# TheWebOrion Web Security LAB REPORT

Vulnerability Assessment & Penetration Testing (VAPT)

## IOS Application Security Assessment Report

Application Name: [redacted]

# Table of Contents:

1. **Executive Summary..... 3**
2. **Project Scope..... 4**
3. **Findings Summary..... 5**
4. **High Severity Findings and Details... .....6**
    1. **App is allowed to run on jail broken devices.....6**
    2. **Insecure App transport security(ATS).....13**
5. **Medium Severity Findings and Details.....18**
    1. **Weak Hashing algorithm.....18**
    2. **Weak random number generator algorithm.....21**
    3. **Unsecure function used for memory allocation (Can cause memory leak if not properly implemented).....24**
6. **Low Severity Findings and Details.....28**
    1. **Use of banned API functions.....28**
    2. **Unencrypted app preferences and local storage.....31**
    3. **Unsecure poor cryptographic protocols.....36**
7. **Recommendations.....38**

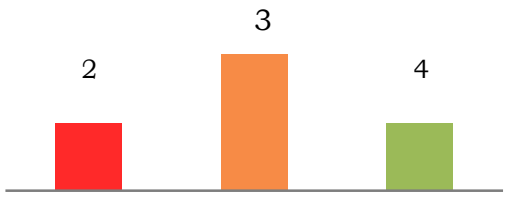


# 1. Executive Summary:

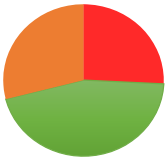
Timesheet engaged to conduct vulnerability assessment and penetration testing of its timesheet iOS application. The objective of this testing was primarily to identify security vulnerabilities in the provided iOS application and determine the level of depth that an attacker can penetrate, perform fraudulent or misuse application functionality.

Business Criticality of the Application: **HIGH**

**Severity based Vulnerability Distribution posture:**




**Impact to security**



## Key Findings and impact summary

1. **Insecure app transport security (ATS) has unsafe setting allowing the retrieval of content over insecure channels.**
2. **Weak Random number generator.**
3. **Unsecure and poor cryptographic protocols.**
4. **App is allowed to run on jail broken device.**
5. **Unencrypted app preferences and storage.**
6. **Weak hashing algorithm.**
7. **Use of banned API methods/functions.**
8. **Use of unsecure function malloc ().**



[redacted]

# 2. Project Scope:

| Property | Value |
|---|---|
| **Application Name** | **iOS App** [redacted] |
| **Description** | This application provides collaborative platform for making/editing Time-Sheet and/or schedule of different tasks. And It has connected with People Cloud! |
| **IOS App package name** | Timesheet: [redacted] |
| **Credentials** | **N/A** |
| **Test Scope** | **Black-Box** |
| **Duration** | **6 Days (POC)** |

# 3. Finding Summary:

| Finding | CVSS | Severity |
|---|---|---|
| App is allowed to run on jail broken device! | 8.1 | HIGH |
| Insecure app transport security (ATS) has unsafe setting allowing the retrieval of content over insecure channels. | 6.5 | HIGH |
| Weak Hashing algorithms. | 5.5 | MEDIUM |
| Weak Random number generator. | 4.8 | MEDIUM |
| Use of insecure method to allocate memory! | 4.0 | MEDIUM |
| Used of banned API methods/functions. | 3.6 | LOW |
| Unencrypted app preferences and storage | 3.4 | LOW |
| Unsecure cryptography protocols | 3.3 | LOW |



# 4. High severity findings and summary:

| **App is allowed to run on jail broken device!** | |
|---|---|
| Severity | **HIGH**<br><br>**OWASP MASVS: 8.1[R]**<br><br>**AV:N/AC:H/PR:N/UI:N/S:C/C:H/I:L/A:L** |
| Vulnerability | App is able to run on jail broken device. |
| Finding Description | Jail breaking is the process where the iOS system kernel is patched in order to remove the software restrictions (the "jail") imposed on Apple devices. It permits root access to the iOS file system, allowing the download of additional/pirated applications, Extensions and themes that are unavailable through the official Apple App Store At app level, jail breaking is used to obtain the access to all the apps sandboxes [e.g. Documents, Library, tmp folders]. At device level, jail breaking is needed in order to remove all the limitations imposed on the mobile by the carrier. |
| Method of Identification and Validation | Manual |
| Affected Resources | IOS App (../payload/) |
| Affected Thing | IOS Application |
| Evidence | Steps to reproduce :<br><br>Step 1: Go to source code of your IOS App.<br><br>Step 2: Run Xcode and open swift IDE OR you can use any Swift IDE to compile and run this code. |



Step 3: Just compile and run the swift program, Just make sure your phone is connected with PC/Mac and properly configured to use iTunes.

Step 4: In a non-jail broken environment applications can only read and write inside the application sandbox. If the application is able to access files outside of its sandbox, it's probably running on the jail broken device.

Step 5: Code:

```
if TARGET_IPHONE_SIMULATOR != 1
{
// Check 1 : existence of files that are common for jail broken devices

if FileManager.default.fileExists(atPath: "/Applications/Cydia.app")
[redacted]
|| FileManager.default.fileExists(atPath: "/bin/bash")
|| FileManager.default.fileExists(atPath: "/usr/sbin/sshd")
|| FileManager.default.fileExists(atPath: "/etc/apt")
|| FileManager.default.fileExists(atPath: "/private/var/lib/apt/")
||
UIApplication.shared.canOpenURL(URL(string:"cydia://package/com.example.package")!)
{
        return true
}

// Check 2 : Reading and writing in system directories (sandbox violation)

let stringToWrite = "Jailbreak Test"
```

```
do
{
        try stringToWrite.write(toFile:"/private/JailbreakTest.txt",
        encoding:String.Encoding.utf8)

        //Device is jail broken

        return true

}

catch
{
        return false
}

}

else
{
        return false
}
```

Step 6: Just run this code, If output shows Yes, then iPhone is Jail Broken if output says no, then device is not jail broken!

Step 7: Navigate through /payload/Info.plist, if it is ASCII or UTF-8 Encoded, then there is some problem. This file should be in binary format and cannot be opened by any text editor directly.



Step 8: Just see the below Code,

*NSBundle *bundle = [NSBundle mainBundle];*

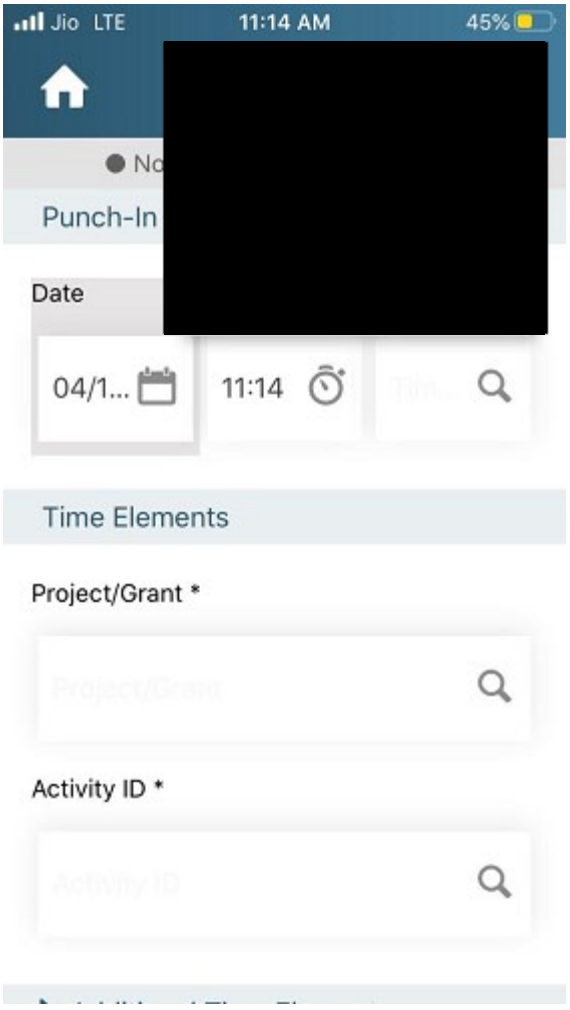
[redacted]

*{*

***/* do something */***

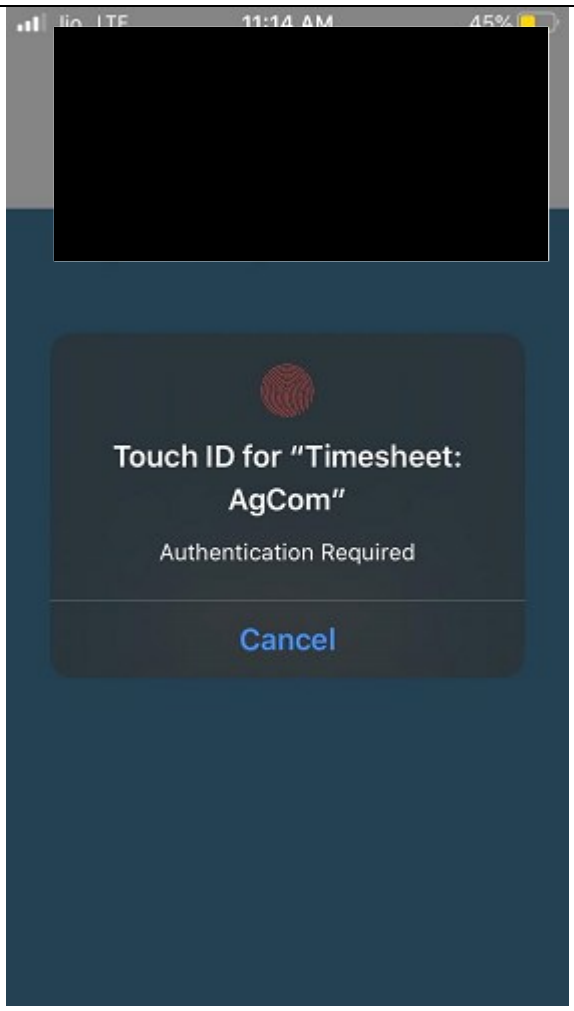
*}*

Step 9: Here see the above screenshot, this is how application looks like this.

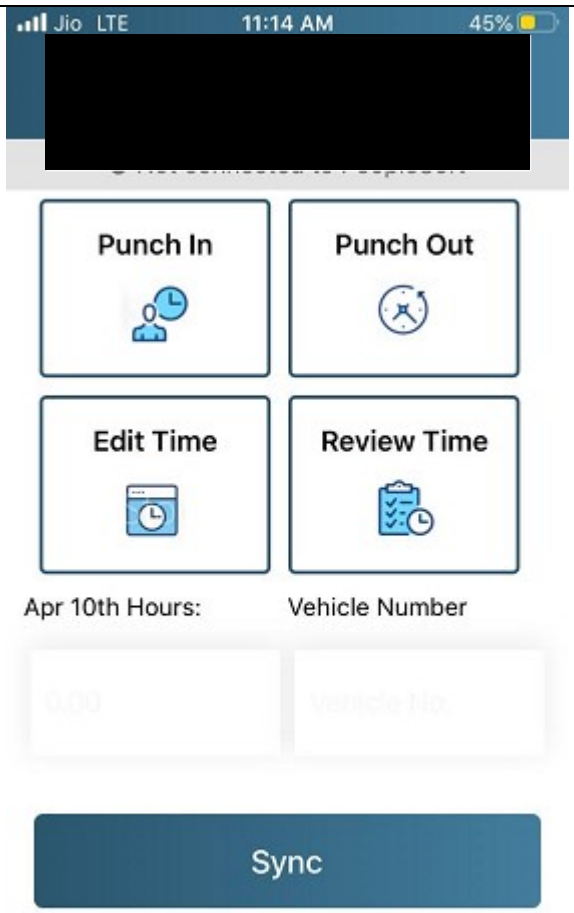





[redacted]




Step 10: As soon as you click on sign in, the application uses Apple's Touch ID/ FaceID, whichever is available for authentication purpose. After verifying this, you can see the home page of the main application. It requires to app shall be connected with PeopleSoft cloud.

| | |
|---|---|
| |  |
| Technical Impact | Cyber criminals target jail broken devices to inject or to install malware much more easily. Jail breaking is indeed the first step for simplifying the hacking procedures. |

| | |
|---|---|
| Business Impact | This App is allowed to run in without sandbox environment, it means cyber criminals and use this app and its resources to distribute malware and which directly affects customer's privacy. |
| Remediation | For Application Developers :-<br><br>Use Sandbox environment which is provided by Apple. App should be run with the help of sandboxing. So it can use limited resources and doesn't directly affects another resources.<br><br>Set some restrictions in installing App, App should check strictly whether the device is jail broken or not, and if it is so then app denied itself to run in devices.<br><br>For more information,<br><br>https://support.apple.com/en-in/HT201954<br><br>https://forums.developer.apple.com/thread/70603<br><br>https://forums.developer.apple.com/thread/66363 |



| **Insecure app transport security (ATS) has unsafe setting allowing the retrieval of content over insecure channels.** | |
|---|---|
| Severity | **HIGH**<br>**CVSS: 6.5**<br>**AV:L/AC:H/PR:N/UI:N/S:C/C:H/I:N/A:L** |
| Vulnerability | Insecure app transport security (ATS) |
| Finding Description | App Transport Security (ATS) enforces best practices in the secure connections between an app and its back end. ATS prevents accidental disclosure, provides secure default behavior, and is easy to adopt; it is also on by default in iOS 9 and OS X v10.11. You should adopt ATS as soon as possible, regardless of whether you're creating a new app or updating an existing one. |
| Method of Identification and Validation | Manual |
| Affected Resources | IOS App (./payload/Info.plist) |
| Affected Thing | **NSAllowsArbitraryLoads** property |
| Evidence | Steps to reproduce :<br><br>Step 1: Decompile IPA using comprehensive decompiler (Note: you can use any decompiler of your choice. For this you can also use built in compiler that is already in Xcode!)<br><br>Step 2: Extract source code folder (/Payload/FILES_LIST).<br><br>Step 3: Navigate through /payload/Info.plist file. |

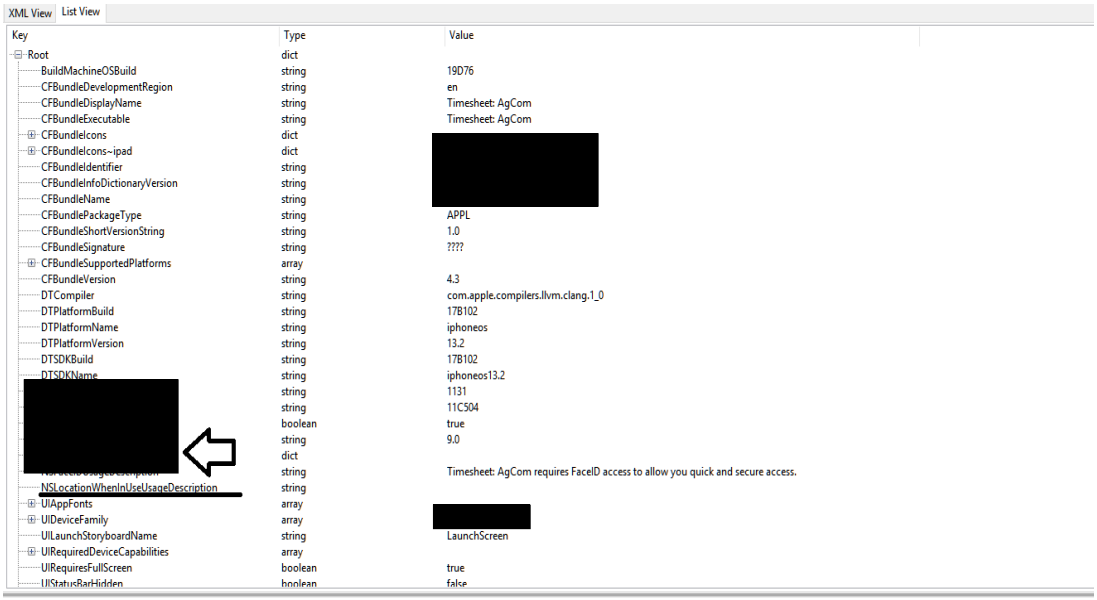


Step 4: Open using proper IDE like Plist Viewer.

Step 5: Now, you can see file is opened and go to **NSAllowArbitraryLoads and**

**NSExceptionAllowsInsecureHTTPLoads**, etc.

Step 6: Look at its value, it has Boolean type value. And for security reason, it marked as **True, Although it is for local host domain. But for other domain, this value should be marked as false.**




Step 7: See the highlighted code!

| | |
|---|---|
| Technical Impact | • **NSAllowsArbitraryLoads:** If set to YES, disables all ATS restrictions for all network connections, apart from the connections to domains that you configure individually in the optional NSExceptionDomains dictionary. Default value is NO.<br>• **NSAllowsArbitraryLoadsForMedia**: If set to YES, disables all ATS restrictions for media that your app loads using the AV Foundation framework. Employ this key only for loading media that are already encrypted, such as files protected by Fair Play or by secure HLS, and that do not contain personalized information. Default value is NO.<br>• **NSAllowsArbitraryLoadsInWebContent:** If set to YES, disables all ATS restrictions for requests made from web views. This lets your app use an embedded browser that can display arbitrary content, without disabling ATS for the rest of your app. Default value is NO.<br>• **NSExceptionAllowsInsecureHTTPLoads:** If set to YES, allows insecure HTTP loads for the named domain, but does not change Transport Layer Security (TLS) requirements and does not affect HTTPS loads for the named domain. Default value is NO.<br>• **NSExceptionMinimumTLSVersion:** Specifies the minimum TLS version for network connections for the named domain, allowing connection using an older, less secure version of Transport Layer Security. |
| Business Impact | If it is allowed for other remote domains like, example.com; the property **NSExceptionAllowsInsecureHTTPLoads** as well **NSAllowsArbitraryLoads,** attacker can easily load arbitrary malicious code that can bypass all security checks and leads to further attacks. Insecure method also cause serious security issue! |

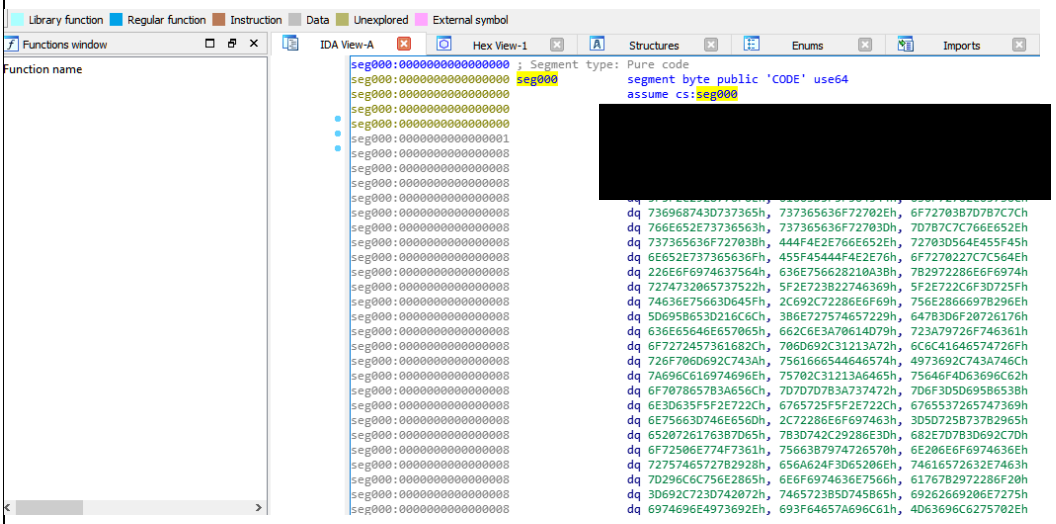


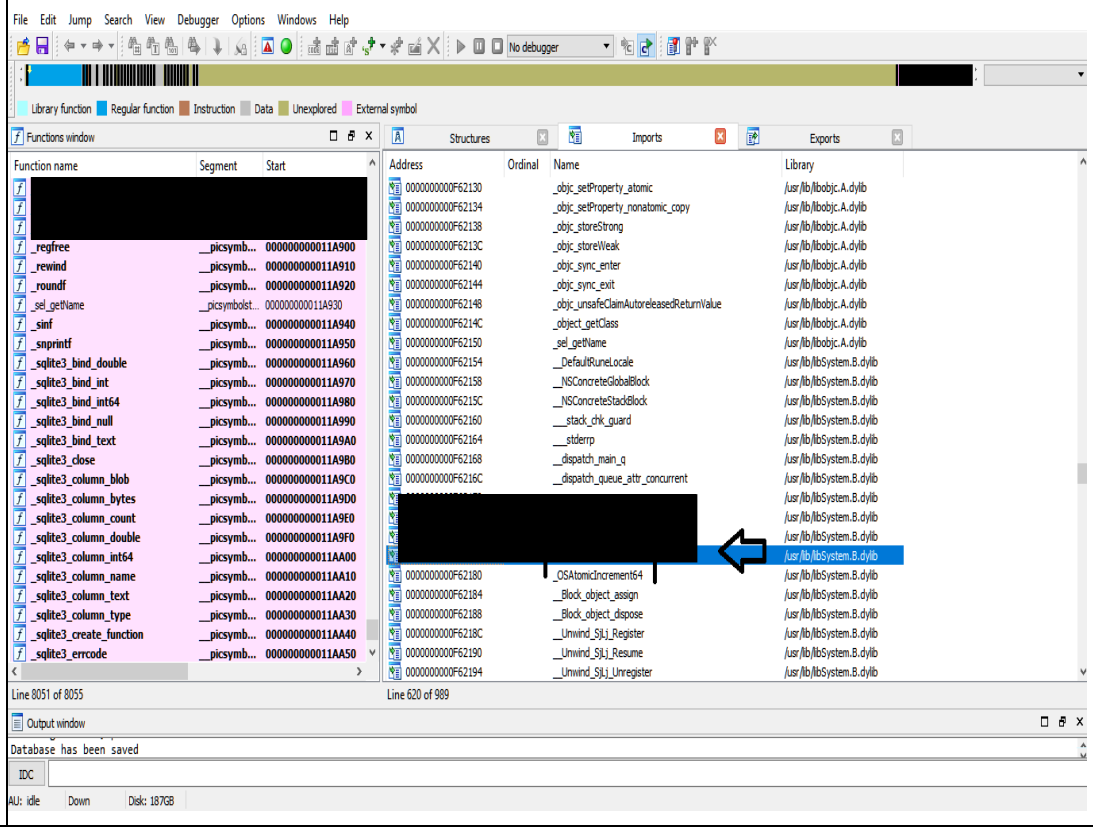
| | |
|---|---|
| Remediation | For Application Developers :-<br><br>1. Simply change the value of property **`NSExceptionAllowsInsecureHTTPLoads`** **false**, as it prevents to load content from unsafe Resources respectively.<br>2. And also change the property **`NSAllowsArbitraryLoads`** value to false. So it also prevents to load content from random and unsafe sources. Which may protect application.<br><br>For more information,<br><br>https://developer.apple.com/documentation/security/preventing_insecure_network_connections<br>https://developers.google.com/admob/ios/app-transport-security<br>https://developer.apple.com/documentation/bundleresources/information_property_list/nsapptransportsecurity |



# 5. Medium severity findings and summary:

| **Weak hashing algorithm** | |
|---|---|
| Severity | **MEDIUM**<br>**CVSS: 5.5**<br>**AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N** |
| Vulnerability | Weak hashing algorithm |
| Finding Description | The mobile application uses weak hashing algorithms. Weak hashing algorithms (e.g. MD2, MD4, MD5 or SHA-1) can be vulnerable to collisions and other security weakness, and should not be used when reliable hashing of data is required. |
| Method of Identification and Validation | Manual |
| Affected Resources | /payload/ios.app/BINARY_FILES |
| Affected Parameter | N/A |
| Evidence | Steps to Reproduce:<br>Step 1: Go to the extracted source,<br>Step 2: Decompile binary file named [redacted]<br>Step 3: Now open with any assembler or debugger of your choice.<br>Step 4: Find following by searching this,<br>Binary match usage of 'CC_SHA1' function or method.<br>Binary match usage of 'CC_MD5' function or method. |


Library function
Regular function
Instruction
Data
Unexplored
External symbol
Functions window
Function name
IDA View-A
Hex View-1
Structures
Enums
Imports
seg000:0000000000000000 ; Segment type: Pure code
seg000:0000000000000000 seg000 segment byte public 'CODE' use64
seg000:0000000000000000 assume cs:seg000
[redacted]
seg000:0000000000000008 dq 736968743D737365h, 737365636F72702Eh, 6F72703B7D7B7C7Ch
seg000:0000000000000008 dq 766E652E73736563h, 737365636F72703Dh, 7D7B7C7C766E652Eh
seg000:0000000000000008 dq 737365636F72703Bh, 444F4E2E766E652Eh, 72703D564E455F45h
seg000:0000000000000008 dq 6E652E737365636Fh, 455F45444F4E2E76h, 6F7270227C7C564Eh
seg000:0000000000000008 dq 226E6F6974637564h, 636E756628210A3Bh, 7B2972286E6F6974h
seg000:0000000000000008 dq 7274732065737522h, 5F2E723B22746369h, 5F2E722C6F3D725Fh
seg000:0000000000000008 dq 74636E75663D645Fh, 2C692C72286E6F69h, 756E28666697B296h
seg000:0000000000000008 dq 5D695B653D216C6Ch, 3B6E72757465729h, 647B3D6F20726176h
seg000:0000000000000008 dq 636E65646E657065h, 662C6E3A70614D79h, 723A79726F746361h
seg000:0000000000000008 dq 6F72724573616 82Ch, 706D692C31213A72h, 6C6C41646574726Fh
seg000:0000000000000008 dq 726F706D692C743Ah, 7561666544646574h, 4973692C743A746Ch
seg000:0000000000000008 dq 7A696C616974696Eh, 75702C31213A6465h, 75646F4D63696C62h
seg000:0000000000000008 dq 6F70786573B3A656Ch, 7D7D7D7B3A737472h, 7D6F3D5D695B6538h
seg000:0000000000000008 dq 6E3D635F5F2E722Ch, 6765725F5F2E722Ch, 67655372657473 69h
seg000:0000000000000008 dq 6E75663D746E656Dh, 2C72286E6F697463h, 3D5D725B737B2965h
seg000:0000000000000008 dq 652072617638 7D65h, 7B3D742C29286E3Dh, 682E7D7B3D692C7Dh
seg000:0000000000000008 dq 6F72506E774F7361h, 75663B7974726570h, 6E206E6F6974636Eh
seg000:0000000000000008 dq 727574657278 2928h, 656A624F3D65206Eh, 74616572632E7463h
seg000:0000000000000008 dq 7D296C6C756E2865h, 6E6F6974636E7566h, 61767B2972286F20h
seg000:0000000000000008 dq 3D692C723D742072h, 74657523B5D745B65h, 69262669206E7275h
seg000:0000000000000008 dq 6974696E4973692Eh, 693F64657A696C61h, 4D63696C6275702Eh


File Edit Jump Search View Debugger Options Windows Help
No debugger
Library function
Regular function
Instruction
Data
Unexplored
External symbol
Functions window
Function name
Segment
Start
[redacted]
_regfree __picsymb... 000000000011A900
_rewind __picsymb... 000000000011A910
_roundf __picsymb... 000000000011A920
_sel_getName __picsymbolst... 000000000011A930
_sinf __picsymb... 000000000011A940
_snprintf __picsymb... 000000000011A950
_sqlite3_bind_double __picsymb... 000000000011A960
_sqlite3_bind_int __picsymb... 000000000011A970
_sqlite3_bind_int64 __picsymb... 000000000011A980
_sqlite3_bind_null __picsymb... 000000000011A990
_sqlite3_bind_text __picsymb... 000000000011A9A0
_sqlite3_close __picsymb... 000000000011A9B0
_sqlite3_column_blob __picsymb... 000000000011A9C0
_sqlite3_column_bytes __picsymb... 000000000011A9D0
_sqlite3_column_count __picsymb... 000000000011A9E0
_sqlite3_column_double __picsymb... 000000000011A9F0
_sqlite3_column_int64 __picsymb... 000000000011AA00
_sqlite3_column_name __picsymb... 000000000011AA10
_sqlite3_column_text __picsymb... 000000000011AA20
_sqlite3_column_type __picsymb... 000000000011AA30
_sqlite3_create_function __picsymb... 000000000011AA40
_sqlite3_errcode __picsymb... 000000000011AA50
Line 8051 of 8055
Structures
Imports
Exports
Address
Ordinal
Name
Library
0000000000F62130 _objc_setProperty_atomic /usr/lib/libobjc.A.dylib
0000000000F62134 _objc_setProperty_nonatomic_copy /usr/lib/libobjc.A.dylib
0000000000F62138 _objc_storeStrong /usr/lib/libobjc.A.dylib
0000000000F6213C _objc_storeWeak /usr/lib/libobjc.A.dylib
0000000000F62140 _objc_sync_enter /usr/lib/libobjc.A.dylib
0000000000F62144 _objc_sync_exit /usr/lib/libobjc.A.dylib
0000000000F62148 _objc_unsafeClaimAutoreleasedReturnValue /usr/lib/libobjc.A.dylib
0000000000F6214C _object_getClass /usr/lib/libobjc.A.dylib
0000000000F62150 _sel_getName /usr/lib/libobjc.A.dylib
0000000000F62154 __DefaultRuneLocale /usr/lib/libSystem.B.dylib
0000000000F62158 __NSConcreteGlobalBlock /usr/lib/libSystem.B.dylib
0000000000F6215C __NSConcreteStackBlock /usr/lib/libSystem.B.dylib
0000000000F62160 ___stack_chk_guard /usr/lib/libSystem.B.dylib
0000000000F62164 ___stderrp /usr/lib/libSystem.B.dylib
0000000000F62168 __dispatch_main_q /usr/lib/libSystem.B.dylib
0000000000F6216C __dispatch_queue_attr_concurrent /usr/lib/libSystem.B.dylib
[redacted] /usr/lib/libSystem.B.dylib
0000000000F62180 _OSAtomicIncrement64 /usr/lib/libSystem.B.dylib
0000000000F62184 __Block_object_assign /usr/lib/libSystem.B.dylib
0000000000F62188 __Block_object_dispose /usr/lib/libSystem.B.dylib
0000000000F6218C __Unwind_SjLj_Register /usr/lib/libSystem.B.dylib
0000000000F62190 __Unwind_SjLj_Resume /usr/lib/libSystem.B.dylib
0000000000F62194 __Unwind_SjLj_Unregister /usr/lib/libSystem.B.dylib
Line 620 of 989
Output window
Database has been saved
IDC
AU: idle
Down
Disk: 187GB



[redacted]

| | |
|---|---|
| Technical Impact | Weak cryptographic hashes are susceptible to attacks like rainbow table searches. Hashing algorithms like MD5 and SHA-1 has been marked obsolete according to latest coding standards. This risk the integrity of security critical data to be compromised. |
| Business Impact | It directly connected with internal core function of an IOS Application. If algorithm is weak then it is vulnerable to rainbow table attack. While looking into application, malicious hacker can easily exploit weak algorithms and all business logic of app can be leaked due to this vulnerability. So that It is considered as critical business logic weakness! |
| Remediation | For Application Developers,<br>Implement strong hash functions with strongest available algorithms. Like 3DES, AES-128 Bit, etc.<br>For more information,<br>https://developer.apple.com/library/content/documentation/Security/Conceptual/cryptoservices/GeneralPurposeCrypto/GeneralPurposeCrypto.html |



| **Weak Random number generator.** | |
|---|---|
| Severity | **MEDIUM**<br><br>**OWASP MASVS: 4.8 [L1, L2]**<br><br>**AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:N** |
| Vulnerability | Weak random number generator |
| Finding Description | Developers generally implement random number generators [RNGs] where the random number is fully determined by the seed knowledge. This is the reason why they are called pseudo-random number generators [PRNGs]. When it comes to cryptography random numbers play a fundamental role in: key generation nonces one-time pads salts in certain signature schemes. |
| Method of Identification and Validation | Manual |
| Affected Resources | RNGs(Random Number Generator) |
| Affected Parameter | N/A |
| Evidence | Steps to Reproduce:<br><br>Step 1: Example Code for PSRNGs (Pseudo Random number generators) that is basically weak.<br><br>Step 2: Code,<br><br>**FILE *fp = fopen("/dev/random", "r");**<br>**if (!fp)**<br>**{** |

```
  perror("randgetter");
  exit(-1);
}
uint64_t value = 0;
int i;
for (i=0; i<sizeof(value); i++)
[redacted]
}
fclose(fp);
```

Step 3: The below is the example of strong PRNGs.

Step 4: Code,

```
uint8_t randomBytes[16];
int result = SecRandomCopyBytes(kSecRandomDefault, 16, randomBytes);
[redacted]
 NSLog(@"uuidStringReplacement is %@",     uuidStringReplacement);
}       ■
else
{
  NSLog(@"SecRandomCopyBytes failed for some reason");
}
```

Step 5:  Implement strong RNGs, like above one!



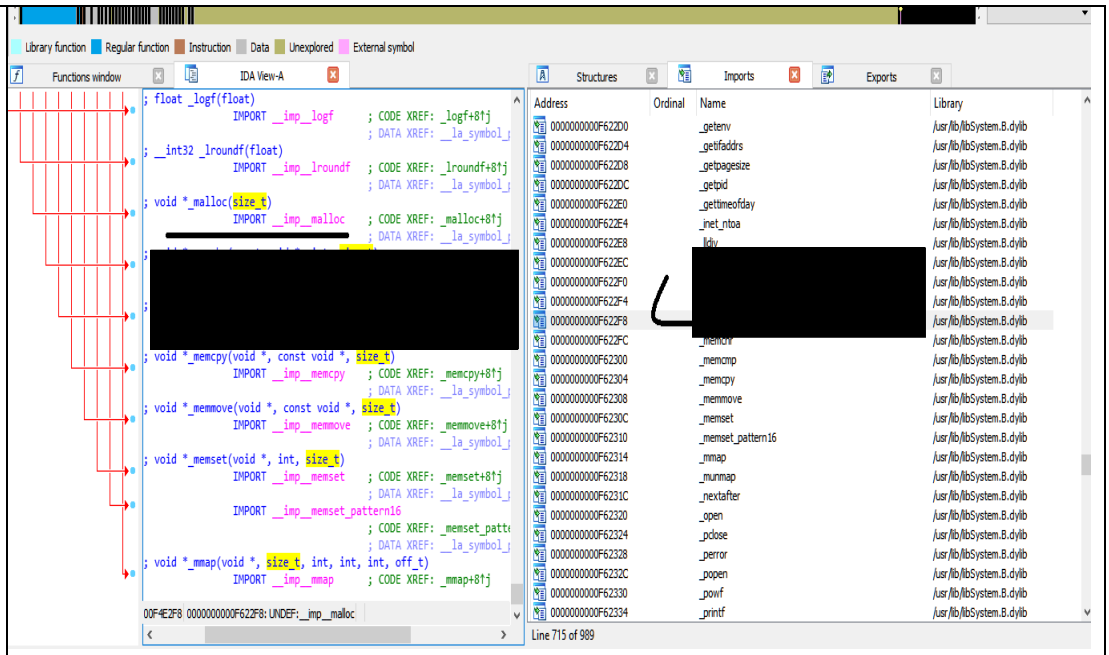
[redacted]

| | |
|---|---|
| Technical Impact | Using standard PRNGs is a bad practice when implementing security mechanisms, since the attacker may be able to guess the logic behind and predict the generated random numbers. In this case the confidentiality and/or integrity of the vulnerable app might be undermined. Under certain conditions this weakness may jeopardize mobile application data encryption or other protection based on randomization. For example, if encryption tokens are generated inside of the application and an attacker can provide application with a predictable token to validate and then execute a sensitive activity within the application or its backend. |
| Business Impact | It directly connected with internal core function of an IOS Application. If RNGs generate number that is not enough to strong, then while looking into application, malicious hacker can easily exploit weak RNGs and all business logic of app can be leaked due to this vulnerability. So that It is considered as critical business logic weakness! |
| Remediation | For Application Developers,<br>Implement strong PSRNGs with strongest available algorithms. Like 3DES, AES-128 Bit, etc. For function that generate PSRNGs, provide protected attribute to limit function usage and prevent return wrong values.<br>https://developer.apple.com/documentation/security/1399291-secrandomcopybytes<br>https://developer.apple.com/documentation/security/ksecrandomdefault |



| **Use of insecure function for memory allocation (If not implemented properly then it causes buffer overflows and memory leak).** | |
|---|---|
| Severity | **MEDIUM**<br><br>**CVSS: 4.0**<br><br>**AV:L/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:N** |
| Vulnerability | Use of insecure function malloc() to allocate memory |
| Finding Description | The mobile application uses malloc() function to allocate new memory instead of more secure calloc(), thus endangering application privacy under certain circumstances (e.g. if freed memory can be accessed by an attacker). |
| Method of Identification and Validation | Manual |
| Affected Resources | /payload/ios.app |
| Affected Parameter | Memory allocation |
| Evidence | Steps to Reproduce:<br><br>Step 1: Go to extracted source,<br><br>Step 2: As done in earlier stage, open decompiled source with desired IDE as well as Reverse engineering tool.<br><br>Step 3:<br>Open search bar, and search for malloc, |

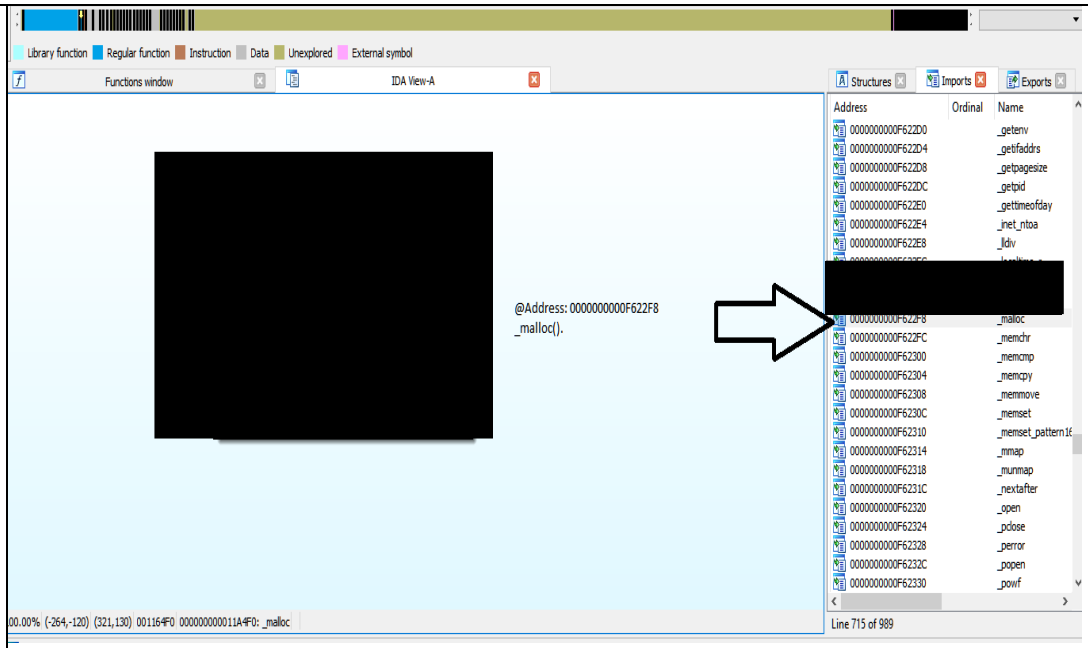




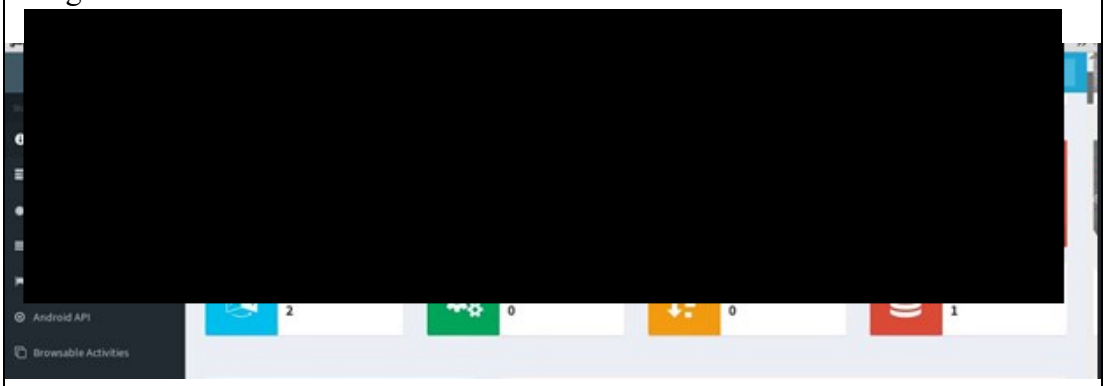

Step 4: As you can see in the above image, the highlighted portion is for malloc(), which is used to distribute and allocate memory but that has no protection against buffer overflow, NULL pointer reference and memory leak, so it is risky to use this function. And Apple has also made restricted use of this function.

Step 5: Instead of this, you can use Calloc(), which has same purpose but with enforced security.

Step 6: calloc () provides secure allocation that has limited access to required resources. So memory leaked don't happen.




Step 7: Here how developer had declared this function, Any pointer or variable outside this can access this malloc () property value arbitrarily. Which is very dangerous.




| | |
|---|---|
| Technical Impact | When malloc () has incorrectly implemented, memory leak has happened and memory leak reduces the performance of the computer by reducing the amount of available memory. Eventually, in the worst case, too much of the available memory may become allocated and all or part of the system or device stops working correctly, the application fails, or the system slows down vastly due to thrashing. |

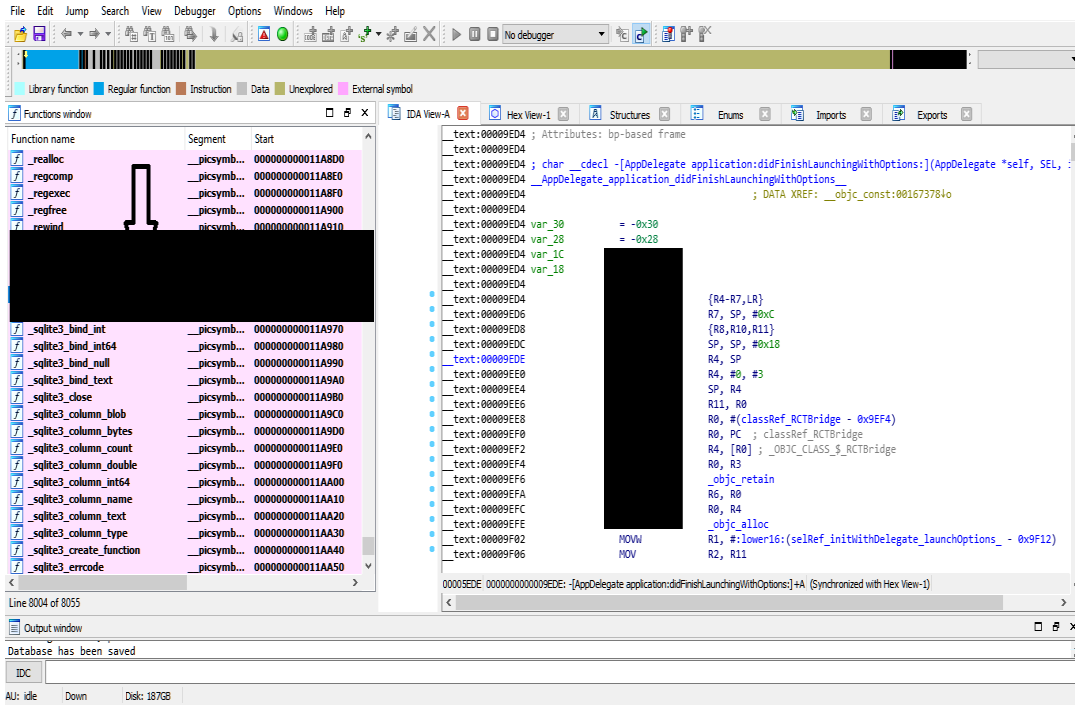
| | |
|---|---|
| Business Impact | Due to memory leak, app may slow down and it stops running smoothly and if buffer overflow also happened then it affects other app's performance. So this problem can be solved earlier as soon as possible. |
| Remediation | For Application Developers,<br><br>https://developer.apple.com/library/archive/documentation/Performance/Conceptual/ManagingMemory/Articles/MemoryAlloc.html<br><br>https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneOS/man3/calloc.3.html<br><br>https://forums.developer.apple.com/thread/108319 |

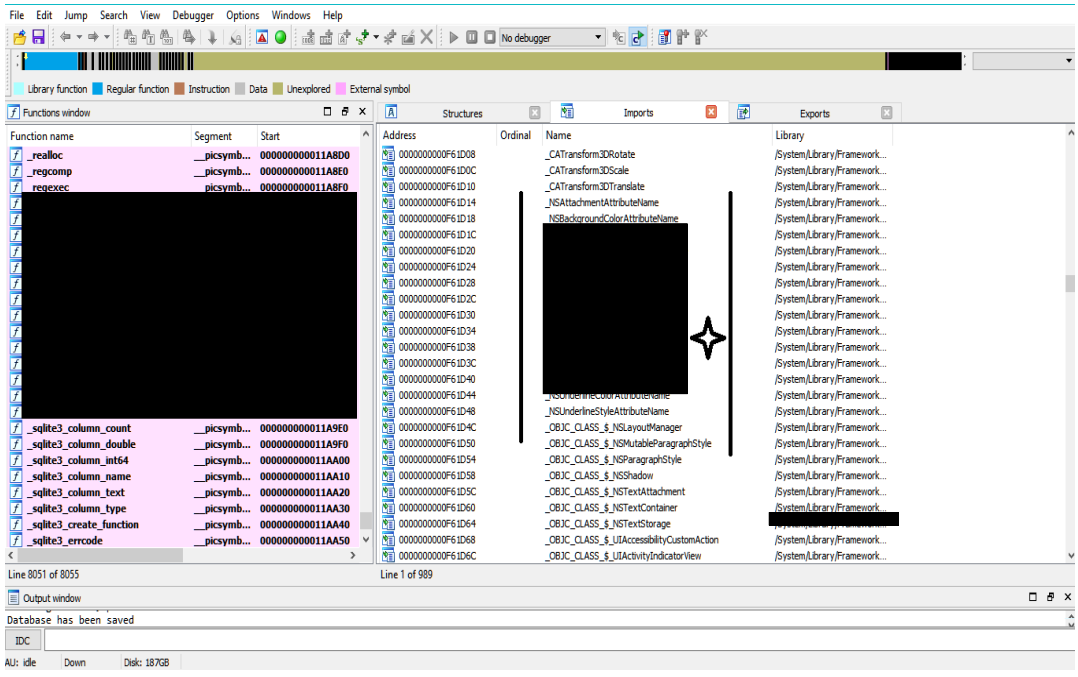


# 6. Lower severity findings and summary:

| **Usage of banned API functions.** | |
|---|---|
| Severity | **LOW**<br>**CVSS: 3.6**<br>**AV:L/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:N** |
| Vulnerability | Usage of banned API methods |
| Finding Description | The mobile application uses some of the banned API functions. API functions are usually banned for compelling security and privacy reasons and shall not be used for better security. |
| Method of Identification and Validation | Manual |
| Affected Resources | API Methods |
| Affected Parameter | N/A |
| Evidence | Steps to Reproduce:<br>Step 1: Go to the extracted source<br>,<br>Step 2: Decompile binary file named [redacted]<br>Step 3: Now open with any assembler or debugger of your choice.<br>Step 4: Find following by searching this,<br>You will find following functions,<br>**Alloca , memcpy, printf, Sprintf , strcat**<br>**Strcpy, strncpy, vsnprintf, gets** |

## Scanf, sscanf, strlen, wcslen, etc.

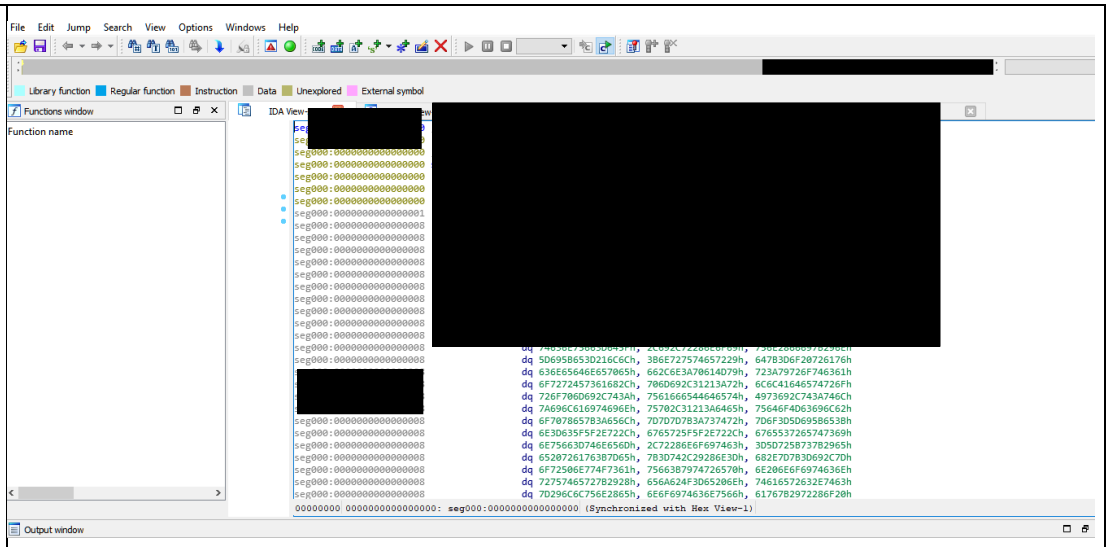


| | |
|---|---|
| Technical Impact | Apple has banned some functions that are vulnerable or inappropriate for some app architecture. So the use of these functions are restricted. And for performance improvement and security reasons, use methods or functions that are suggested by Apple itself. So it is not recommend to use banned methods or functions. |
| Business Impact | As said in the technical impact, the trust of app may be decrease due to use of banned things. And it directly effects on the app ratings and users. So we advised not to use of these functions. |
| Remediation | For Application developers,<br>Refer following documentations for more information,<br>https://msdn.microsoft.com/en-us/library/bb288454.aspx<br>https://developer.apple.com/library/content/documentation/security/Conceptual/SecureCodingGuide/Articles/BufferOverflows.html<br>https://developer.apple.com/library/content/documentation/security/Conceptual/SecureCodingGuide/SecurityDevelopmentChecklists/SecurityDevelopmentChecklists.html#//apple_ref/doc/uid/TP40002415-CH1-SW1 |



| **Unencrypted app preferences and storage** | |
|---|---|
| Severity | **LOW**<br><br>**OWASP MASVS: 3.4 [L1, L2]**<br><br>**AV:L/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N** |
| Vulnerability | Unsecure App preferences |
| Finding Description | Preferences are pieces of information used to configure the appearance and behavior of an app. Most of the preferences are persistently stored locally using the Cocoa preferences system—known as the User Defaults system. |
| Method of Identification and Validation | Manual |
| Affected Resources | App Preferences |
| Affected Parameter | N/A |
| Evidence | Steps to Reproduce:<br><br>Step 1: [redacted] well /payload<br><br>Step 2: Open any Debugger of your choice.<br><br>Step 3: Open contents inside symbol table file respectively. And open in Debugger. |

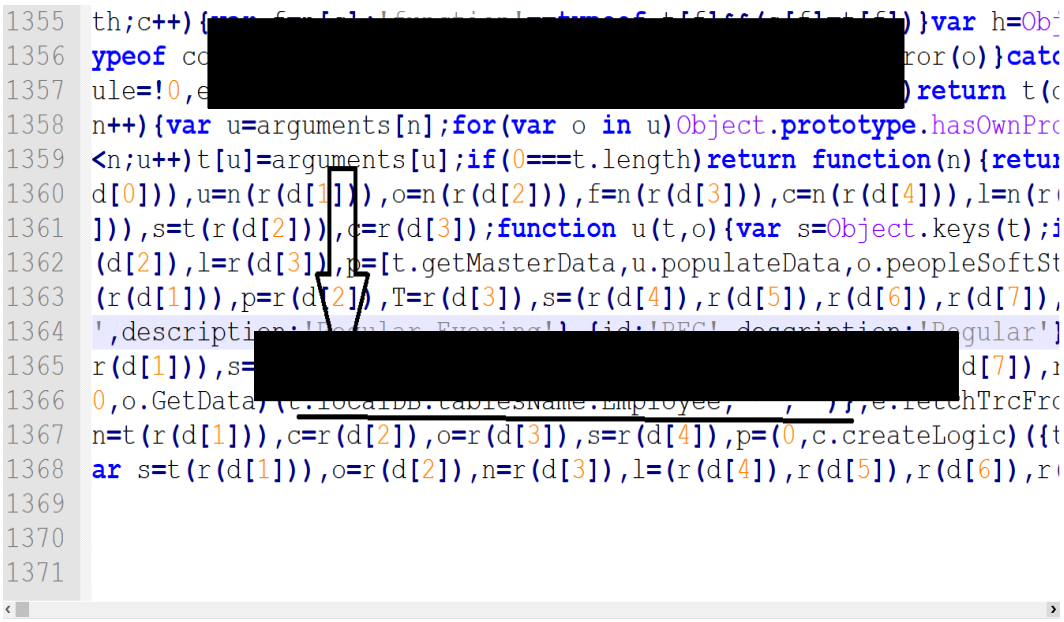


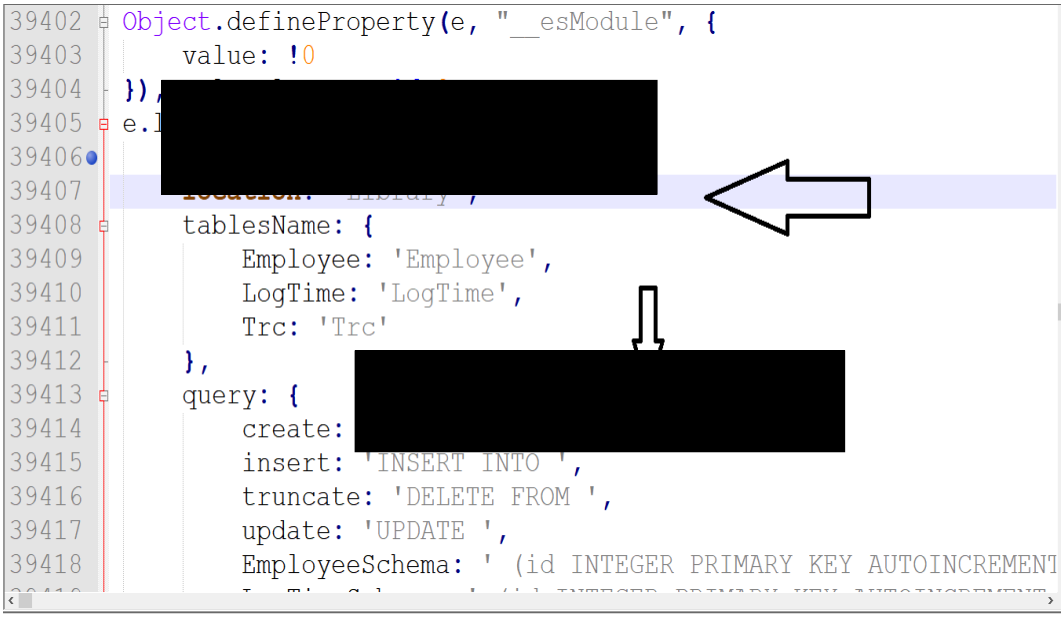

Step 4: See the below image,

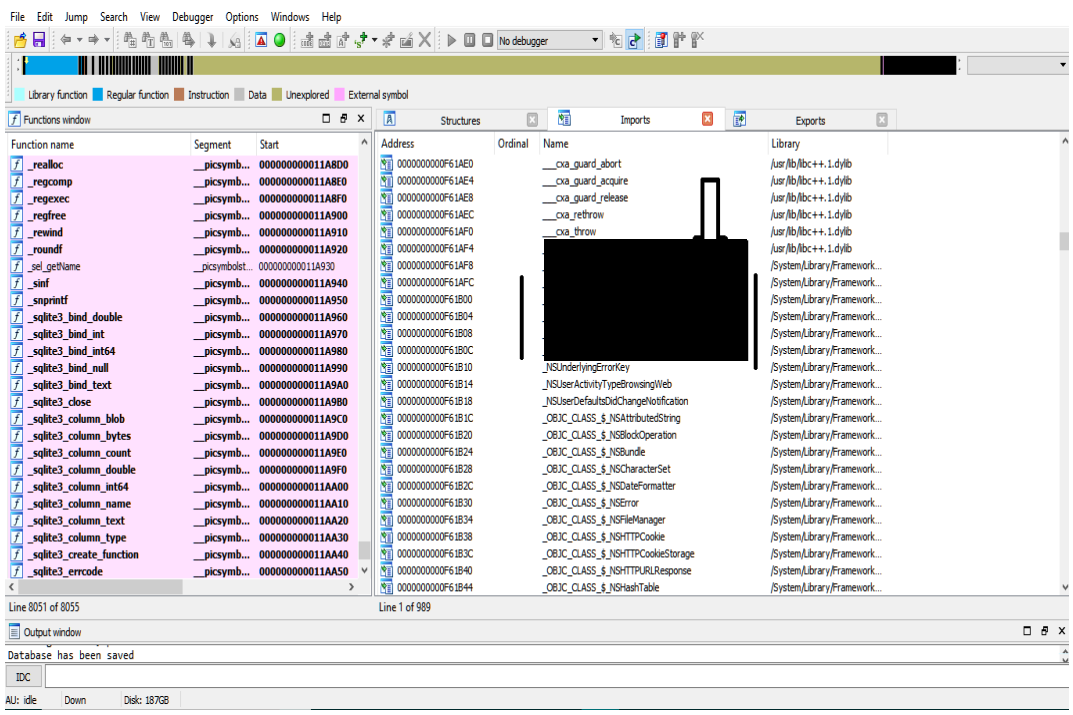
```
574 _kSecSharedPassword
575 _NSOSStatusErrorDomain
576 _
577 __
578 _U
579 _SecRequestSharedWebCredential
580 _UIAccessibilityAnnouncementDidFinishNotification
581 __ZNSt13runtime_errorC1ERKNSt3__112basic_stringIcNS0_11char_traits
582 _SecItemCopyMatching
583 _OBJC_CLASS_$_NSLock
584 
585 
586 
587 _OBJC_CLASS_$_NSMutableCharacterSet
588 __ZTSU13block_pointerFbRKNSt3__112basic_stringIcNS_11char_traitsIc
589 __ZNSt3__112basic_stringIcNS_11char_traitsIcEENS_9allocatorIcEEEC1
590 __ZNSt3__113basic_ostreamIcNS_11char_traitsIcEEElsEi
```

Step 5: See the highlighted _NSAttrib property! Which is for App preferences and this is simply encoded without any encryption.

Step 6: app preferences file should be encrypted using high-grade encryption algorithms to prevent unwanted behavior.





| | |
|---|---|
| |  |
| Technical Impact | Due to unencrypted or unsecure app preferences, malicious user can modify preferences by changing or adding extra attributes that may cause unwanted behavior of application and compromization of data! This may change application behavior. |
| Business Impact | As described in technical impact, after changing application behavior, malicious user can use for their purpose. And can inject code, and end user unaware of it. So he/she didn't notice this unknown behavior. And it further leads to compromise of sensitive data of customer. |

| | |
|---|---|
| Remediation | For Application Developers,<br><br>Encrypt files that stores Default app preferences and/or configurations with strongest possible algorithm to protect user preferences.<br><br>For more information,<br><br>https://developer.apple.com/documentation/appkit/nscolor/1524856-controlcolor<br><br>https://developer.apple.com/documentation/appkit/nstextfield<br><br>https://developer.apple.com/documentation/uikit/protecting_the_user_s_privacy/encrypting_your_app_s_files |

| Unsecure and poor cryptographic protocols | |
|---|---|
| Severity | **LOW**<br>**OWASP MASVS: 3.3 [L1, L2]**<br>**AV:L/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N** |
| Vulnerability | Unsecure and poor cryptographic protocols |
| Finding Description | A weak cipher is defined as an encryption/decryption algorithm that is unsafe, and likely because it uses a key of insufficient length. Using an insufficient length for a key in an encryption/decryption algorithm opens up the possibility for that encryption scheme to be cracked So in general, the larger the key size the stronger the cipher. Weak ciphers usually use key sizes that are not less than 128 bits in length. |
| Method of Identification and Validation | Manual |
| Affected Resources | N/A |
| Affected Parameter | N/A |
| Evidence | N/A<br>This is just informational purpose. No major risk is there. But should be corrected. |

| | |
|---|---|
| Technical Impact | A **weak cipher** is defined as an encryption/decryption algorithm that uses a key of insufficient length. Using an insufficient length for a key in an encryption/decryption algorithm opens up the possibility (or probability) that the encryption scheme could be broken (i.e. cracked). |
| Business Impact | N/A |
| Remediation | For Application Developers,<br><br>Just implement and use strong cipher suite.<br><br>For more information,<br><br>https://developer.apple.com/videos/play/wwdc2019/709/<br><br>https://developer.apple.com/library/archive/documentation/Security/Conceptual/SecureCodingGuide/Introduction.html<br><br>https://developer.apple.com/documentation/security<br><br>https://developer.apple.com/library/archive/documentation/Security/Conceptual/cryptoservices/SecureNetworkCommunicationAPIs/SecureNetworkCommunicationAPIs.html |

# 7. Our recommendations

- Refer Apple security standards and new security methods for implementing strong app.
- Upgrade to latest packages, security updates and configurations for better security.
- Make sure to check for business logic remains confidential to respected company, it should not be leaked in any forms, whether it is application or website.
- App needs to store user preferences and other default settings in order to run seamlessly, so make sure you've make encryption for this purpose. In other words make sure that app configurations is encrypted using strong encryption methods.
- Do not allow insecure source, in which app is loaded from there, respectively.
- As apple announce, in 2017, a security update to Apple's operating systems removed support for SHA-1 signed certificates used for Transport Layer Security (TLS) in Safari and Web Kit. Make sure to use SHA-256 signed certificates.
- ATS establishes best-practice policies for secure network communications using Apple platforms, employing Transport Layer Security (TLS) version 1.2, forward secrecy, and strong cryptography. So make sure you don't forgot to implement this!
- To prevent app modification, implement self-defense mechanism to make your app more secure. Which prevents unknown users to make any system level changes. And don't allow your app to be run on Jail-broken devices, which will risks your business!