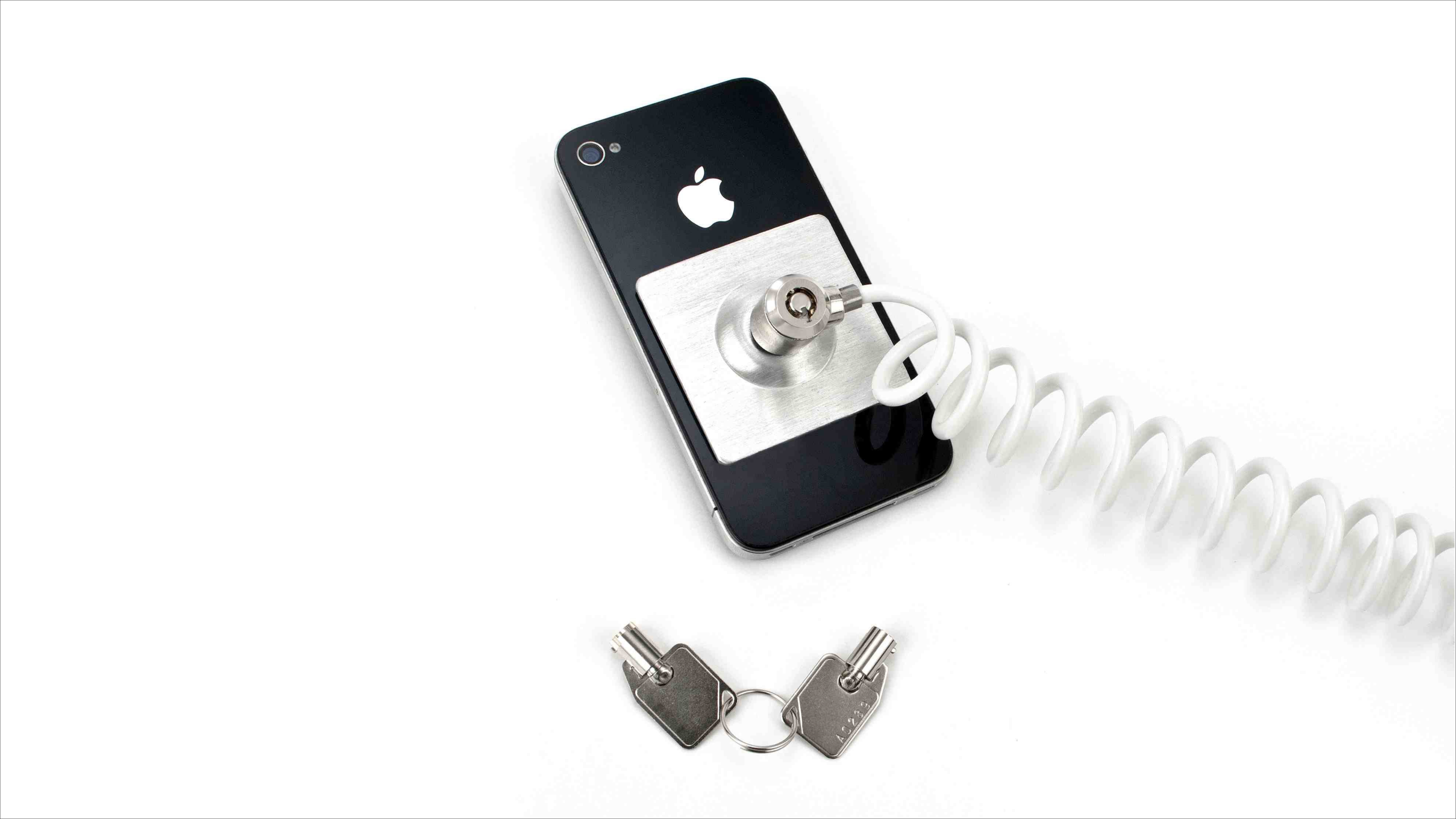
Demystifying Security Best Practices

Conrad Stoll
Architect
@conradstoll - conradstoll.com
mutualmobile

So you want to make software secure?

Well, that’s easy.

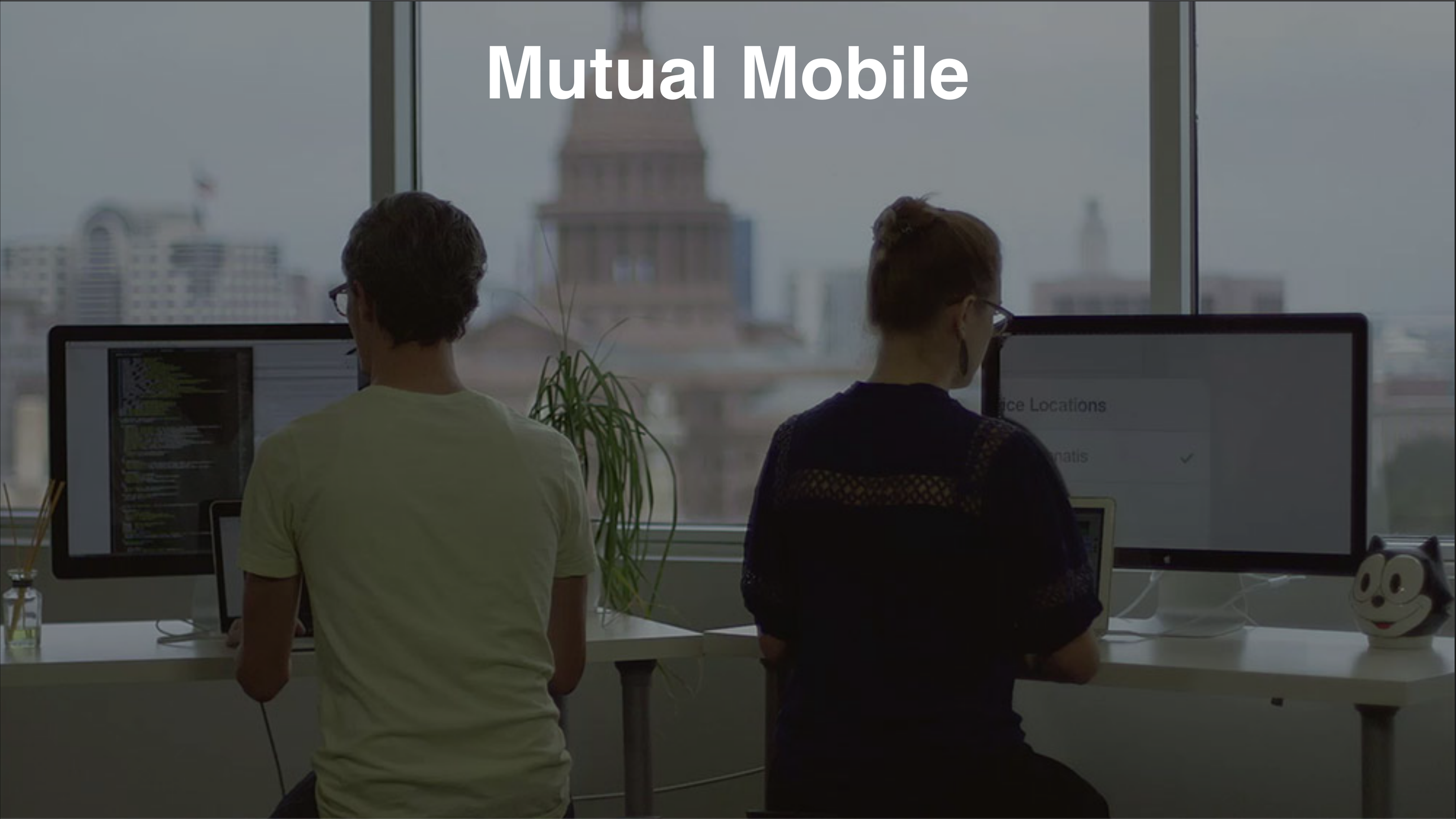


Mobile devices are not secure.

Users assume security.

We are expected to secure our software.

Mutual Mobile



UnionBank®

Cigna

Greenway

ISIS™

Products are judged by the quality of the user experience, and the absence of security issues.

Security is hard, and a lot of apps get it wrong.

goto fail;

Apple actually does take security very seriously.

iOS Security

February 2014

http://images.apple.com/ipad/business/docs/iOS_Security_Feb14.pdf

Security is about being responsible with a user's information.

The type of information your app handles defines how secure it needs to be.

Not everyone is building a banking app.

Security is all about making tradeoffs.

Every app we build has an appropriate balance between security and usability.

Security Best Practices

(Demystified)

14

things you can do to build more secure software

Number 1:
NSLog

But logs are helpful!
What could possibly go wrong?

```
            NSLog(@"Downloading issues...");
        });
    });
}

- (void)loginWithUsername:(NSString *)username password:(NSString *)password {
    NSLog(@"Logging in with username:%@ password:%@", username, password);

    NSURLRequest *request = [[NSURLRequest alloc] initWithURL:[NSURL URLWithString:kLoginURL]];

    [NSURLConnection
     sendAsynchronousRequest:request
     queue:[NSOperationQueue mainQueue]
     completionHandler:^(NSURLResponse *response, NSData *data, NSError *connectionError) {

     }];
}

@end
```

Security


1
Mashable

Dangers of Over Logging

Any good lumberjack knows not to over-log.

Logging provides helpful debug information but can accidentally leak sensitive user data.

- User passwords
- Sensitive web requests
- Locations
- Never know what could be storing those logs (other apps, add ons, etc.).

How to prevent production logs

Think before you log

Use a logging macro and limit the scope of production logs.

- Log only what is required.
- Super sensitive data should not be logged.
- Set different logging levels for release build.

Logging Macro

Using a logging macro to programmatically disable logs

```
#pragma mark - LOGGING

//----  To disable all Logs in release mode ----//
#ifdef DEBUG
#define UBLog( s, ... )  NSLog(s, ## __VA_ARGS__)
#else
#define UBLog( s, ... )
#endif
```

Great case for writing tests

Run tests that verify you have no logs on your build system

We wrote a script to analyze our code and look for ‘NSLog’ in all of our files that would fail the build if it found any.

- Build scripts keep everyone honest.
- Well suited for automation.
- Works great with a Continuous Integration system.

```
ruby file_sieve.rb -d Classes/ -e UnionBankGlobals NSLog
```

Number 2:
Fast App Switching


Favd
Instacast
Tweetbot
Clear
Camera
Runtime
Safari
Music
PayPal
Tweetbot

Hiding Sensitive Data

Hide your kids, hide your text fields

Hiding sensitive information when the app goes to the background can be a good idea to protect user privacy.

- Blur out sensitive data.
- Remove sensitive data.
- Obscure sensitive data.

PayPal does this really well

Hiding Sensitive Data

When the app enters the background

```
#pragma mark - Security

- (void)applicationDidEnterBackground:(UIApplication *)application {
    RootViewController *rootVC = (RootViewController *)self.window.rootViewController;
    UIViewController *viewController = (UIViewController *)[rootVC.presentedViewControllers lastObject];

    /**
     If the last presented view controller is a SplashViewController we must not present the splash
     screen again.
     */
    if (![viewController isKindOfClass:[SplashViewController class]]) {
        /**
         We're presenting a splash screen when the app backgrounds as a privacy and security feature.
         This view needs to block the UI so that a screenshot of sensitive data does not accidentally
         get saved to the background.
         */
        [self presentSplashViewController];
    }
}
```

Number 3:
SSL Pinning

Easy way to ~~prevent MITM~~

^

(make MITM much more difficult)

Compare a server’s certificate with a known certificate

Certificate Pinning

More secure, but fragile

Compare the exact certificate used in the request with the known certificate included in your app bundle.

- Communication will fail after the certificate expires.

Public Key Pinning

Still secure, less fragile

Compare a known public key to the network request's certificate public key.

- Public key's don't change when certificates expire.
- One public key might be used for several web services at a large company.

AFNetworking

Most popular Objective-C library on GitHub!

Built in support for SSL Pinning in 1.0 and 2.0.

- Super easy to use implementation of SSL Pinning.
- AFSecurityPolicy on version 2.0.
- Allows you to select between certificate or public key pinning modes.

Number 4:
ARC PIE

An ARC PIE a day
keeps memory attacks away

Being practical with memory attacks

So, how likely is this really?

In all honesty, protecting against memory attacks probably is not your job as an application developer.

- No, scribbling over NSString isn’t practical.
- Don’t make it easy on an attacker.


1
Mashable

Use ARC

Don’t let objects you aren’t using stick around

ARC helps make sure you’re correctly managing memory.

- Makes sure objects get released and dealloc’d when they should be.

Keep PIE enabled

Who doesn't like PIE?

Position Independent Execution prevents memory address attacks. This option is enabled by default in Xcode.

- Prevents an attack against specific memory addresses.
- Honestly, there's no reason to disable this…

Number 5:
Keychain

Device needs a PIN code

Or a really long passcode

The keychain is more secure if the user sets a passcode on the device

- There’s no way to tell if the user set a passcode.
- Encrypted with the device key, so they don’t transfer to new devices.

Careful what you store in the keychain

Would you put your social security number there?

Best not to store a user's password. Try and store a hash or session token instead.

- Access controls for when data becomes available from the keychain.

Number 6: NSUserDefaults

NSUserDefaults != Keychain

If you wouldn't store it in the keychain, definitely don't store it in NSUserDefaults!

NSUserDefaults is a great place to store basic app preferences or bits of state, like whether or not this is the app's first launch. But it should NOT be used to store anything sensitive.

- Unencrypted key value storage.
- Should never be used to store anything sensitive.


1
Mashable

Number 7:
Text Fields

Sensitive Autocorrect

Be careful what you allow to autocorrect

Autocorrect results are stored unencrypted on the device.

- Autocorrect is disabled by default on secure text fields.
- Disable it on yours if you’re not using the UITextInputTraits secureTextEntry property.
- Consider disabling it anyway if you anticipate sensitive data being entered into a text field or text view.

UIPasteBoard

Do you trust every app with your bank account number?

The pasteboard can be used to share data between apps. A malicious app could look for sensitive data on the pasteboard.

- Avoid storing sensitive data in the pasteboard.
- Disable copying on sensitive text input fields.

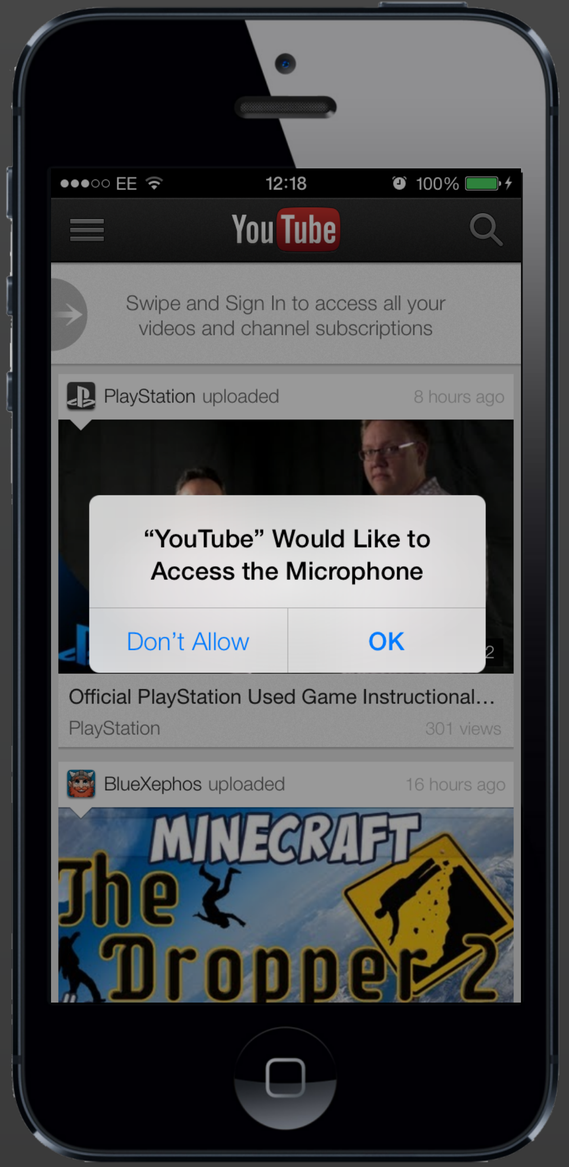
UIPasteBoard

Protect potentially sensitive pasteboard data

Expire sensitive content and mark it as org.nspasteboard.ConcealedType.

- http://nspasteboard.org
- 1Password marks content as “concealed”.
- 1Password removes content after 90 seconds by default.

Number 8:
Asking Permission


●●●○○ EE
12:18
100%
YouTube
Swipe and Sign In to access all your videos and channel subscriptions
PlayStation uploaded
8 hours ago
“YouTube” Would Like to Access the Microphone
Don’t Allow
OK
Official PlayStation Used Game Instructional…
PlayStation
301 views
BlueXephos uploaded
16 hours ago
MINECRAFT
The Dropper 2

iOS- How to avoid the dialog “Would like to use your current location”

0

In my app, one of my input is to get the location of the user as well as the contacts of the user. This is done from the code.

When the user runs the app for the first time,they get a dialog

"AppName" would like to use your current location. I wish to avoid this dialog since this is an important data and dont want the users to accidently press "Dont Allow"

How to avoid this dialog. Could any one please let me know. Thanks

Being polite
when asking for permission

How and when you ask is important

Many users distrust software

Remember that context for your request is very important.

- Don’t ask on app launch or login.
- Ask in response to a user action.
- Only ask for what you clearly need.
- Explain why you need permission.

Treat user permission with respect

Be a good house guest

Be considerate of a user's permission, and don't overstep your bounds.

- Don't copy the user's address book.
- Don't log location data to the server.
- Don't transmit user data in the clear.
- Don't store passwords on your server.
- Don't record all of someone's conversations with their microphone and pick up words like "soda" and attempt to turn around and sell them "Coke".

Permission Resources

Helpful sites talking about permission

- http://techcrunch.com/2014/04/04/the-right-way-to-ask-users-for-ios-permissions/

Number 9:
Local Storage

Implementation Trade Offs

Theoretical versus Practical

Downloading a file using NSURLSessionDownloadTask gives you the ability to download in the background, but removes the ability to encrypt files on the fly.

- Is it ok to have a file stored in the clear for 10 seconds?
- What if you know the user is in physical possession of the device?
- Would this concern prevent you from supporting “open in” or “open with” features?

Good things to store

Pretty much zero risk of storing this kind of stuff

Storing data locally can help improve the user experience. Here's some things you can feel pretty safe about storing.

- Most user-generated photography
- Most user-generated content (todo list)
- Publicly available content
- Podcasts, music, videos
- Application metadata database
- Application preferences

Bad things to store

Storing this stuff should keep you up at night

Some apps deal with sensitive data, and storing this on the device can be dangerous.

- Personal medical information
- Personal financial information
- Copyrighted content
- Secret corporate information

In-Memory Core Data

Core Data, but without the storage part

Sometimes you want to use Core Data even though you can’t store anything.

- In-Memory store is pretty fast
- Great for facilitating object graph management and fetching/sorting
- Easy way to use Core Data if persistence isn’t an option or a requirement

Number 10:
Encryption

File Encryption

AES all the things!

Use the latest file encryption provided by common crypto.

- AES 256
- User generated encryption key

Pass the salt.

Data Protection

Apple wants to make this easy for you

Data Protection APIs provide an easy to use baseline protection for files on disk.

- Available in iOS 4.0.
- Enabled by default now in iOS 7.0.
- Protected by the user passcode and device key.
- Makes it harder to access files, but can be defeated with physical access to the device.
- Far more secure if the user sets a passcode.

Encrypted Core Data

Core Data + Incremental Store + Encryption. What could possibly go wrong?

Sometimes you really need an encrypted database. When you do, there's NSIncrementalStore + Encryption.

- Implement your own encrypted Core Data persistent store.
- Several existing implementations are available.
- Core Data is already a complex system. Encrypted Core Data is obviously even more complex.
- Consider using a file based encryption system for small bits of data.

Encryption Resources

Helpful sites talking about encryption

- http://www.stoeger-it.de/en/secureincrementalstore/
- https://github.com/project-imas/encrypted-core-data
- http://adcdownload.apple.com//videos/wwdc_2012__sd/session_714__protecting_the_users_data.mov
- http://blog.agilebits.com/2013/03/09/guess-why-were-moving-to-256-bit-aes-keys/

Number 11:
UIWebView

Whitelist domains in web views

Support safe browsing

Prevent local network spoofing attacks by only allowing UIWebView requests to specific domains.

- Compare requests made by your web view to a specific list of domains.
- Enforce HTTPS on all requests from the web view.

Whitelisting Domains

Teaching someone how not to phish

```
#pragma mark - Security

/*
 Only allow the web view to load requests to whitelisted domains in order to prevent
 phishing attacks.
 */
- (BOOL)webView:(UIWebView *)webView
        shouldStartLoadWithRequest:(NSURLRequest *)request
        navigationType:(UIWebViewNavigationType)navigationType {
    if ([self shouldAllowWebViewNavigationToURLRequest:request]) {
        return YES;
    }

    [self presentUnsafeBrowsingWarningForRequest:request];

    return NO;
}
```

No SSL Pinning Support

UIWebView doesn't benefit from SSL Pinning elsewhere in your app

Web views don't get challenge callbacks, so it is hard to implement SSL Pinning with them (though not impossible).

- Try not to use web views for anything super sensitive.
- If you're relying on a web view for most of your app, its worth going to the trouble of bootstrapping SSL Pinning for it.

WebView Resources

Helpful sites talking about web views

- https://www.isecpartners.com/blog/2013/february/ssl-pinning-on-ios.aspx
- http://stackoverflow.com/questions/11573164/uiwebview-to-view-self-signed-websites-no-private-api-not-nsurlconnection-i

Number 12:
Third Party Libraries

Hope is NOT a strategy

Third Party Library Security

Hope is not a strategy. Make sure libraries are secure.

You're responsible for what third party libraries are doing in your app. If you're following the rules above, make sure they are too.

- Check and see if they are writing anything to disk.
- Look at their networking code to see what they're sending over the air.
- Check on their implementation of debug logging and whether or not it can be disabled.

Number 13:
Enterprise MDM

If you’re building an enterprise app…

Use MDM Features

There’s lots of powerful security features in MDM. If you’re building an enterprise app, make sure your client knows about this!

- Force device passcode
- Per-app VPN
- Configuration enforcement
- Encrypted backups
- Remote wipe

Number 14:
Have a Backup Plan

If all else fails…

The last one standing blows the bridge.

You can always use a remote server to force an update. You can lock out the app in the event of a security breach, and force users to install the update that fixes it.

- Gives you a pretty heavy hammer.
- Easy to implement.
- Important for high-risk apps like banking.

Thanks!!

Secure all the things!

Conrad Stoll

@conradstoll - conradstoll.com

mutualmobile


SECURE
ALL THE THINGS!