Analysis Results

DVIA-v2-master.zip

Report Date
2024-09-10 14:21:53
Report Author
admin
Classification Method
OWASP MASVS L1
Product Version
11.0-SNAPSHOT+f428841a
Rules Version
11-SNAPSHOT.130321

Confidentiality Note

This report is intended only for the person(s) or entity to which it is addressed and contains confidential and privileged information. If you are not the intended recipient, you must not view, use, copy, disclose, or otherwise disseminate this report or any part of it. Doing so is strictly prohibited, and may result in legal proceedings. If you received this in error, please notify the sender immediately and destroy any copies of this information.

01 Project Information

Project Name

DVIA-v2-master.zip

UUID

2e312ca0-6e81-4a21-966f-abee5b5de6bf

Project in DerScanner

Dynamics by vulnerabilities

Vulnerabilities are divided by severity level: critical, medium, low and info.

CRITICAL LEVEL	MEDIUM LEVEL	LOW LEVEL	INFORMATION
Likely to lead to compromise confidential data and violation of the integrity of the system.	May be less likely to lead to compromising confidential data and violating the integrity of the system, or are less serious security	Can become a potential security risk.	Signal a violation of good programming practice.

First of all, pay attention to vulnerabilities of critical and medium levels.

Dynamics by rating

The app score is calculated on a scale from 0 to 5. Score is calculated based on the number of critical and medium level vulnerabilities. The impact of critical vulnerabilities is greater than that of medium level vulnerabilities, and does not take into account the amount of code. Medium level vulnerabilities are taken into account based on their frequency and total number of source code lines.

Scan History

	Date and Time	Status	Languages	Lines of Code	Number of Vulnerabilities					Score
					Critical	Medium	Low	Info	Total	
1/1	2024-09-04 10:09:22	completed	Config files, Objective-C, C/C++, Swift	581 879	17	829	62	3	911	1.1/5.0

02

Scan Information

1/1 2024-09-04 10:09:22
11-SNAPSHOT.130256

Scan Statistics

Status
completed

Score
1.1/5.0

Duration
0:47:13

Lines of Code
581 879

Vulnerabilities

17	829	62	3
Critical	Medium	Low	Info

911
Total

Language	Status	Duration	Lines of Code	Number of Vulnerabilities				
				Critical	Medium	Low	Info	Total
Config files	completed	0:00:29	332 847	8	31	0	0	39
Objective-C	completed	0:44:45	122 146	6	759	5	0	770
C/C++	completed	0:01:46	122 136	0	9	45	0	54
Swift	completed	0:00:13	4 750	3	30	12	3	48

Language Statistics

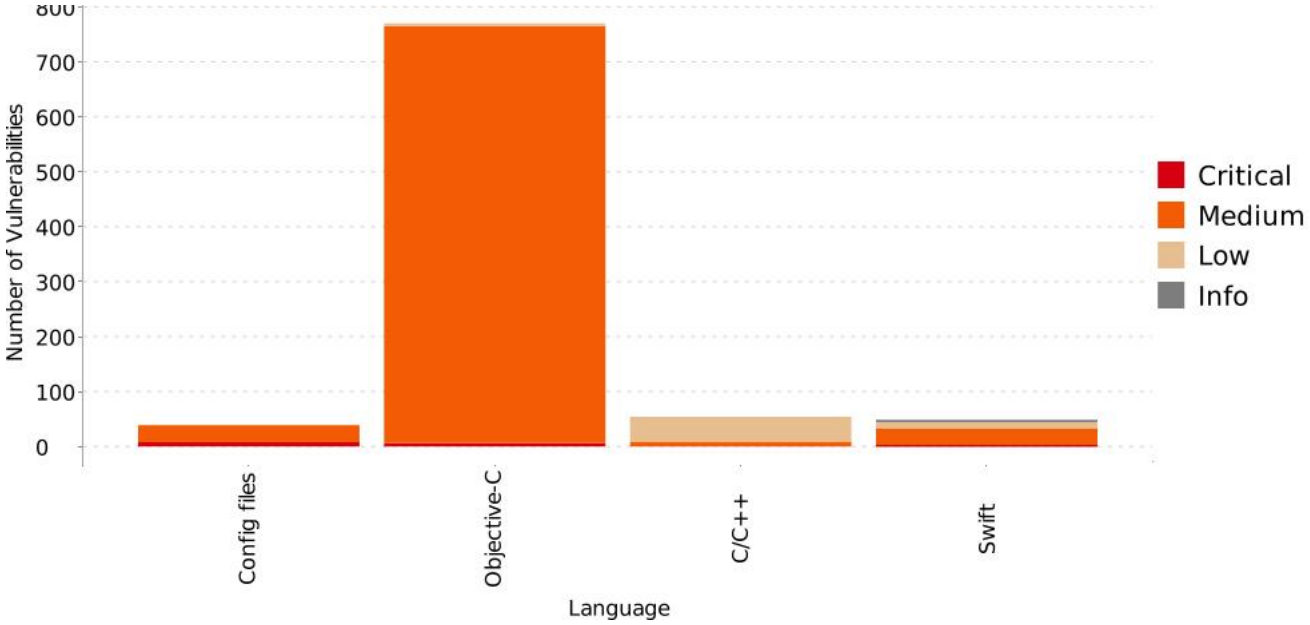
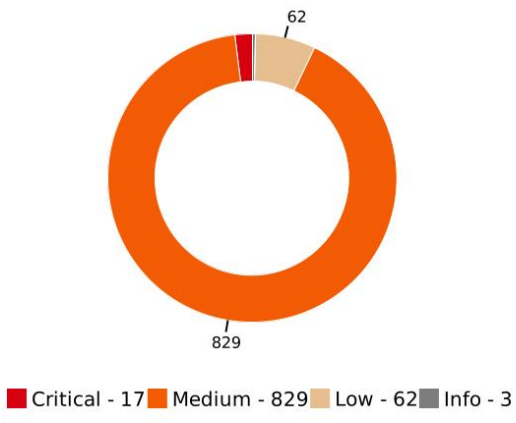
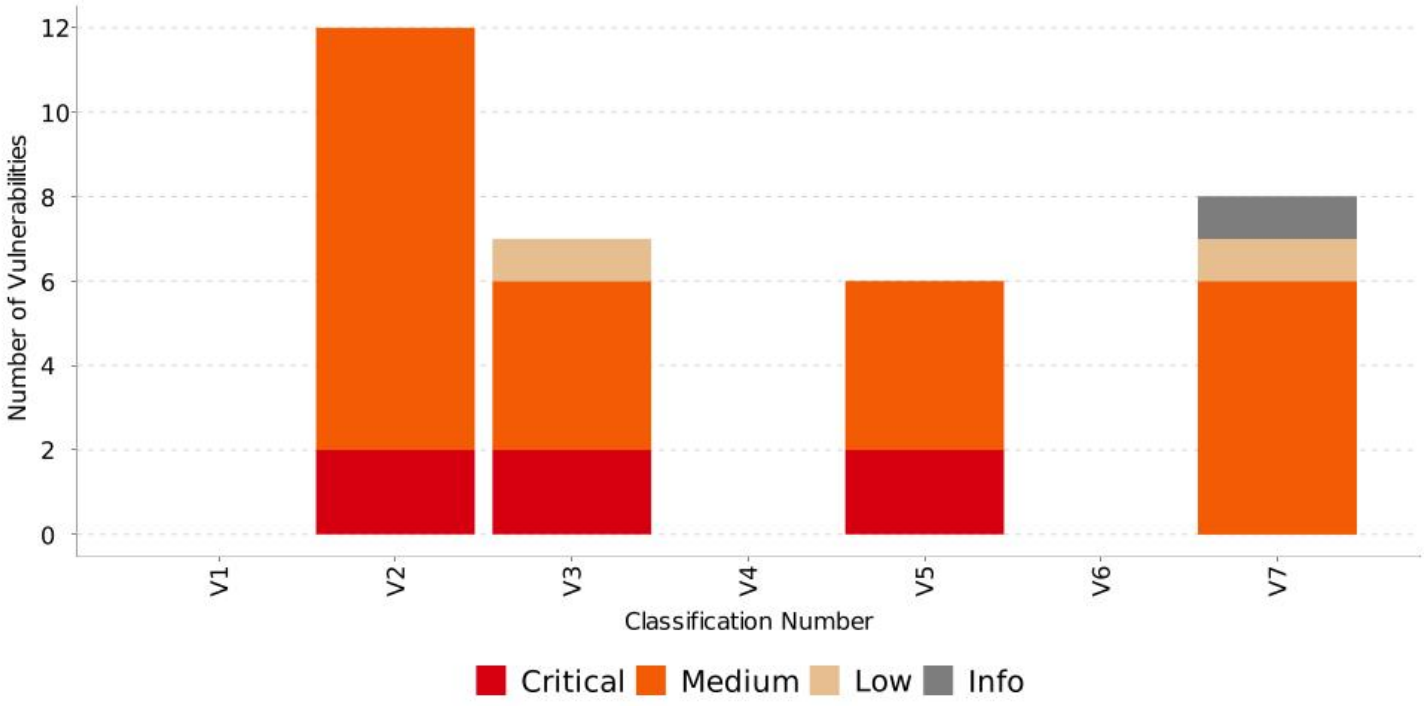


Diagram of identified vulnerabilities



Vulnerability Types

Classification by OWASP MASVS L1



	Vulnerabilities					Occurrences				
	Critical	Medium	Low	Info	Total	Critical	Medium	Low	Info	Total
V1	0	0	0	0	0	0	0	0	0	0
V2	2	10	0	0	11	4	121	0	0	125
V3	2	4	1	0	7	5	32	2	0	39

	Vulnerabilities					Occurrences				
	Critical	Medium	Low	Info	Total	Critical	Medium	Low	Info	Total
V4	0	0	0	0	0	0	0	0	0	0
V5	2	4	0	0	6	2	35	0	0	37
V6	0	0	0	0	0	0	0	0	0	0
V7	0	6	1	1	8	0	421	1	1	423

Vulnerability List

Vulnerabilities are displayed accordingly to export settings: **26 selected**

Actual: **26 of 911**

V2

Data Storage and Privacy Requirements

Critical vulnerabilities		**2***
Hardcoded password	Swift	2
DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Sensitive Information in Memory/Controller/SensitiveInformationInMemoryDetailsViewController.swift:32		Confirmed
DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Sensitive Information in Memory/Controller/SensitiveInformationInMemoryDetailsViewController.swift:19		Confirmed
Medium-level vulnerabilities		**6***
Keyboard caching	Swift	2
DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Broken Cryptography/Controller/BrokenCryptographyDetailsViewController.swift:25		Confirmed
DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Transport Layer Protection/Controller/TransportLayerProtectionViewController.swift:22		Confirmed
External storage usage	Swift	2
DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Broken Cryptography/Controller/BrokenCryptographyDetailsViewController.swift:44		Confirmed
DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Broken Cryptography/Controller/BrokenCryptographyPinDetailsViewController.swift:98		Confirmed
Unsafe internal storage	Objective-C	2
DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Side Channel Data Leakage/Controller/Objective-C Methods/SetSharedCookies.m:55		Confirmed
DVIA-v2-master/DVIA-v2/Pods/Realm/Realm/RLMAnalytics.mm:240		Confirmed
Low-level vulnerabilities		**0**
Info-level vulnerabilities		**0**

* Rejected vulnerabilities are not taken into account

V3 Cryptography Requirements

Critical vulnerabilities		2*
Weak hashing algorithm	Objective-C	2
DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/YapDatabase/Extensions/CloudKit/YapDatabaseCloudKitTransaction.m:1033		Confirmed
DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/RNCryptor/RNOpenSSLCryptor.m:60		Confirmed
Medium-level vulnerabilities		2*
Weak random number generator	Objective-C	2
DVIA-v2-master/DVIA-v2/Pods/Parse/Parse/Parse/Internal/Object/LocalIdStore/PFObjectLocalIdStore.m:198		Confirmed
DVIA-v2-master/DVIA-v2/Pods/Parse/Parse/Parse/Internal/Commands/CommandRunner/URLSession/PFURLSessionCommandRunner.m:223		Confirmed
Low-level vulnerabilities		0
Info-level vulnerabilities		0

V5 Network Communication Requirements

Critical vulnerabilities		2*
Unsafe SSL settings	Config files	1
DVIA-v2-master/DVIA-v2/DVIA-v2/Info.plist:40		Confirmed
Unsafe SSL settings	Objective-C	1
DVIA-v2-master/DVIA-v2/Pods/Realm/Realm/RLMAnalytics.mm:240		Confirmed
Medium-level vulnerabilities		2*
Cookie: transmission not over SSL	Swift	2
DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Side Channel Data Leakage/Controller/CookiesViewController.swift:74#78		Confirmed
DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Side Channel Data Leakage/Controller/CookiesViewController.swift:67#71		Confirmed
Low-level vulnerabilities		0

* Rejected vulnerabilities are not taken into account

V5 Network Communication Requirements

Info-level vulnerabilities	0

V7 Code Quality and Build Setting Requirements

Critical vulnerabilities		0
Medium-level vulnerabilities		10*
Unsafe authentication (LocalAuthentication framework)	Objective-C	2
DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/TouchIDBypass/Controller/TouchIDAuthentication.m:39		Confirmed
DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/TouchIDBypass/Controller/TouchIDAuthentication.m:39		Confirmed
Memory leak	Objective-C	2
DVIA-v2-master/DVIA-v2/DVIA-v2/Home/DeviceInfo.m:77		Confirmed
DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/PDKeychainBindings/PDKeychainBindingsController.m:58		Confirmed
Lack of dealloc method	Objective-C	2
DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/YapDatabase/Extensions/CloudKit/YapDatabaseCloudKitOptions.m:3		Confirmed
DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/YapDatabase/Extensions/CloudKit/Internal/YDBCKChangeQueue.m:15		Confirmed
NSLog usage	Objective-C	2
DVIA-v2-master/DVIA-v2/DVIA-v2/Home/DeviceInfo.m:51		Confirmed
DVIA-v2-master/DVIA-v2/Pods/Bolts/Bolts/Common/BFTask.m:20#21		Confirmed
release call in dealloc method is missed	Objective-C	2
DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/YapDatabase/Extensions/Relationships/YapDatabaseRelationshipConnection.m:67		Confirmed
DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/YapDatabase/Utilities/YapCache.m:113		Confirmed
Low-level vulnerabilities		0
Info-level vulnerabilities		0

* Rejected vulnerabilities are not taken into account

Analysis Results

V5

Unsafe SSL settings (Config files)

Description

The application establishes the SSL connection with insecure settings.
To establish a secure connection the application must verify that the certificate corresponds to the requested host, the certificate term has not expired, and that the chain of trust goes back to one of the set in the system trusted root certificates. Disabling any of these checks may lead to compromise of transferred data.
Insecure Communication takes the third place in the “OWASP Mobile Top 10 2016” mobile platforms vulnerabilities ranking.

Example

The following example shows a code fragment from the Info.plist file in which developer explicitly discards App Transport Security by adding the <true/> value in <key>NSAllowsArbitraryLoads</key>:

```
<key>NSAppTransportSecurity</key>
<dict>
   <key>NSAllowsArbitraryLoads</key>
   <true/>
</dict>
```

Using the connection without checking the certificate for the data transfer leads to its compromise.

Recommendations

- Check the certificate completely when establishing a connection via SSL / TLS protocol.

Links

1. CWE-295: Improper Certificate Validation
2. OWASP Mobile Top 10 2014: Insufficient Transport Layer Protection
3. Using Networking Securely - developer.apple.com
4. OWASP Mobile Top 10 2016-M3-Insecure Communication

Vulnerability Entries

DVIA-v2-master/DVIA-v2/DVIA-v2/Info.plist:40

Level Critical

Status Confirmed

```
37 <key>NSAppTransportSecurity</key>
38 <dict>
39     <key>NSAllowsArbitraryLoads</key>
40     <true/>
41 </dict>
42 <key>NSCameraUsageDescription</key>
43 <string>To demonstrate the misuse of Camera, please grant it permission once.</string>
```

V5

Unsafe SSL settings (Objective-C)

Description

The application establishes the SSL connection with insecure settings.
To establish a secure connection the application must verify that the certificate corresponds to the requested host, the certificate term has not expired, and that the chain of trust goes back to one of the set in the system trusted root certificates. Disabling any of these checks may lead to compromise of transferred data.
Insecure Communication takes the third place in the “OWASP Mobile Top 10 2016” mobile platforms vulnerabilities ranking.

Example

In the following example, the application creates a query disabling certificate check:
[NSURLRequest setAllowsAnyHTTPSCertificate:YES forHost:[[NSURL URLWithString: url] host]];
Using the connection without checking the certificate for the data transfer leads to its compromise.

Recommendations

- Check the certificate completely when establishing a connection via SSL / TLS protocol.

Links

1. CWE-295: Improper Certificate Validation
2. OWASP Mobile Top 10 2014: Insufficient Transport Layer Protection
3. Using Networking Securely - developer.apple.com
4. OWASP Mobile Top 10 2016-M3-Insecure Communication

Vulnerability Entries

DVIA-v2-master/DVIA-v2/Pods/Realm/Realm/RLMAnalytics.mm:240

Level Critical

Status Confirmed

```
237
238    // No error handling or anything because logging errors annoyed people for no
239    // real benefit, and it's not clear what else we could do
240    [[NSURLSession.sharedSession dataTaskWithURL:[NSURL URLWithString:url]] resume];
241 }
242
243 #else
```

V3

Weak hashing algorithm (Objective-C)

Description

The used hash function is insecure. Its use can lead to a data confidentiality loss.
The MD2, MD5, SHA1 hash functions have known vulnerabilities. Finding collisions for MD2 and MD5 functions does not require substantial resources; a similar problem for SHA1 was solved. If these functions are used to store valuable information (such as passwords), its confidentiality can be violated.
The hash function used to store passwords not only should be resistant to collisions but also should not be too fast. This complicates the attack by exhaustive search. For this purpose specialized hash functions have been developed: bcrypt, scrypt.
Suppose that user passwords are stored on the server in encrypted form with the use of insecure hash function (e.g., MD5). A possible attack scenario:

1. The attacker gets access to the encrypted passwords.
2. An attacker exploits a vulnerability of hashing algorithm and calculates the string for which the hash algorithm gives the same value as for the user's password.
3. The attacker passes the authentication using a calculated string.

Insufficient Cryptography vulnerabilities take the fifth place in the “OWASP Top 10 2016” mobile application vulnerabilities ranking.

Example

In the following example, a value of the insecure MD5 hash function (CommonCrypto library) is calculated:

```
+ (NSString *)md5HexDigest:(NSString *)plainText {
   const char* str = [plainText UTF8String];
   unsigned char result[CC_MD5_DIGEST_LENGTH] ;
   CC_MD5(str, strlen(str), result);
   CC_MD5_DIGEST_LENGTH=16,
   NSMutableString *ret = [NSMutableString stringWithCapacity:
CC_MD5_DIGEST_LENGTH * 2];

   for (int i = 0; i < CC_MD5_DIGEST_LENGTH; i++) {
      [ret appendFormat:@"%02x", result[i]];
   }
   return ret;
}
```

Recommendations

- Use secure hash functions (SHA-2).
- For hashing passwords use specialized hash functions (bcrypt, scrypt) and salt obtained from a cryptographically resistant PRNG (pseudorandom number generators).

Links

1. Mobile Top 10 2016-M5-Insufficient Cryptography
2. OWASP Top 10 2013-A6-Sensitive Data Exposure
3. OWASP: Top 10 2010-A7-Insecure Cryptographic Storage
4. CWE-326: Inadequate Encryption Strength
5. NIST Approved Algorithms
6. How to securely hash passwords - Thomas Pornin / stackoverflow.com
7. MD5 considered harmful today. Creating a rogue CA certificate - Alexander Sotirov, Marc Stevens, Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, Benne de Weger / win.tue.nl
8. Encrypting and Hashing Data - developer.apple.com
9. CWE-328
10. Bleichenbacher’s attack

Vulnerability Entries

DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/RNCryptor/RNOpenSSLCryptor.m:60

Level Critical

Status Confirmed

```
57
58  NSMutableData *hashMaterial = [NSMutableData dataWithData:hash];
59  [hashMaterial appendData:passwordSalt];

60  CC_MD5([hashMaterial bytes], (CC_LONG)[hashMaterial length], md);

61
62  return [NSData dataWithBytes:md length:sizeof(md)];
63 }
```

DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/YapDatabase/Extensions/CloudKit/YapDatabaseCloudKitTransaction.m:1033

Level Critical

Status Confirmed

```
1030            buffer = malloc((size_t)maxLen);
1031
1032 CC_SHA1_CTX ctx;

1033 CC_SHA1_Init(&ctx);

1034
1035 NSUInteger used = 0;
1036
```

V2

Hardcoded password (Swift)

Description

Password is hardcoded. This may lead to an application data compromise.
Eliminating security risks related to hardcoded passwords is extremely difficult. These passwords are at least accessible to every developer of the application. Moreover, after the application is installed, removing password from its code is possible only via an update. Constant strings are easily extracted from the compiled application by decompilers. Therefore, an attacker does not necessarily need to have an access to the source code to know the parameters of the special account. If these parameters become known to an attacker, system administrators will be forced either to neglect the safety, or to restrict the access to the application.
In case of a mobile application, security threat is even higher, considering the risk of the device loss.

Example

In the following example, the variable Password is defined, the value of which is explicitly indicated in the code:
var Password = "password"

Recommendations

- Store not passwords but values of cryptographically secure hash function from the password. Use specialized hash functions designed for this purpose. Use salt obtained from cryptographically secure pseudorandom number generator to resist attacks which use rainbow tables.
- If the hardcoded password is used for the initial authorization, provide the special authentication mode for this purpose in which the user is required to provide his/her own unique password.
- Store authentication information in an encrypted form in a separate configuration file or in a database. Secure the encryption key. If encryption is not possible, limit the access to the repository as much as possible.

Links

1. Use of hard-coded password
2. CWE-259: Use of Hard-coded Password
3. OWASP Top 10 2013-A5-Security Misconfiguration
4. OWASP Top 10 2013-A6-Sensitive Data Exposure
5. How to securely hash passwords? - security.stackexchange.com
6. OWASP Mobile Top 10 2016-M4-Insecure Authentication
7. CWE-798: Use of Hard-coded Credentials

Vulnerability Entries

DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Sensitive Information in Memory/Controller/SensitiveInformationInMemoryDetailsViewController.swift:19

Level Critical

Status Confirmed

```
16
17 class SensitiveInformationInMemoryDetailsViewController: UIViewController {
18     let username = "edhillary"
19     let password = "ev8848@1953"
20     override func viewDidLoad() {
21         super.viewDidLoad()
22         self.navigationItem.title = "Sensitive Information in Memory"
```

DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Sensitive Information in Memory/Controller/SensitiveInformationInMemoryDetailsViewController.swift:32

Level Critical

Status Confirmed

```
29 }
30
31 func initializeIVars(){
32     let passwd = "tenzinnorgay"
33     let concatenated = username + passwd + password
34 }
35
```

V7

Lack of dealloc method (Objective-C)

Description

There is no dealloc method in the class when ARC is disabled. This may cause a memory leak.
The dealloc method frees the memory occupied by the object. This method is automatically called just before the object is deallocated or any of its instance variables are destroyed. When ARC is disabled, you need to add the method dealloc in the implementation of the class, which correctly frees the instance variables. When using ARC, instance variables are destroyed automatically, but it is necessary to override dealloc in order to, for example, remove an object from other services and managers to which it is subscribed, invalidate timers, as well as to release non-Objective-C objects.

Example

In the following example, there is no dealloc method:

```
@interface MyObject : NSObject {
  id _ivar;
}
@end

@implementation MyObject // warn: lacks 'dealloc'
@end
```

Recommendations

- Free the object after the completion of all operations with it.
- Always override dealloc when ARC is disabled.

Links

1. CWE-401: Improper Release of Memory Before Removing Last Reference (‘Memory Leak’)
2. Mobile Top 10 2016-M7-Poor Code Quality
3. dealloc

Vulnerability Entries

DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/YapDatabase/Extensions/CloudKit/Internal/YDBCKChangeQueue.m:15

Level Medium

Status Confirmed

```
12 @property (atomic, readwrite, strong) NSString *lockUUID;
13 @end
14

15 @implementation YDBCKChangeQueue

16 {
17     BOOL isMasterQueue;
18     NSLock *masterQueueLock;
```

Trace

@implementation YDBCKChangeQueue

DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/YapDatabase/Extensions/CloudKit/Internal/YDBCKChangeQueue.m:15

```
12 @property (atomic, readwrite, strong) NSString *lockUUID;
13 @end
14

15 @implementation YDBCKChangeQueue

16 {
17     BOOL isMasterQueue;
18     NSLock *masterQueueLock;
```

@implementation YDBCKChangeQueue

DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/YapDatabase/Extensions/CloudKit/Internal/YDBCKChangeQueue.m:15

```
12 @property (atomic, readwrite, strong) NSString *lockUUID;
13 @end
14

15 @implementation YDBCKChangeQueue

16 {
17     BOOL isMasterQueue;
18     NSLock *masterQueueLock;
```

DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/YapDatabase/Extensions/CloudKit/YapDatabaseCloudKitOptions.m:3

Level Medium

Status Confirmed

```
1 #import "YapDatabaseCloudKitOptions.h"
2
3 @implementation YapDatabaseCloudKitOptions
4
5 @synthesize allowedCollections = allowedCollections;
6
```

Trace

@implementation YapDatabaseCloudKitOptions

DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/YapDatabase/Extensions/CloudKit/YapDatabaseCloudKitOptions.m:3

```
1 #import "YapDatabaseCloudKitOptions.h"
2
3 @implementation YapDatabaseCloudKitOptions
4
5 @synthesize allowedCollections = allowedCollections;
6
```

@implementation YapDatabaseCloudKitOptions

DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/YapDatabase/Extensions/CloudKit/YapDatabaseCloudKitOptions.m:3

```
1 #import "YapDatabaseCloudKitOptions.h"
```

```
2
3 @implementation YapDatabaseCloudKitOptions
4
5 @synthesize allowedCollections = allowedCollections;
6
```

V7

Memory leak (Objective-C)

Description

The application allows incorrect work with memory. Some errors related to memory management: no release of allocated memory (memory leak), double memory allocation, the use of the wrong class to free memory. Objective C language is based on C and involves working with memory at a relatively low level. The developer must allocate memory for the structures used and release it correctly after the end of the work. Otherwise, memory available for the application will decrease, eventually causing malfunction of the application.

Example

In the following example, the application does not release the allocated memory (the SecKeychainItemFreeContent call is missing):

```
void errRetVal() {
   unsigned int *ptr = 0;
   OSStatus st = 0;
   UInt32 length;
   void *outData;
   st = SecKeychainItemCopyContent(2, ptr, ptr, &length, &outData);
   if (st == GenericError)
      SecKeychainItemFreeContent(ptr, outData);
}
```

Recommendations

- Free the allocated memory correctly.
- Check the documentation of methods of work with memory.

Links

1. Keychain Services Reference - developer.apple.com
2. About Memory Management - developer.apple.com
3. Mobile Top 10 2016-M7-Poor Code Quality

Vulnerability Entries

DVIA-v2-master/DVIA-v2/DVIA-v2/Home/DeviceInfo.m:77

Level Medium

Status Confirmed

```
74
75      NSLog(@"wifi info: bssid: %@, ssid:%@, ssidData: %@", info[@"BSSID"], info[@"SSID"], info
[@"SSIDDATA"]);
76    }
77    return [NSString stringWithFormat:@"Sysname: %s\nNodename: %s\nRelease: %s\nVersion: %
s\nMachine: %s\nMemory in use (in MB): %f\nPID: %d\n", u.sysname,u.nodename,u.release,u.
version,u.machine,((CG...
78
79 }
80
```

Trace

kerr == KERN_SUCCESS

DVIA-v2-master/DVIA-v2/DVIA-v2/Home/DeviceInfo.m:49

```
46                  TASK_BASIC_INFO,
47                  (task_info_t)&info,
48                  &size);
49 if( kerr == KERN_SUCCESS ) {
50
51    NSLog(@"Memory in use (in MB): %f", ((CGFloat)info.resident_size / 1000000));
52 } else {
```

return [NSString stringWithFormat:@"Sysname: %s\nNodename: %s\nRelease: %s\nVersion: %s\nMachine: %s\nMemory in use (in MB): %f\nPID: %d\n", u.sysname,u.nodename,u.release,u.version,u.machine,((CGFloat)info.resident_size / 1000000),getpid()]

DVIA-v2-master/DVIA-v2/DVIA-v2/Home/DeviceInfo.m:77

```
74
75      NSLog(@"wifi info: bssid: %@, ssid:%@, ssidData: %@", info[@"BSSID"], info[@"SSID"], info[@"SSIDDATA"]);
76  }
77  return [NSString stringWithFormat:@"Sysname: %s\nNodename: %s\nRelease: %s\nVersion: %s\nMachine: %s\nMemory in use (in MB): %f\nPID: %d\n", u.sysname,u.nodename,u.release,u.version,u.machine,((CG...
78
79 }
80
```

DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/PDKeychainBindings/PDKeychainBindingsController.m:58

Level Medium

Status Confirmed

```
55  CFRelease(stringData);
56 #else //OSX
57  NSString *string = [[NSString alloc] initWithBytes:stringBuffer length:stringLength encoding:NSUTF8StringEncoding];
58  SecKeychainItemFreeAttributesAndData(NULL, stringBuffer);
59 #endif
60  return string;
61 }
```

Trace

&stringBuffer

DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/PDKeychainBindings/PDKeychainBindingsController.m:49

```
46    void *stringBuffer;
47    status = SecKeychainFindGenericPassword(NULL, (uint) [[self serviceName]
lengthOfBytesUsingEncoding:NSUTF8StringEncoding], [[self serviceName] UTF8String],
48                                            (uint) [key lengthOfBytesUsingEncoding:NSUTF8StringEncoding],
[key UTF8String],
49                                            &stringLength, &stringBuffer, NULL);
50    #endif
51 if(status) return nil;
52
```

SecKeychainItemFreeAttributesAndData(NULL, stringBuffer)

DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/PDKeychainBindings/PDKeychainBindingsController.m:58

```
55     CFRelease(stringData);
56 #else //OSX
57     NSString *string = [[NSString alloc] initWithBytes:stringBuffer length:stringLength
encoding:NSUTF8StringEncoding];
58     SecKeychainItemFreeAttributesAndData(NULL, stringBuffer);
59 #endif
60     return string;
61 }
```

V7

NSLog usage (Objective-C)

Description

The application uses an NSLog method. This method is used for debugging and not supposed to be left in a release version of an application. All messages generated by NSLog can be read using XCode, leaving a possibility of unnecessary information disclosure. Extensive use of NSLog can also significantly slows down the application.

In some cases, using NSLog on the jailbroken device may lead to redirecting output NSLog to syslog. So developers should avoid using NSLog for logging sensitive or system information.

Example

In the following example, the application logges previously saved login and password:
NSLog(@"Login: %@, Password: %@", login, password);

Recommendations

- Disable NSLog in release versions using preprocessor macros.

Links

1. Stackoverflow: Do I need to disable NSLog before release Application?
2. Mobile Top 10 2016-M7-Poor Code Quality

Vulnerability Entries

DVIA-v2-master/DVIA-v2/DVIA-v2/Home/DeviceInfo.m:51

Level Medium

Status Confirmed

```
48                          &size);
49 if( kerr == KERN_SUCCESS ) {
50
51     NSLog(@"Memory in use (in MB): %f", ((CGFloat)info.resident_size / 1000000));
52 } else {
53     NSLog(@"Error with task_info(): %s", mach_error_string(kerr));
54 }
```

DVIA-v2-master/DVIA-v2/Pods/Bolts/Bolts/Common/BFTask.m:20#21

Level Medium

Status Confirmed

```
17 NS_ASSUME_NONNULL_BEGIN
18
19 __attribute__ ((noinline)) void warnBlockingOperationOnMainThread() {

20     NSLog(@"Warning: A long-running operation is being executed on the main thread. \n"
21           " Break on warnBlockingOperationOnMainThread() to debug.");

22 }
23
24 NSString *const BFTaskErrorDomain = @"bolts";
```

V7

release call in dealloc method is missed (Objective-C)

Description

dealloc method is implemented incorrectly in the application. An instance variable was retained by the synthesized property, but was not freed (lack of release method call). This may cause a memory leak.
The dealloc method frees the memory occupied by the object. This method is automatically called just before the object is deallocated or any of its instance variables are destroyed. When ARC is disabled, you need to add the method dealloc in the implementation of the class, which correctly frees the instance variables. When using ARC, instance variables are destroyed automatically, but it is necessary to override dealloc in order to, for example, remove an object from other services and managers to which it is subscribed, invalidate timers, as well as to release non-Objective-C objects.

Example

In the following example, instance variable was retained by the synthesized property, but was not freed:

```
@interface MyObject : NSObject {
  id _myproperty;
}
@property(retain) id myproperty;
@end

@implementation MyObject
@synthesize myproperty = _myproperty;
  // warn: var was retained but wasn't released
- (void)dealloc {
  [super dealloc];
}
```

```
@end
```

Recommendations

- Free the object after the completion of all operations with it.
- Do not use the object after releasing it.

Links

1. CWE-401: Improper Release of Memory Before Removing Last Reference (‘Memory Leak’)
2. Mobile Top 10 2016-M7-Poor Code Quality
3. dealloc

Vulnerability Entries

DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/YapDatabase/Extensions/Relationships/YapDatabaseRelationshipConnection.m:67

Level Medium

Status Confirmed

```
64 - (void)dealloc
65 {
66     [self _flushStatements];
67 }
68
69 - (void)_flushStatements
70 {
```

Trace

```
}
```

DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/YapDatabase/Extensions/Relationships/YapDatabaseRelationshipConnection.m:67

```
64 - (void)dealloc
```

```
65 {
66     [self _flushStatements];
67 }
68
69 - (void)_flushStatements
70 {
```

}

DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/YapDatabase/Extensions/Relationships/YapDatabaseRelationshipConnection.m:67

```
64 - (void)dealloc
65 {
66     [self _flushStatements];
67 }
68
69 - (void)_flushStatements
70 {
```

DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/YapDatabase/Utilities/YapCache.m:113

Level Medium

Status Confirmed

```
110 - (void)dealloc
111 {
112   if (cfdict) CFRelease(cfdict);
113 }
114
115 - (NSUInteger)countLimit
116 {
```

Trace

if (cfdict) CFRelease(cfdict)

DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/YapDatabase/Utilities/YapCache.m:112

```
109
110 - (void)dealloc
111 {
112   if (cfdict) CFRelease(cfdict);
113 }
114
115 - (NSUInteger)countLimit
```

}

DVIA-v2-master/DVIA-v2/DVIA-v2/Vendor/YapDatabase/Utilities/YapCache.m:113

```
110 - (void)dealloc
111 {
112   if (cfdict) CFRelease(cfdict);
113 }
114
115 - (NSUInteger)countLimit
116 {
```

V7

Unsafe authentication (LocalAuthentication framework) (Objective-C)

Description

The application uses framework LocalAuthentication to authenticate the user. This framework doesn’t use Secure Enclave and is prone to hooking on jailbroken devices.
Authentication contexts are used to evaluate authentication policies, allowing apps to request the user to authenticate themselves using personal information such as a fingerprint registered with Touch ID. Touch ID can be implemented in two ways: using the LocalAuthentication framework or using access control based on the Touch ID in the Keychain service. Although both methods must be suitable for most applications, LocalAuthentication has some characteristics that make it less secure for high-risk applications such as banking

and insurance:

- LocalAuthentication is determined outside the device's Secure Enclave, which means that their APIs can be connected and modified on jailbrocken devices.
- LocalAuthentication verifies the authenticity of the user by evaluating the context policy that can be either true or false. This logical assessment implies that the application can not be authenticated by anyone.
- In addition, fingerprints that can be registered in the future will also be successfully evaluated as true.

An LAContext object represents an authentication context. The LAContext class provides a programmatic interface for evaluating authentication policies and access controls, managing credentials, and invalidating authentication contexts.
LAContext.evaluatePolicy does not successfully authenticate a user without the risk of other registered fingerprints being used. It also poses the risk a malicious actor can steal or find a victim's iOS device and has the ability to bypass the TouchID used in other applications.
Insecure Authentication vulnerabilities take the fourth place in the "OWASP Top 10 2016" mobile application vulnerabilities ranking.

Example

In the following example, the application uses LocalAuthentication framework to authenticate user by fingerprint.

```
LAContext *myContext = [[LAContext alloc] init];
NSError *authError = nil;
NSString *myLocalizedReasonString = @"Authenticate using your finger";
if ([myContext canEvaluatePolicy:LAPolicyDeviceOwnerAuthenticationWithBiometrics
error:&authError]) {
   [myContext evaluatePolicy:LAPolicyDeviceOwnerAuthenticationWithBiometrics
         localizedReason:myLocalizedReasonString
            reply:^(BOOL) success, NSError *error) {
      if (success) {
         // fingerprint match
      } else {
         // fingerprint doesn't match
      }
   }];
} else {
   NSLog(@"Can not evaluate Touch ID");
}
```

Using the connection without checking the certificate for the data transfer leads to its compromise.

Recommendations

- Use Touch ID based access controls in the Keychain service.

Links

1. LocalAuthentication - developer.apple.com
2. OWASP Mobile Top 10 2016-M1-Improper Platform Usage

Vulnerability Entries

DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/TouchIDBypass/Controller/TouchIDAuthentication.m:39

Level Medium

Status Confirmed

```
36
37 +(void)authenticateWithTouchID {
38
39    LAContext *myContext = [[LAContext alloc] init];
40    NSError *authError = nil;
41    NSString *myLocalizedReasonString = @"Please authenticate yourself";
42
```

DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/TouchIDBypass/Controller/TouchIDAuthentication.m:39

Level Medium

Status Confirmed

```
36
37 +(void)authenticateWithTouchID {
38
39    LAContext *myContext = [[LAContext alloc] init];
```

```
40  NSError *authError = nil;
41  NSString *myLocalizedReasonString = @"Please authenticate yourself";
42
```

V2

Unsafe internal storage (Objective-C)

Description

Storing data in the app's home directory is insecure. If it is necessary to store data in app's home directory then data should be stored in encrypted form. Besides, use secure encryption settings.
iOS allows developers to specify which data must be encrypted when writing to the file. It uses the Data Protection API. The default mode is NSFileProtectionNone, in which the data is protected only by the basic encryption based on the device UID-key. Thus, by default the data is stored insecurely and is available at boot time or when the device is unlocked.
Possible values of constants that define the level of encryption are set for NSFileManager:

- NSFileProtectionComplete
- NSDataWritingFileProtectionComplete
- NSFileProtectionCompleteUnlessOpen
- NSDataWritingFileProtectionCompleteUnlessOpen
- NSFileProtectionCompleteUntilFirstUserAuthentication
- NSDataWritingFileProtectionCompleteUntilFirstUserAuthentication
- NSFileProtectionNone
- NSDataWritingFileProtectionNone

Example

In the following example, the application stores data without adequate protection:
filepath = [self.GetDocumentDirectory stringByAppendingPathComponent:self.setFilename];
NSDictionary *protection = [NSDictionary dictionaryWithObject:NSFileProtectionNone forKey:NSFileProtectionKey];
[[NSFileManager defaultManager] setAttributes:protection ofItemAtPath:filepath error:nil];
BOOL ok = [testToWrite writeToFile:filepath atomically:YES encoding: NSUnicodeStringEncoding error:&err];
Secure alternative:
filepath = [self.GetDocumentDirectory stringByAppendingPathComponent:self.setFilename];
NSDictionary *protection = [NSDictionary dictionaryWithObject:

```
NSFileProtectionComplete forKey:NSFileProtectionKey];
[[NSFileManager defaultManager] setAttributes:protection ofItemAtPath:filepath
error:nil];
BOOL ok = [testToWrite writeToFile:filepath atomically:YES encoding:
NSUnicodeStringEncoding error:&err];
```

Recommendations

- When storing data, use only NSFileProtectionComplete and NSDataWritingFileProtectionComplete.

Links

1. OWASP Top 10 2013-A6-Sensitive Data Exposure
2. CWE-359: Exposure of Private Information (‘Privacy Violation’)
3. OWASP Mobile Top 10 2016-M2-Insecure Data Storage
4. CWE-921: Storage of Sensitive Data in a Mechanism without Access Control

Vulnerability Entries

DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Side Channel Data Leakage/Controller/Objective-C Methods/SetSharedCookies.m:55

Level Medium

Status Confirmed

```
52                              };
53    NSHTTPCookie *passwordCookie = [[NSHTTPCookie alloc] initWithProperties:
passwordProperties];
54
55    [[NSHTTPCookieStorage sharedHTTPCookieStorage] setCookies:@[usernameCookie,
passwordCookie] forURL:siteURL mainDocumentURL:nil];
56 }
57
58 @end
```

DVIA-v2-master/DVIA-v2/Pods/Realm/Realm/RLMAnalytics.mm:240

Level Medium

Status Confirmed

```
237
238    // No error handling or anything because logging errors annoyed people for no
239    // real benefit, and it's not clear what else we could do
240    [[NSURLSession.sharedSession dataTaskWithURL:[NSURL URLWithString:url]] resume];
241 }
242
243 #else
```

V3

Weak random number generator (Objective-C)

Description

Used pseudorandom number generator (PRNG) is not secure since it generates predictable sequences. This can be exploited to bypass authentication and hijack the user’s session, as well as to carry out the DNS cache poisoning attack.
PRNGs generate number sequences based on the initial value of the seed. There are two types of PRNG: statistical and cryptographic. Statistical PRNGs generate predictable sequences, which are similar to random according to the statistical characteristics. They may not be used for security purposes. The result of the cryptographic PRNG, on the contrary, is impossible to predict if the value of seed is derived from a source with high entropy. The value of the current time has a small entropy and is also insecure as a seed.
Insufficient Cryptography vulnerabilities take the fifth place in the “OWASP Top 10 2016” mobile application vulnerabilities ranking.

Example

In the following example, the application generates a predictable sequence of pseudorandom numbers:
#include <stdlib.h>
int r = rand() % N // in range 0 to N-1
The random(), arc4random(), arc4random_uniform(), and srandom() methods are also cryptographically insecure.
It is recommended to use the data from the dev/random file (system entropy source):
FILE *fp = fopen("/dev/random", "r");

```
if (!fp) {
   perror("randgetter");
   exit(-1);
}

uint64_t value = 0;
int i;
for (i=0; i<sizeof(value); i++) {
   value <<= 8;
   value |= fgetc(fp);
}

fclose(fp);
```

Recommendations

- Use cryptographic PRNG to generate pseudo-random numbers for information security purposes.
- Use sources of high entropy to generate a seed for PRNG.

Links

1. OWASP: Insecure randomness
2. CWE-330: Use of Insufficiently Random Values
3. CERT: MSC02-J. Generate strong random numbers
4. Generating Random Numbers - developer.apple.com
5. Mobile Top 10 2016-M5-Insufficient Cryptography
6. CWE-338

Vulnerability Entries

DVIA-v2-master/DVIA-v2/Pods/Parse/Parse/Parse/Internal/Commands/CommandRunner/URLSession/PFURLSessionCommandRunner.m:223

Level Medium

Status Confirmed

```
220 // Set the initial delay to something between 1 and 2 seconds. We want it to be
221 // random so that clients that fail simultaneously don't retry on simultaneous
222 // intervals.

223 delay += self.initialRetryDelay * ((double)(arc4random() & 0x0FFFF) / (double)0x0FFFF);

224 return [self _performCommandRunningBlock:block
225                 withCancellationToken:cancellationToken
226                                 delay:delay
```

DVIA-v2-master/DVIA-v2/Pods/Parse/Parse/Parse/Internal/Object/LocalIdStore/PFObjectLocalIdStore.m: 198

Level Medium

Status Confirmed

```
195
196 // Start by generating a number. It will be the localId as a base-52 number.
197 // It has to be a uint64_t because log256(52^10) ~= 7.13 bytes.

198 uint64_t localIdNumber = (((uint64_t)arc4random()) << 32) | ((uint64_t)arc4random());

199 NSString *localId = [NSString stringWithFormat:@"local_%016llx", localIdNumber];
200
201 PFConsistencyAssert([[self class] isLocalId:localId], @"Generated an invalid local id: \"%@\".",
localId);
```

V5

Cookie: transmission not over SSL (Swift)

Description

The application creates cookies without setting the secure flag to true This allows to transfer cookies in clear text over HTTP, which can violate their confidentiality.
Sensitive Data Exposure vulnerabilities take the third place in the “OWASP Top 10 2017” web-application vulnerabilities ranking.

Example

In the following example, the application creates cookies without the secure flag:
```
let cookie = HTTPCookie(properties: [
   .domain: "www.test.com",
   .path: "/some_path",
   .name: "some_name",
   .value: "some_value",
])
```
If the application uses both HTTPS and HTTP, then in the absence of the secure flag cookies that were created within HTTPS request will be transferred in unencrypted form in subsequent HTTP requests, which may compromise the application. This is particularly dangerous if the cookies contain valuable data, in particular the session identifier.

Recommendations

- Set the secure flag when creating cookies.

```
let cookie = HTTPCookie(properties: [ .domain: “www.test.com”, .path: “/some_path”, .name: “some_name”, .value: “some_value”, .secure: “true” ])
```

Links

1. OWASP Top 10 2013-A5-Security Misconfiguration
2. OWASP Top 10 2017-A2-Broken Authentication
3. CWE-614: Sensitive Cookie in HTTPS Session Without ‘Secure’ Attribute
4. CWE-1028: Broken Authentication

Vulnerability Entries

DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Side Channel Data Leakage/Controller/CookiesViewController.swift:67#71

Level Medium

Status Confirmed

```
64
65 guard let siteUrl = URL(string: SiteURLString) else { return }
66

67 let usernameProperties: [HTTPCookiePropertyKey : Any] = [.domain: siteUrl.host ?? "",
68                                                          .path: siteUrl.path,
69                                                          .name: "username",
70                                                          .value: CookieUsername,
71                                                          .expires: expireInterval!]

72 guard let usernameCookie = HTTPCookie(properties: usernameProperties) else {return}
73
74 let passwordProperties: [HTTPCookiePropertyKey: Any] = [.domain: siteUrl.host ?? "",
```

DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Side Channel Data Leakage/Controller/CookiesViewController.swift:74#78

Level Medium

Status Confirmed

```
71                                                          .expires: expireInterval!]
72 guard let usernameCookie = HTTPCookie(properties: usernameProperties) else {return}
73

74 let passwordProperties: [HTTPCookiePropertyKey: Any] = [.domain: siteUrl.host ?? "",
75                                                          .path: siteUrl.path,
76                                                          .name: "password",
77                                                          .value: CookiePassword,
78                                                          .expires: expireInterval!]

79 guard let passwordCookie = HTTPCookie(properties: passwordProperties) else {return}
80
81 HTTPCookieStorage.shared.setCookies([usernameCookie, passwordCookie], for: siteUrl,
mainDocumentURL: nil)
```

V2

External storage usage (Swift)

Description

The application writes data to an external storage device.
Files written to external storage device are readable by all applications and can be changed. Besides, files stored in external storage will remain there even after the application is deleted. This can lead to a valuable data confidentiality loss.

Example

In the following example, the application writes file to documents directory:
guard let directory = FileManager.default.urls(for: .documentDirectory, in: .userDomainMask).first else { return }
let fileURL = directory.appendingPathComponent(file)
text.write(to: fileURL, atomically: false, encoding: .utf8)

Recommendations

- Store files in the internal memory, then they will only be available to the application that stored them.

Links

1. Reading, Writing, and Deleting Files in Swift by Corey Davis
2. OWASP: Insecure Storage
3. CWE-250: Execution with Unnecessary Privileges
4. CWE-921: Storage of Sensitive Data in a Mechanism without Access Control

Vulnerability Entries

DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Broken Cryptography/Controller/BrokenCryptographyDetailsViewController.swift:44

Level Medium

Status Confirmed

```
41 } else {
42     let data = passwordTextField.text?.data(using: String.Encoding.utf8)
43     let encryptedData = try? RNEncryptor.encryptData(data, with: kRNCryptorAES256Settings, password: "@daloq3as$qweasdlasasjdnj")
44     try? encryptedData?.write(to: dataPath, options: .atomic)
```

```
45    UserDefaults.standard.set(true, forKey: "loggedIn")
46    UserDefaults.standard.synchronize()
47    firstTimeUserView.isHidden = true
```

Trace

URL

DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Broken Cryptography/Controller/BrokenCryptographyDetailsViewController.swift:36

```
33 extension BrokenCryptographyDetailsViewController: UITextFieldDelegate {
34
35    func textFieldShouldReturn(_ textField: UITextField) -> Bool {

36        let dataPath = URL(fileURLWithPath: NSSearchPathForDirectoriesInDomains(.
documentDirectory, .userDomainMask, true).first!).appendingPathComponent("/secret-data").
absoluteURL

37        if textField == passwordTextField {
38            textField.resignFirstResponder()
39            if textField.text == nil {
```

dataPath

DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Broken Cryptography/Controller/BrokenCryptographyDetailsViewController.swift:44

```
41 } else {
42    let data = passwordTextField.text?.data(using: String.Encoding.utf8)
43    let encryptedData = try? RNEncryptor.encryptData(data, with: kRNCryptorAES256Settings,
password: "@daloq3as$qweasdlasasjdnj")

44    try? encryptedData?.write(to: dataPath, options: .atomic)

45    UserDefaults.standard.set(true, forKey: "loggedIn")
46    UserDefaults.standard.synchronize()
47    firstTimeUserView.isHidden = true
```

DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Broken Cryptography/Controller/BrokenCryptographyPinDetailsViewController.swift:98

Level Medium

Status Confirmed

```
95 } else {
96     //First time user, save the encrypted data
97     print("encryptedData (SHA1): \(encryptedData! as NSData)")

98     try? encryptedData?.write(to: dataPath, options: .atomic)

99     UserDefaults.standard.set(true, forKey: "loggedIn")
100    UserDefaults.standard.synchronize()
101    firstTimeUserView.isHidden = true
```

Trace

URL

DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Broken Cryptography/Controller/BrokenCryptographyPinDetailsViewController.swift:89

```
86 let keyByteCount = 16
87 //Could be 100000 :/
88 let rounds = 500

89 let dataPath = URL(fileURLWithPath: NSSearchPathForDirectoriesInDomains(.
documentDirectory, .userDomainMask, true).first!).appendingPathComponent
("/v324dsa13qasd.enc").absoluteURL

90 let encryptedData = pbkdf2(password:textField.text!, salt:salt, keyByteCount:keyByteCount,
rounds:rounds)
91 if textField == passwordTextField {
92     textField.resignFirstResponder()
```

dataPath

DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Broken Cryptography/Controller/BrokenCryptographyPinDetailsViewController.swift:98

```
95 } else {
```

```
96    //First time user, save the encrypted data
97    print("encryptedData (SHA1): \(encryptedData! as NSData)")

98    try? encryptedData?.write(to: dataPath, options: .atomic)

99    UserDefaults.standard.set(true, forKey: "loggedIn")
100    UserDefaults.standard.synchronize()
101    firstTimeUserView.isHidden = true
```

V2

Keyboard caching (Swift)

Description

The identified text field does not disable the iOS keyboard caching mechanism, as a result any information recently entered from the keyboard will be cached in order to improve the autocorrect feature.
iOS caches input to text fields in order to improve the performance of the autocorrect feature and predictive typing. Any information entered into such a text field or other input control can be written to the keyboard cache file stored in the file system. Since this file is stored on the device, then if device is lost, you can restore it and reveal any confidential information contained in it.
Private data can enter a program in a variety of ways:

- Directly from the user in the form of a password or personal information,
- Accessed from a database,
- From other database,
- Indirectly from other fird party,
- From the cloud storage (for instance, iCloud), including Address book, configuration files, archived messages, snapped photos and etc.

Generally, the overall risk is associated with inappropriate reliance on the operating environment in which the program runs. Storage of personal information in file systems, registries or other locally managed resources is unacceptable.

Example

The following example shows that an application uses an input control designed to collect confidential information:

```
//...
@IBOutlet weak var creditCardNum: UITextField!
//...
```

Recommendations

- Do not trust all persons with access to file resources.
- Use different levels of access for different people when organizing data storage in file systems.

Links

1. Privacy Policy - Federal Trade Commission

Vulnerability Entries

DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Broken Cryptography/Controller/BrokenCryptographyDetailsViewController.swift:25

Level Medium

Status Confirmed

```
22 @IBOutlet var passwordTextField: UITextField!
23 @IBOutlet var returningUserView: UIView!
24 @IBOutlet var welcomeReturningUserLabel: UILabel!
25 @IBOutlet var returningUserPasswordTextField: UITextField!
26 @IBOutlet var loggedInLabel: UILabel!
27
28 override func viewDidLoad() {
```

DVIA-v2-master/DVIA-v2/DVIA-v2/Vulnerabilities/Transport Layer Protection/Controller/TransportLayerProtectionViewController.swift:22

Level Medium

Status Confirmed

```
19 class TransportLayerProtectionViewController: UIViewController {
20
21    @IBOutlet var cardNumberTextField: UITextField!
22    @IBOutlet var CVVTextField: UITextField!
23    @IBOutlet var nameOnCardTextField: UITextField!
24
25    //As per Apr 11, 2018 for example.com
```

03

About OWASP MASVS L1

MASVS L1 - List of general security controls for mobile applications. An application that complies with MASVS L1 complies with mobile application security best practices. This level of validation provides basic requirements in terms of code quality, handling sensitive data, and interoperability with the mobile environment. This level is suitable for all mobile applications.

V1

Architecture, Design and Threat Modeling Requirements

The category “V1” lists requirements pertaining to architecture and design of the app. Besides the technical controls, the MASVS requires processes to be in place that ensure that the security has been explicitly addressed when planning the architecture of the mobile app, and that the functional and security roles of all components are known. Since most mobile applications act as clients to remote services, it must be ensured that appropriate security standards are also applied to those services - testing the mobile app in isolation is not sufficient.

V2

Data Storage and Privacy Requirements

The protection of sensitive data, such as user credentials and private information, is a key focus in mobile security. Firstly, sensitive data can be unintentionally exposed to other apps running on the same device if operating system mechanisms like IPC are used improperly. Data may also unintentionally leak to cloud storage, backups, or the keyboard cache. Additionally, mobile devices can be lost or stolen more easily compared to other types of devices, so an adversary gaining physical access is a more likely scenario. In that case, additional protections can be implemented to make retrieving the sensitive data more difficult.

V3

Cryptography Requirements

Cryptography is an essential ingredient when it comes to protecting data stored on a mobile device. The purpose of the controls in this chapter is to ensure that the verified application uses cryptography according to industry best practices, including: Use of proven

V4

Authentication and Session Management Requirements

In most cases, users logging into a remote service is an integral part of the overall mobile app architecture. Even though most of the logic happens at the endpoint, MASVS defines some basic requirements regarding how user accounts and sessions are to be managed.

cryptographic libraries; Proper choice and configuration of cryptographic primitives; A suitable random number generator wherever randomness is required.

V5

Network Communication Requirements

The purpose of the controls listed in this section is to ensure the confidentiality and integrity of information exchanged between the mobile app and remote service endpoints. At the very least, a mobile app must set up a secure, encrypted channel for network communication using the TLS protocol with appropriate settings.

V6

Environmental Interaction Requirements

The controls in this group ensure that the app uses platform APIs and standard components in a secure manner. Additionally, the controls cover communication between apps (IPC).

V7

Code Quality and Build Setting Requirements

The goal of this control is to ensure that basic security coding practices are followed in developing the app, and that "free" security features offered by the compiler are activated.