:55:11 GMT

```
If-None-Match: "43a01-5b-4868915f"
```

Access control (authorization)

On the iOS device itself, apps have access to everything in their sandbox

Server side must be designed and built in like any web app



Authorization basics

Question every action –Is the user allowed to access this

- File
- Function
- Data
- Etc.

By role or by user -Complexity issues -Maintainability issues -Creeping exceptions

Role-based access control

Must be planned carefully Clear definitions of -Users

- -Objects
- -Functions
- -Roles
- -Privileges

Plan for growth Even when done well, exceptions will happen

ESAPI access control

In the presentation layer:

```
<% if ( ESAPI.accessController().isAuthorizedForFunction( ADMIN_FUNCTION ) ) { %>
<a href="/doAdminFunction">ADMIN</a>
<% } else { %>
<a href="/doNormalFunction">NORMAL</a>
<% } %>
```

In the business logic layer:

```
try {
```

ESAPI.accessController().assertAuthorizedForFunction(BUSINESS_FUNCTION); // execute BUSINESS_FUNCTION } catch (AccessControlException ace) { ... attack in progress

Protecting secrets at rest

- The biggest problem by far is key management
- -How do you generate a strong key?
- –Where do you store the key?
- -What happens if the user loses his key?
- Too strong and user support may be an issue



Built-in file protection (weak)

// API for writing to a file using writeToFile API

- (BOOL)writeToFile:(NSString *)path options: (NSDataWritingOptions)mask error:(NSError **)errorPtr

// To protect the file, include the
// NSDataWritingFileProtectionComplete option

Protecting secrets at rest (keychain)

// Write username/password combo to keychain.

BOOL writeSuccess = [SFHFKeychainUtils storeUsername:username andPassword:password

forServiceName:@"com.krvw.ios.KeychainStorage" updateExisting:YES error:nil];

•••

// Read password from keychain given username.
NSString *password = [SFHFKeychainUtils getPasswordForUsername:username
andServiceName:@"com.krvw.ios.KeychainStorage" error:nil];

// Delete username/password combo from keychain.
BOOL deleteSuccess = [SFHFKeychainUtils deleteItemForUsername:username
andServiceName:@"com.krvw.ios.KeychainStorage" error:nil];

Enter SQLcipher

Open source extension to SQLite

- -Free
- -Uses OpenSSL to AES-256 encrypt database
- -Uses PBKDF2 for key expansion
- -Generally accepted crypto standards
- Available from
- <u>http://sqlcipher.net</u>



Protecting secrets at rest (SQLcipher)

sqlite3_stmt *compiledStmt;

// Unlock the database with the key (normally obtained via user input).

// This must be called before any other SQL operation.

sqlite3_exec(credentialsDB, "PRAGMA key = 'secretKey!'", NULL, NULL, NULL);

```
// Database now unlocked; perform normal SQLite queries/statments.
```

```
•••
```

```
// Create creds database if it doesn't already exist.
```

const char *createStmt =

```
"CREATE TABLE IF NOT EXISTS creds (id INTEGER PRIMARY KEY AUTOINCREMENT, username TEXT, password TEXT)";
```

```
sqlite3_exec(credentialsDB, createStmt, NULL, NULL, NULL);
```

// Check to see if the user exists.

```
const char *queryStmt = "SELECT id FROM creds WHERE username=?";
```

```
int userID = -1;
```

```
if (sqlite3_prepare_v2(credentialsDB, queryStmt, -1, &compiledStmt, NULL) == SQLITE_OK) {
```

```
sqlite3_bind_text(compiledStmt, 1, [username UTF8String], -1, SQLITE_TRANSIENT);
```

```
while (sqlite3_step(compiledStmt) == SQLITE_ROW) {
```

```
userID = sqlite3_column_int(compiledStmt, 0);
```

```
}
if (userID >= 1) {
```

```
// User exists in database.
```

```
...
}
Copyright© 2012 KRvW Associates, LLC
```

Don't forget screen shots

When an app backgrounds, a screen shot is snapped

- -Safest bet is to disallow
 - UIApplicationExitsOnSuspend
 - Set in info.plist
- -If not feasible, clear data
- -Detect/control backgrounds
 - Several key methods for controlling backgrounding



Relevant backgrounding methods

Also look at

- -applicationWillEnterForeground:
- -applicationWillTerminate:
- -applicationDidBecomeActive
- -applicationWillResignActive
- -applicationDidEnterBackground
- -application: didFinishLaunchingWithOptions:

Protecting secrets in transit

Key management still matters, but SSL largely takes care of that

- -Basic SSL is pretty easy in NSURL
- -Mutual certificates are stronger, but far more complicated
- –NSURL is awkward, but it works
 - See previous example



Protecting secrets in transit

// Note the "https" protocol in the URL.

NSString *userJSONEndpoint =

[[NSString alloc] initWithString:@"https://www.secure.com/api/user"];

// Initialize the request with the HTTPS URL.

NSMutableURLRequest *request =

[MSMutableURLRequest requestWithURL:[NSURL URLWithString:userJSONEndpoint]];

// Set method (POST), relevant headers and body (jsonAsString assumed to be // generated elsewhere). [request setHTTPMethod:@"POST"]; [request setValue:@"application/json" forHTTPHeaderField:@"Content-Type"]; [request setValue:@"application/json" forHTTPHeaderField:@"Accept"]; [request setHTTPBody:[jsonAsString dataUsingEncoding:NSUTF8StringEncoding]];

// Submit the request.
[[NSURLConnection alloc] initWithRequest:request delegate:self];

// Implement delegate methods for NSURLConnection to handle request lifecycle.

••

SQL connections

Biggest security problem is using a mutable API

-Weak to SQL injection

Must use immutable API

-Similar to PreparedStatement in Java or C#



SQL connections

// Update a users's stored credentials.

sqlite3_stmt *compiledStmt;

const char *updateStr = "UPDATE credentials SET username=?, password=? WHERE id=?";

// Prepare the compiled statement.

if (sqlite3_prepare_v2(database, updateStr, -1, &compiledStmt, NULL) == SQLITE_OK) { // Bind the username and password strings.

sqlite3_bind_text(compiledStmt, 1, [username UTF8String], -1, SQLITE_TRANSIENT); sqlite3_bind_text(compiledStmt, 2, [password UTF8String], -1, SQLITE_TRANSIENT);

```
// Bind the id integer.
sqlite3_bind_int(compiledStmt, 3, userID);
```

```
// Execute the update.
if (sqlite3_step(compiledStmt) == SQLITE_DONE) {
    // Update successful.
}
```

Other pitfalls

```
Format string issues from C
```

```
NSString outBuf = @"String to be appended";
outBuf = [outBuf stringByAppendingFormat:[UtilityClass
```

```
formatBuf: unformattedBuff.text]];
```

vs.

```
NSString outBuf = @"String to be appended";
outBuf = [outBuf stringByAppendingFormat:@"%@",[UtilityClass
formatBuf: unformattedBuff.text]];
```

Putting it together - design patterns

- Let's dive into a few patterns
- -Class/whiteboard discussions
- -App scenarios



Stand-alone

App contains some user data

- -Consumer grade
 - Recipes, wine cellar, etc.
- -No networking
- -All data local
- Location data perhapsWhat should we do?What are the issues?



Client-server social net app

- Social network app
- -The real data is on the server side
- -Authentication via app
- Presentation layer/view in app
- What are the issues? -And the solutions?



Client-server financial app

- This one is used for financial information and transactions
- -Stock trading site
- -Mobile payments
- How would we proceed?
- -Issues and security requirements?
- -Special concerns?



Client-server enterprise app

Internal enterprise app

- -Used by employees for some important enterprise purpose
 - Supply chain, customer data, sales, etc.
 - Company's "crown jewels"
- What are the issues?



Kenneth R. van Wyk KRvW Associates, LLC

Ken@KRvW.com http://www.KRvW.com



Copyright© 2012 KRvW Associates, LLC