nccgroup
freedom from doubt

Exploring SecAccessControl Obj in iOS 9 Keychain

Manipulating iOS keychain just got easier

nccgroup
freedom from doubt

Myself

- Senior Security Consultant @NCC Group
  - Nitin.jami@nccgroup.trust
- Specialize in Mobile Security
- Like to understand OS internals
- Currently working OS X / iOS
- Author of KeychainEditor and KeychainEditrUI
- First Time Speaker!!!

nccgroup
freedom from doubt

Outline

- Introduction to iOS Keychain
- Research Goals
- Evolution of iOS Keychain
- Demystifying SecAccessControl Obj
- Introducing keychaineditor & keychainEditrUI
- Future Work

nccgroup
freedom from doubt

Outline

- **Introduction to iOS Keychain**
- Research Goals
- Evolution of iOS Keychain
- Demystifying SecAccessControl Obj
- Introducing keychaineditor & keychainEditrUI
- Future Work

nccgroup
freedom from doubt

iOS System Keychain

- Secure storage for storing secrets, certificates onto iDevice
- Single keychain stores items for all apps
- Maintained as a SQLite database
  - /private/var/Keychains/keychain-2.db

item	Description
genp	**Generic Passwords** -- **kSecClassGenericPassword**
inet	Internet Passwords -- kSecClassInternetPassword
Cert and Keys	**Certificates, Keys and Digital Identities (cert + keys) items** -- **kSecClassCertificates and kSecClassIdentity**

Data Protection API

- Simpler API than OS X

- Keychain entries are protected with certain class keys -- **Data Protection**
  - Defined by `kSecAttrAccessible`

```
extern const CFStringRef kSecAttrAccessibleWhenUnlocked
    __OSX_AVAILABLE_STARTING(__MAC_10_9, __IPHONE_4_0);
extern const CFStringRef kSecAttrAccessibleAfterFirstUnlock
    __OSX_AVAILABLE_STARTING(__MAC_10_9, __IPHONE_4_0);
extern const CFStringRef kSecAttrAccessibleAlways
    __OSX_AVAILABLE_STARTING(__MAC_10_9, __IPHONE_4_0);
extern const CFStringRef kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly
    __OSX_AVAILABLE_STARTING(__MAC_10_10, __IPHONE_8_0);
extern const CFStringRef kSecAttrAccessibleWhenUnlockedThisDeviceOnly
    __OSX_AVAILABLE_STARTING(__MAC_10_9, __IPHONE_4_0);
extern const CFStringRef kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly
    __OSX_AVAILABLE_STARTING(__MAC_10_9, __IPHONE_4_0);
extern const CFStringRef kSecAttrAccessibleAlwaysThisDeviceOnly
    __OSX_AVAILABLE_STARTING(__MAC_10_9, __IPHONE_4_0);
```

Apple open source. SecItem.h

Keychain -- Some Considerations

- Use the highest data protection level you can
  - `kSecAttrAccessibleWhenUnlocked` default and best
  - `kSecAttrAccessibleAfterFirstUnlock` for background apps
  - `kSecAttrAccessibleAlways` will be deprecated in iOS 9

Source: Apple WWDC 2015, Security and Your Apps

Keychain -- Some Considerations

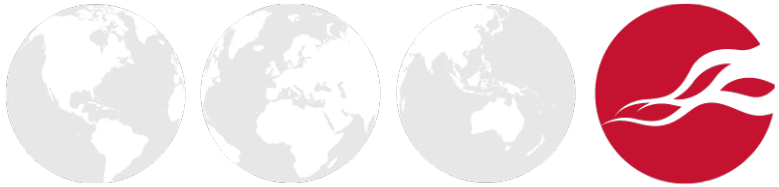
~~`kSecAttrAccessibleAlways`~~ ~~will be deprecated in iOS 9~~

iOS 9.0 API Diffs

Modified `kSecAttrAccessibleAlways`

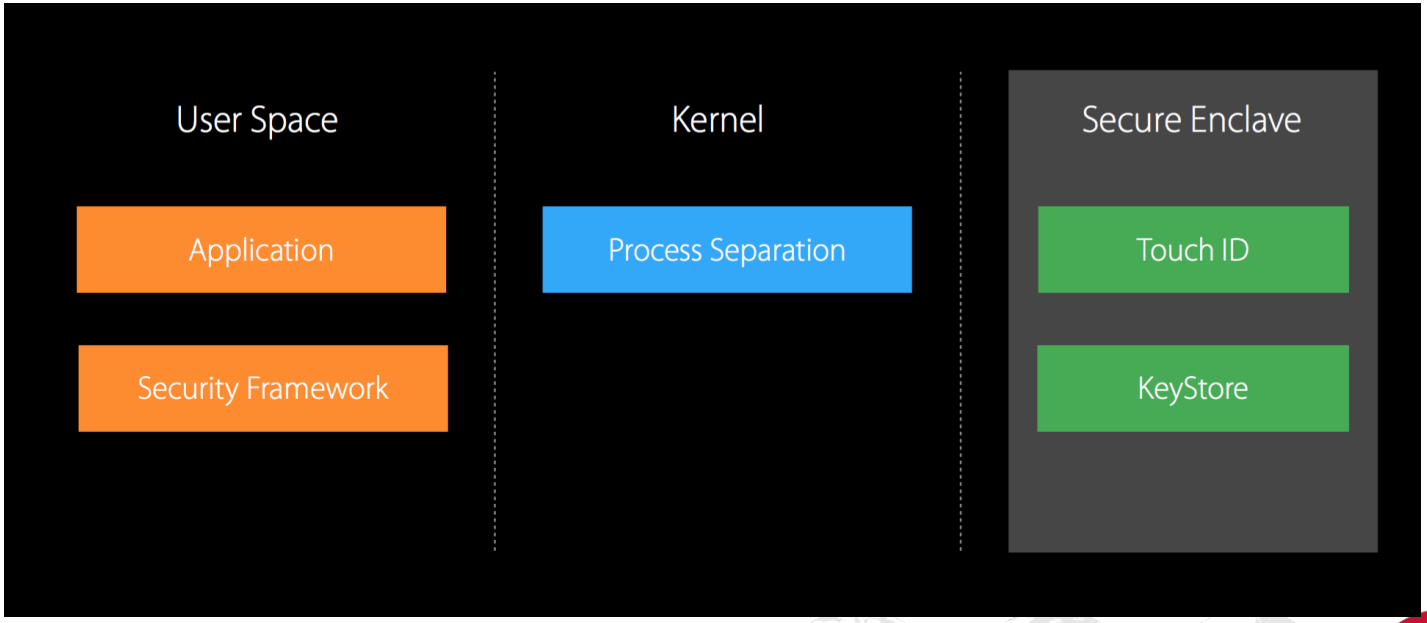
	Declaration
From	`var kSecAttrAccessibleAlways: CFStringRef`
To	`let kSecAttrAccessibleAlways: CFString`

https://developer.apple.com/library/ios/releasenotes/General/iOS90APIDiffs/Swift/Security.html



Architecture

- iOS security domains



Source: Apple WWDC 2015, Security and Your Apps

Using the Keychain

nccgroup
freedom from doubt

- Available APIs
  - SecItemAdd
  - SecItemUpdate
  - SecItemCopyMatching
  - SecItemDelete
- Undocumented C functions
  - SecAccessControlGetProtection
  - SecAccessControlGetConstraints

SecItem.c
https://opensource.apple.com/source/Security/Security-57031.1.35/Security/sec/Security/SecItem.c

Keychain Attributes

nccgroup
freedom from doubt

- Account Name
- Service Name
- Access Group
  - <keychain-access-group>
- Protections (Accessibility)

nccgroup
freedom from doubt

Outline

- Introduction to iOS Keychain
- **Research Goals**
- Evolution of iOS Keychain
- Demystifying SecAccessControl Obj
- Introducing keychaineditor & keychainEditrUI
- Future Work

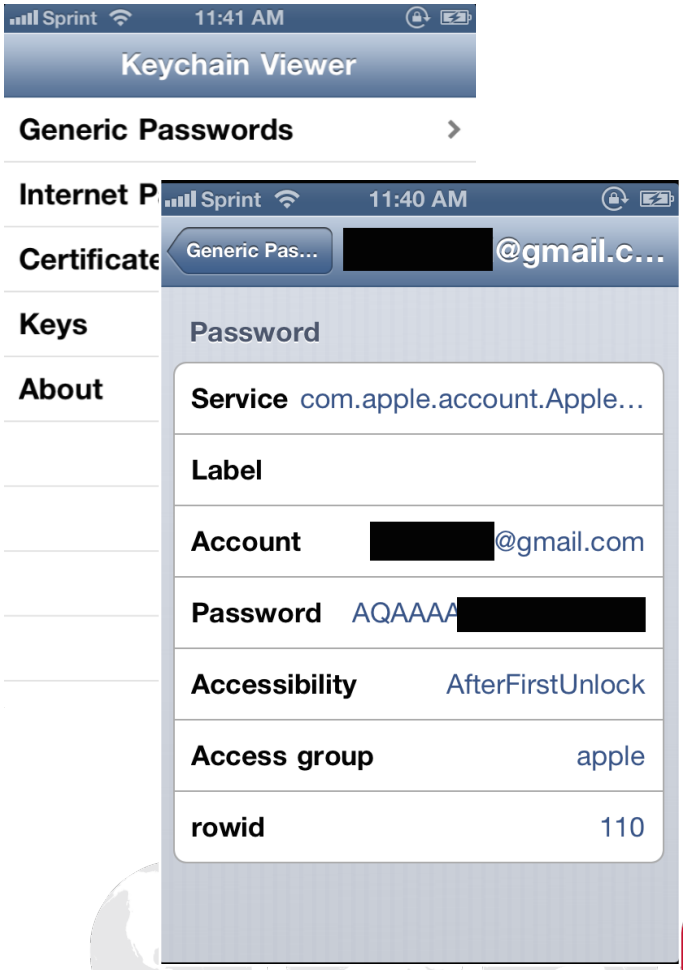
Research Goals

nccgroup
freedom from doubt

- Keychain is the recommended storage area, but…
- iOS 8 started to break a lot of tools
- iOS 8 brought changes
  - iOS 9 brought even more changes
- Keychain is sensitive
  - Let's hack it...

Previous Work

- Access all keychain items
  - Sign entitlement file with a wildcard
- Available Tools
  - Keychain_dumper – command line
  - Keychainviewer – graphical app



Source: 44 Con, NCC Workshop

WildCard Access-Group

nccgroup
freedom from doubt

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
3  <plist version="1.0">
4  <dict>
5      <key>keychain-access-groups</key>
6      <array>
7          <string>*</string>
8      </array>
9  </dict>
10 </plist>
11
```

Motivation

nccgroup
freedom from doubt

- Existing tools do not work as expected on newer iOS versions.
- Lack of support for iOS 8+ changes
- No means to edit or delete a keychain item
- More importantly, no search through the massive dump

```
switch (unwrappedValue as! String) {
    case "ak": return "kSecAttrAccessibleWhenUnlocked"
    case "ck": return "kSecAttrAccessibleAfterFirstUnlock"
    case "dk": return "kSecAttrAccessibleAlways"
    case "aku": return "kSecAttrAccessibleWhenUnlockedThisDeviceOnly"
    case "cku": return "kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly"
    case "dku": return "kSecAttrAccessibleAlwaysThisDeviceOnly"
    case "akpu": return "kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly"
    case "": return "NIL"
    default: return (unwrappedValue as! String)
}
```

nccgroup
freedom from doubt

Outline

- Introduction to iOS Keychain
- Research Goals
- **Evolution of iOS Keychain**
- Demystifying SecAccessControl Obj
- Introducing keychaineditor & keychainEditrUI
- Future Work

nccgroup
freedom from doubt

Multi-Factor Authentication

- Device Passcode
- TouchID

- Used By
  - Keychain APIs
  - LocalAuthentication API

nccgroup
freedom from doubt

Changes in iOS 8+

- New Accessibility Constant
  - `kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly`
- New Attribute
  - `kSecAttrAccessControl`
  - Provides fine-grained access control
  - Only one policy supported for now

`.UserPresence`

Changes in iOS 9+

- More Policies
  - `UserPresence`
  - `TouchIDAny`
  - `TouchIDCurrentSet`
  - `DevicePasscode`
  - `Or`
  - `And`
  - `PrivateKeyUsage`
  - `ApplicationPassword`

```
49 typedef CF_OPTIONS(CFIndex, SecAccessControlCreateFlags) {
50     kSecAccessControlUserPresence           = 1 << 0,
51     kSecAccessControlTouchIDAny             CF_ENUM_AVAILABLE(NA, 9_0)     = 1 << 1,
52     kSecAccessControlTouchIDCurrentSet      CF_ENUM_AVAILABLE(NA, 9_0)     = 1 << 3,
53     kSecAccessControlDevicePasscode         CF_ENUM_AVAILABLE(10_11, 9_0) = 1 << 4,
54     kSecAccessControlOr                     CF_ENUM_AVAILABLE(NA, 9_0)     = 1 << 14,
55     kSecAccessControlAnd                    CF_ENUM_AVAILABLE(NA, 9_0)     = 1 << 15,
56     kSecAccessControlPrivateKeyUsage        CF_ENUM_AVAILABLE(NA, 9_0)     = 1 << 30,
57     kSecAccessControlApplicationPassword    CF_ENUM_AVAILABLE(NA, 9_0)     = 1 << 31,
58 } __OSX_AVAILABLE_STARTING(__MAC_10_10, __IPHONE_8_0);
59
```

nccgroup
freedom from doubt

Outline

- Introduction to iOS Keychain
- Research Goals
- Evolution of iOS Keychain
- **Demystifying SecAccessControl Obj**
- Introducing keychaineditor & keychainEditrUI
- Future Work

Demystifying SecAccessControl Obj

nccgroup
freedom from doubt

```
if let acl = SecAccessControlCreateWithFlags(kCFAllocatorDefault,
    kSecAttrAccessibleWhenUnlocked, [.UserPresence], &error) {

        let query = [
            kSecClass as String          :   kSecClassGenericPassword as String,
            kSecAttrAccount as String    :   account,
            kSecAttrService as String    :   service,
            //kSecAttrAccessible as String:   accessibleConstant,
            // Uncomment the following line to add AccessControl. Make sure
            // "acl" is defined above in the if let scope.
            kSecAttrAccessControl as String :   acl,
            kSecValueData as String      :   data
            ] as NSDictionary

        status = SecItemAdd(query, nil)

        if status != errSecSuccess {
            NSLog("[addItem::SecItemAdd] \(osstatusToHumanReadable(status))")
        }
} else {
    NSLog("[addItem::SecAccessControl] - \(osstatusToHumanReadable(status))")
}
```

Demystifying SecAccessControl Obj

nccgroup
freedom from doubt

```
if let acl = SecAccessControlCreateWithFlags(kCFAllocatorDefault,
    kSecAttrAccessibleWhenUnlocked, [.UserPresence], &error) {
```

- Return value of `SecAccessControlCreateWithFlags()`
  - `SecAccessControl?`
- What is SecAccessControl?

```
struct __SecAccessControl {
    CFRuntimeBase _base;
    CFMutableDictionaryRef dict;
};
```

Let's build that Dictionary

nccgroup
freedom from doubt

```
if let acl = SecAccessControlCreateWithFlags(kCFAllocatorDefault,
    kSecAttrAccessibleWhenUnlocked, [.UserPresence], &error) {
```

- First call is made to `SecAccessControlCreate()`
- Second call is made to `SecAccessControlSetProtection()`
- Accessing the dictionary, `__SecAccessControl -> dict`

```
{
  prot = ak;
}
```

```
(lldb) po $rdi
{
    prot = ak;
}
```

Let's build that Dictionary

nccgroup
freedom from doubt

```
if let acl = SecAccessControlCreateWithFlags(kCFAllocatorDefault,
    kSecAttrAccessibleWhenUnlocked, [.UserPresence], &error) {
```

- Next step is to create a sub-dictionary for the actual access control policies
  - `const kAKSKeyAcl = "acl"`
- A call is made to `SecAccessControlGetMutableConstraints()`
- Accessing the dictionary, `__SecAccessControl -> dict`

```
{
    acl =     {
    };
    prot = ak;
}
```

```
(lldb) po 0x00007f8041d074e0
{
    acl =     {
    };
    prot = ak;
}
```

Let's build that Dictionary

nccgroup
freedom from doubt

```
if let acl = SecAccessControlCreateWithFlags(kCFAllocatorDefault,
    kSecAttrAccessibleWhenUnlocked, [.UserPresence], &error) {
```

- Flags (Policies) determine the overall structure of `kAKSKeyAcl` dictionary
- If no policies are specified, default operation is registered.
  - `kAKSKeyOpDefaultAcl = kCFBooleanTrue`
- Accessing the dictionary, `__SecAccessControl -> dict`

```
{
    acl =   {
        dacl = 1;
    };
    prot = ak;
}
```

```
(lldb) po 0x00007f8041d074e0
{
    acl =     {
        dacl = 1;
    };
    prot = ak;
}
```

Let's build that Dictionary

nccgroup
freedom from doubt

```
if let acl = SecAccessControlCreateWithFlags(kCFAllocatorDefault,
    kSecAttrAccessibleWhenUnlocked, [.UserPresence], &error) {
```

- If `.UserPresence` policy is specified
- We get three in-built "operations" set for this item.
- Accessing the dictionary, `__SecAccessControl -> dict`

```
{
        acl =     {
                od =         {
                        cpo = DeviceOwnerAuthentication;
                };
                odel = 1;
                oe = 1;
        };
        prot = ak;
}
```

Let's take a step back

nccgroup
freedom from doubt

- What are these operations
  - `kAKSKeyOpDefaultAcl = "dacl"`
  - `kAKSKeyOpEncrypt = "oe"`
  - `kAKSKeyOpDecrypt = "od"`
  - `kAKSKeyOpDelete = "odel"`
  - `kAKSKeyOpSign = "osgn"`
  - `kAKSKeyOpSync` (Never Used)

Let's take a step back

nccgroup
freedom from doubt

- Available Constraints

```
• .UserPresence = "cpo" // DevicePasscode + TouchIDAny
    {
        cpo = DeviceOwnerAuthentication;
    }

• .DevicePasscode = "cup"
• .TouchIDAny = "cbio"
    {
        cbio = {pbioc = }
    }

• .TouchIDCurrentSet = "cbio"
    {
        cbio = {
            pbioc = // biometric constant
            pbioh = // hash of the current set of enrolled fingers.
        }
    }
```

Let’s take a step back

nccgroup
freedom from doubt

- Not Really a Constraint, more like an operation
  - `.ApplicationPassword`
    - `kAKSKeyAclParamRequirePasscode = “prp”`

Operations & Constraints

nccgroup
freedom from doubt

- **Regular Operation**
  - `kAKSKeyOpDecrypt`
    - `"cpo", "cup", "cbio"`
  - `kAKSKeyOpEncrypt`
    - `True`
  - `kAKSKeyOpDelete`
    - `True`
- **.PrivateKeyUsage**
  - `kAKSKeyOpSign`
    - `"cpo", "cup", "cbio"`
  - `kAKSKeyOpDelete`
    - `True`
- **.ApplicationPassword**
  - `kAKSKeyAclParamRequirePasscode`
    - `True Or False`

Undocumented APIs

nccgroup
freedom from doubt

```
let prot = SecAccessControlGetProtection(acl as! SecAccessControlRef).takeUnretainedValue() as! String
```

```
let constr = SecAccessControlGetConstraints(acl as! SecAccessControlRef).takeUnretainedValue() as Dictionary
```

```
{
        od =        {
                cpo = DeviceOwnerAuthentication;
        };
        odel = 1;
        oe = 1;
}
```

nccgroup
freedom from doubt

Outline

- Introduction to iOS Keychain
- Research Goals
- Evolution of iOS Keychain
- Demystifying SecAccessControl Obj
- **Introducing keychaineditor & keychainEditrUI**
- Future Work

~~Introducing~~ keychaineditor (CLI)

nccgroup
freedom from doubt

- Provides dump / edit / delete on a keychain item
  - `https://github.com/NitinJami/keychaineditor`
- Supports iOS 8+
- Integrates with idb (http://www.idbtool.com/)

Usage

nccgroup
freedom from doubt

```
Nccs-iPhone:~ root# ./keychaineditor --help

Usage: ./keychaineditor commands
Commands can be:
        --help:    Prints the usage.
        --action:  Can be either min-dump, dump, edit, delete.
        --find:    uses 'CONTAINS' to find strings from the dump.
        --account: Account name for the keychain item you want to edit/delete.
        --service: Service name for the keychain item you want to edit/delete.
        --agroup:  Optional. Access group for the keychain item you want to edit/delete.
        --data:    Base64 encoded data that is used to update the keychain item.
Account and service is used to uniquely identify a keychain item.
Note: If there is no account name, pass a '' string.
Note: --find is an optional command for dump. It search from
{Account, Service, EntitlementGroup, Protection}.
Examples:
        ./keychaineditor --action dump --find XXX
        Or
        ./keychaineditor --action delete --account XXX --service XXX
        Or
        ./keychaineditor --action edit --account XXX --service XXX --data XXX
```

min-dump

nccgroup
freedom from doubt

```
Nccs-iPhone:~ root# ./keychaineditor --action min-dump
Warning: The names are truncated to max width of 35 charaters. Please use this dump as a reference, and use --find to get ful
l details.
Account                             | Service
----------------------------------- | -----------------------------------
Identity Root                         BluetoothGlobal
Encryption Root                       BluetoothGlobal
identity-rsa-private-key              ids
identity-rsa-key-pair-signature-v1    ids
Byakugou                              AirPort
Group-Lab                             AirPort
kEntitlementDeviceInfoCacheKey        CommCenter
registrationV1                        com.apple.facetime
                                      push.apple.com
,ffffff81ffffffeaffffff80ffffffa8ff   push.apple.com,PerAppToken.v0
                                      AppleIDClientIdentifier
```

dump comes with Search

nccgroup
freedom from doubt

- Search is performed by doing a “CONTAINS” operator
- Currently supported fields
  - Account Name
  - Service Name
  - Access Group
  - Accessibility Constants
- To Support
  - Timestamp
- Dump will tell you if a `.UserPresence` is required or not

```
Nccs-iPhone:~ root# ./keychaineditor --action dump --find whenunlocked
{
  "1" : {
    "Protection" : "kSecAttrAccessibleWhenUnlocked",
    "Account" : "default",
    "EntitlementGroup" : "apple",
    "Creation Time" : "2015-06-29 15:54:19 +0000",
    "Service" : "com.apple.mobileipod.homeSharingPassword",
    "Modified Time" : "2015-06-29 15:54:19 +0000",
    "Data" : "TmNjQDIwMTU=",
    "UserPresence" : "No"
  },
  "2" : {
    "Protection" : "kSecAttrAccessibleWhenUnlockedThisDeviceOnly",
    "Account" : "Private",
    "EntitlementGroup" : "apple",
    "Creation Time" : "2015-06-29 15:55:41 +0000",
    "Service" : "com.apple.managedconfiguration",
    "Modified Time" : "2015-06-29 17:14:56 +0000",
    "Data" : "YnBsaXN0MDDRAQJXaGlzdG9yeaED0wQFBgcICVRoYXNoVHNhbHRaaGFzaE1
cISYxSE0AAAAAAAABAQAAAAAAAAAKAAAAAAAAAAAAAAAAAAAATw==",
    "UserPresence" : "No"
  },
  "3" : {
    "Protection" : "kSecAttrAccessibleWhenUnlockedThisDeviceOnly",
    "Account" : "message-protection-class-a-key",
    "EntitlementGroup" : "ichat",
    "Creation Time" : "2015-06-29 16:05:22 +0000",
    "Service" : "ids",
    "Modified Time" : "2015-06-29 16:05:23 +0000",
    "Data" : "AgAMa2V5cwAAAAAAAAAGAEEEiNoAZX8SRWEXX2jUfHI3cOX7UWLBFos2SwY
Ma2V5cwAAAAAAAAAHAKwwgakCgaEAwyszgzlSYGPKHu8E4JRXraw0bQIjVeLA3IPLmHh58mZc
OenlWyvbJiqkQP8khEo\/9MNtJXfZNrfeH5a3LbGvjQwBpoZLFHHo\/jPaBBkPko571cH+4S2
AQAB",
    "UserPresence" : "No"
  }
}
```

edit

nccgroup
freedom from doubt

- Sometimes you want to edit an item during testing
- Uniquely identify an item
  - `{AccountName, ServiceName, AccessGroup}`
  - `AccessGroup` recommended, but not required
- Currently only accepts base64 encoded data
- **`$ keychaineditor --action edit --account <account-name> --service <service-name> --agroup <access-group> --data <b64-encoded-data>`**

delete

nccgroup
freedom from doubt

- Items are not delete when a app gets uninstalled
- Uniquely identify an item
  - `{AccountName, ServiceName, AccessGroup}`
  - `AccessGroup` recommended, but not required

```
$ keychaineditor –action delete --account <account-name> --service <service-name> --agroup <access-group>
```

Integrates with idb

nccgroup
freedom from doubt

5	5	apple	Byakugou	AirPort
6	6	apple	Group-Lab	AirPort

Dump Keychain

Data | Hexdump | View Plist

#purpl3Panther

Delete | Edit as Text | Edit as Base64

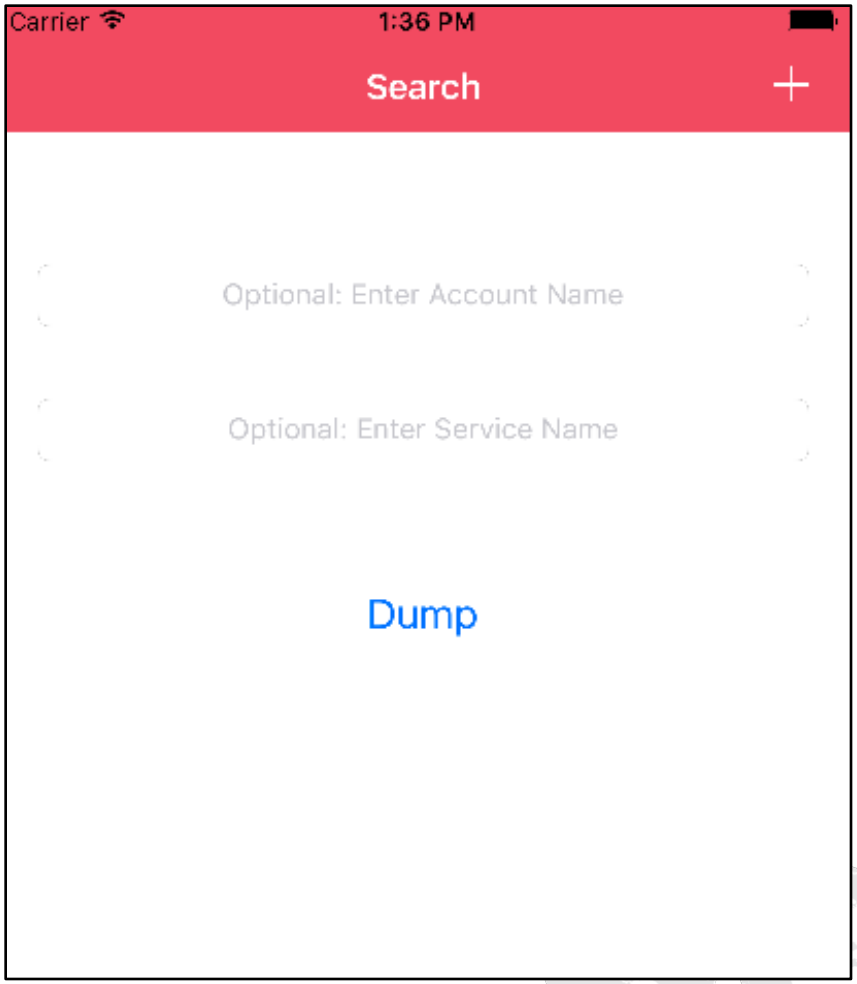
Introducing KeychainEdtorUI

nccgroup
freedom from doubt

- Same features as keychaineditor (CLI)
  - https://github.com/NitinJami/keychainEditrUI
- I just wanted to write in Swift

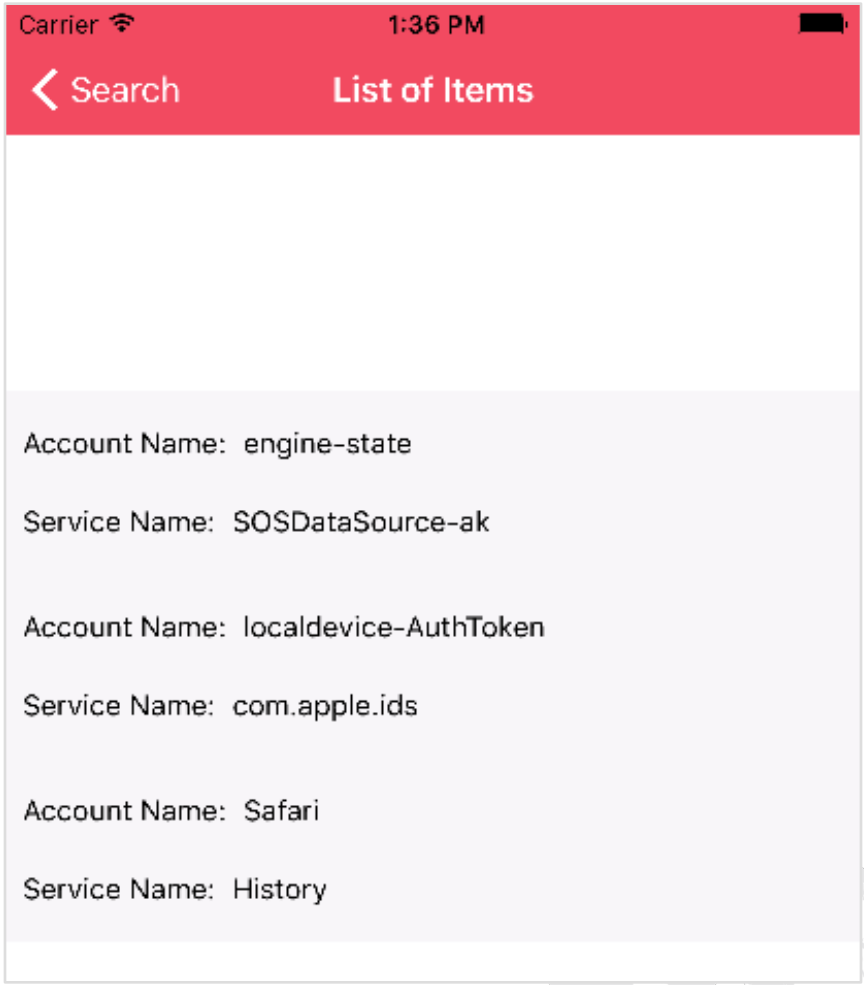
KeychainEditrUI

nccgroup
freedom from doubt



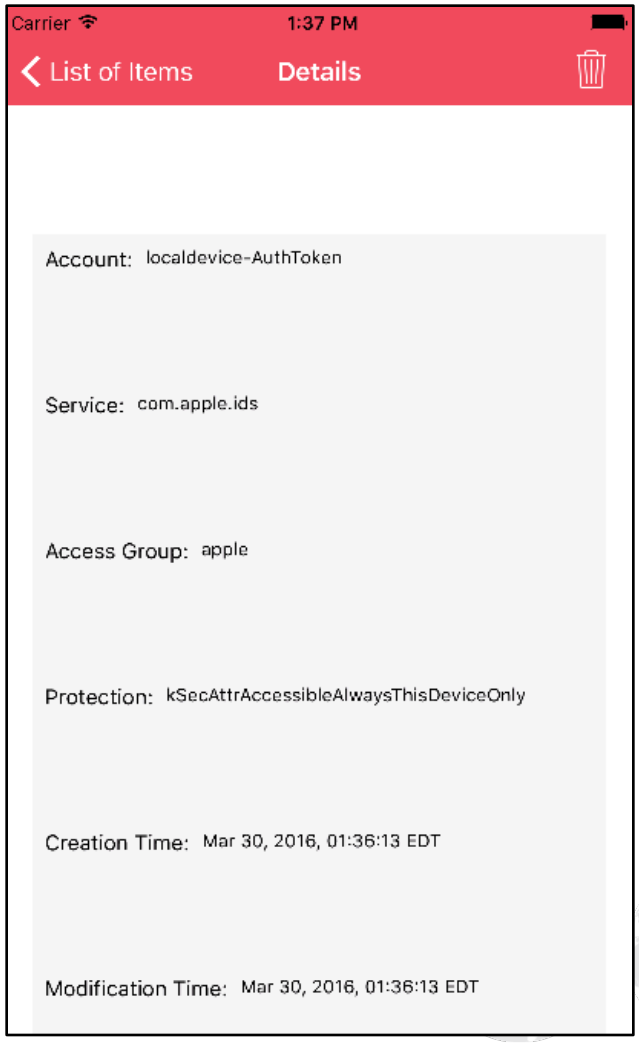
KeychainEditrUI

nccgroup
freedom from doubt



KeychainEditrUI

nccgroup
freedom from doubt



nccgroup
freedom from doubt

Outline

- Introduction to iOS Keychain
- Research Goals
- Evolution of iOS Keychain
- Demystifying SecAccessControl Obj
- Introducing keychaineditor & keychainEditrUI
- **Future Work**

Future Work

nccgroup
freedom from doubt

- Obviously, lot of work remains…
- Support for Internet Passwords and Certificates
- iCloud Web Sharing
- Deep dive into iOS Keychain at kernel level
  - Interested!, ping me.
- For issues or feature requests:
  - https://github.com/NitinJami/keychaineditor/issues
- Happy pen-testing!

nccgroup
freedom from doubt

North America

Atlanta

Austin

Chicago

New York

San Francisco

Seattle

Sunnyvale

Europe

Manchester - Head Office

Amsterdam

Cheltenham

Copenhagen

Edinburgh

Glasgow

Leatherhead

London

Luxembourg

Milton Keynes

Munich

Zurich

Australia

Sydney