DataArt

FinancialSrides

[redacted] iOS mobile app (v1.7.67) & related web service (APIs)

Penetration Testing Report

Prepared by:

[redacted] [redacted]@dataart.com)

Reviewed by:

Justin Surman (justin.surman@finstrides.com)

Limitations on Disclosure and Use of this Document

This report was prepared by DataArt for the exclusive benefit of Financial Strides and is proprietary information. Unauthorized use or reproduction of this document is prohibited. The Non-Disclosure Agreement (NDA) in effect between DataArt and Financial Strides governs the disclosure of this report to all other parties, including product vendors or suppliers

This report contains information about potential vulnerabilities of the [redacted] iOS application & related web service (APIs) and methods for exploiting them. DataArt recommends that special precautions be taken to protect the confidentiality of both document and the information contained herein. DataArt has retained and secured a copy of the document for customer reference. All other copies of the document have been delivered to Financial Strides.

By providing this report to Financial Strides, DataArt does not constitute any form of representation, warranty, or guarantee that the systems are 100% secure from every form of attack. While DataArt's methodology includes both automated and manual testing to identify and attempt exploitation of the most common security issues, testing was limited to an agreed upon time frame.

Document Details

Title	**Penetration Testing Report**
Project Name	[redacted] iOS mobile app (v1.7.67) & Supporting Banking Web Service (API)
Project Staff	[redacted] – Security Consultant [redacted], Justin Surman - PM & Peer Reviewers
Project Duration	15/05/2020 – 22/05/2020

Document History

Version	**Date**	**Author**	**Comments**
1.0	20/05/2020	[redacted]	Initial Version
1.1	21/05/2020	[redacted]	Internal Peer Review
1.2	22/05/2020	Justin Surman	Review and Corrections
1.3	28/05/2020	[redacted]	Verification of remediated issues (H1, M4, L1, L5, M1(partly), M2(partly), M3(partly), M5(partly), M6(partly))
1.4	28/05/2020	[redacted] Justin Surman	Peer Review and Corrections

Contents

Executive Summary

Financial Strides engaged DataArt to perform a penetration testing of the [redacted] native iOS application & related web service APIs, focusing on the newly supported [redacted] banking function/services that have been added to the iOS application in scope. The application's functionality includes quick funding, cash flow tools and digital banking services.

The primary goal of this mobile application (Grey box) penetration testing project was to identify any potential areas of concern associated with the application in its current state and determine the extent to which the system may be breached by an attacker possessing a particular skill and motivation. The assessment was performed in accordance with the "best-in-class" practices as defined by ISECOM`s Open Source Security Testing Methodology Manual (OSSTMM), Open Web Application Security Project (OWASP) and Penetration Test Guidance for PCI DSS Standard.

DataArt conducted penetration testing during the period of May 15 – 21, 2020. All testing activities were performed using the latest build version of the iOS mobile application (version 1.7.67) provided by the [redacted] via the Apple TestFlight service. While performing the testing activities, DataArt emulated an external attacker without prior knowledge of the environment. To test the user-authenticated area and access control privilege escalation vulnerabilities, DataArt used credentials provided by [redacted] for different user accounts. The testing did not attempt any active network-based DoS attacks.

The scope of the assessment included the following:

- [redacted] iOS application (v.1.7.67);
- Related web services:
  - api [redacted] com
  - [redacted] com
  - cognito-idp.us-west-2.amazonaws.com

Note: it should be noted that all testing activities were performed against two different builds of the application delivered via TestFlight. The first build v.1.7.67 (2) was provided by the customer on May 15th and contained the functional issue prohibiting utilization of AR face tracking feature on the device utilized in the account opening workflow. The second build v.1.7.67 (3) of the application was provided on May 20th and contained appropriate fixes for the broken functionality.

During the course of the assessment, DataArt identified two high-risk security issues which can lead to full compromise of application users` data:

- Access control issues.
  The application did not properly verify a user`s permissions for a part of functionality. As an example of exploitation of the issue, a malicious user could transfer money from other accounts or obtain sensitive data belonging to other users.
- Missing multi-factor authentication.
  Multi-factor authentication is recognized as the best defense against the majority of password-related

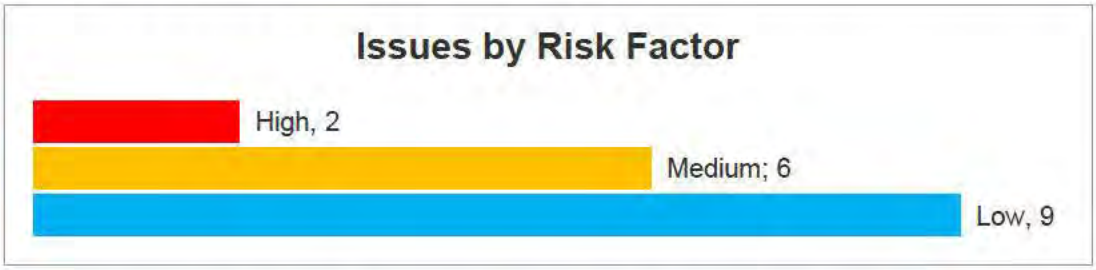
attacks, including brute-force, credential stuffing and password spraying. In case victim`s credentials or a physical device is stolen, an attacker can easily gains the full access over the victim`s account and related financial data.

DataArt also identified a number of medium and low severity security issues which could be used by an attacker within his malicious activities.

As of 27/05/2020, the [redacted] development team remediated part of the identified issues. Detailed information can be found in the "Issues Remediation" section of this report.

DataArt strongly recommends [redacted] to remediate all high and medium severity issues detected to mitigate against the possible risk of a sensitive data compromise. The remediation of the low severity findings is not so urgent due to the low probability of their successful exploitation. Nonetheless, it should be noted that the existence of these known issues could decrease the overall security posture of the system.

This report summarizes what DataArt believes are the most important issues to address in the tested application. The chart below outlines a number of issues identified that are grouped by risk factors. Note the risk ratings were given to help assist in prioritizing remediation efforts. True risk can only be calculated by an in-depth understanding of business processes and data, as well as the likelihood of exploitation.



The tables below summarize the findings for OWASP Mobile Top 10 list for mobile application vulnerabilities and OWASP API Top 10 list for the server side API used by the application. Both lists represent the most critical mobile/API security flaws, which are accompanied by OWASP security experts from around the world. The lists provide powerful awareness documents for native/server side application security and are utilized within many security standards:

OWASP Mobile Top 10	
Category	**Discovered**
A1: Improper Platform Usage	**YES**
A2: Insecure Data Storage	**YES**
A3: Insecure Communication	**YES**
A4: Insecure Authentication	**YES**
A5: Insufficient Cryptography	**YES**
A6: Insecure Authorization	**YES**

A7: Client Code Quality	NO
A8: Code Tampering	YES
A9: Reverse Engineering	YES
A10: Extraneous Functionality	NO

OWASP API Top 10	
Category	Discovered
API1: Broken Object Level Authorization	YES
API2: Broken User Authentication	NO
API3: Excessive Data Exposure	NO
API4: Lack of Resources & Rate Limiting	NO
API5: Broken Function Level Authorization	YES
API6: Mass Assignment	NO
API7: Security Misconfiguration	YES
API8: Injection	NO
API9: Improper Assets Management	NO
API10: Insufficient Logging & Monitoring	NO

DataArt can re-verify the [redacted] remediated issues found during this penetration test within 60 days of this report delivery (**until July 25th, 2020**).

Issues Remediation

May 27, 2020

DataArt verified remediated issues on May 27, 2020. The updated version of the tested application (build v1.7.67 (4)) was received from the development team via TestFlight.

As of **27/05/2020**, the following issues were fully remediated:

- **H1. Missing function-level access control**
- **M4. Weak password requirements**
- **L1. The application server allows unencrypted communications**
- **L5. Verbose error messages**

The following issues have been partly remediated but are still actual:

- **M1. Responses caching is enabled**
- **M2. Unencrypted storage of sensitive data**
- **M3. Missing certificate validation**
- **M5. Enumeration of registered users**
- **M6. Insecure versions of TLS protocol are supported**

All other issues are still actual:

- **H2. Missing multi-factor authentication**
- **L2. Missing password changing functionality**
- **L3. Lack of binary protection**
- **L4. Usage of weak hash algorithms**
- **L6. Platform information is disclosed in server's responses**
- **L7. Using components with known vulnerabilities**
- **L8. Support of TLS ciphers suites without forward secrecy**
- **L9. Missing debugger detection**

Assessment Methodology

DataArt based the finding and recommendations, outlined in this report, on manual penetration testing performed against the mobile application.

Native Application Testing

DataArt developed a documented, proprietary methodology for conducting penetration testing of native applications. The testing included several tasks such as reverse-engineering application logic and security controls, dynamic application analysis, inspection of application traffic and locally stored data.

The penetration test focused on possible vulnerabilities in the applications logic, looking for issues including but not limited to:

- Local data storage
- Caching and temporary files
- Logging
- Privacy issues
- Information leakage
- WebView vulnerabilities
- SSL and transport layer weaknesses
- Authentication and session management defects
- Unmanaged code and memory access
- Memory leaks

During the mobile testing, DataArt also uncovered server-side APIs used by the application. DataArt specifically focused on the API methods used to support the newly integrated banking function/services; these API methods were carefully investigated for potential security issues which are included in this current report.

Criteria for Risk Ratings

The table below outlines the general rules for assigning risk ratings to identified vulnerabilities:

Risk Rating	Description
HIGH	These issues identify conditions that could directly result in the compromise or unauthorized access of a network, system, application or sensitive information. Examples of High-Risk issues include remote execution of commands, known buffer overflows; unauthorized access and disclosure of sensitive information.

MEDIUM	These issues identify conditions that do not immediately or directly result in the compromise or unauthorized access of a network, system, application of information, but do provide a capability or information that could, in combination with other capabilities or information, result in the compromise or unauthorized access of a network, application or information. Examples of Medium Risk issues include directory browsing, partial access to files on the system; disclosure of security mechanisms and unauthorized use of services.
LOW	These issues identify conditions that do not immediately or directly result in compromise of a network, system, application or information, but do provide information that could be used in combination with other information to gain insight into how to compromise or gain unauthorized access to a network, system, application or information.
REMEDIATED	The issue was fully remediated since the previous round of penetration testing
PARTLY REMEDIATED	The issue was partly remediated since the previous round of penetration testing

Assessment Findings

Summary

The table below outlines summary of findings identified during the penetration testing:

Finding Description	Status
High Risk Findings	
H1. Missing function-level access control	~~HIGH~~ REMEDIATED
H2. Missing multi-factor authentication	HIGH
Medium Risk Findings	
M1. Responses caching is enabled	~~MEDIUM~~ PARTLY REMEDIATED
M2. Unencrypted storage of sensitive data	~~MEDIUM~~ PARTLY REMEDIATED
M3. Missing certificate validation	~~MEDIUM~~ PARTLY REMEDIATED
M4. Weak password requirements	~~MEDIUM~~ REMEDIATED
M5. Enumeration of registered users	~~MEDIUM~~ PARTLY REMEDIATED
M6. Insecure versions of TLS protocol are supported	~~MEDIUM~~ PARTLY REMEDIATED
Low Risk Findings	
L1. The application server allows unencrypted communications	~~LOW~~ REMEDIATED
L2. Missing password changing functionality	LOW
L3. Lack of binary protection	LOW
L4. Usage of weak hash algorithms	LOW
L5. Verbose error messages	~~LOW~~ REMEDIATED
L6. Platform information is disclosed in server's responses	LOW
L7. Using components with known vulnerabilities	LOW
L8. Support of TLS ciphers suites without forward secrecy	LOW

L9. Missing debugger detection	LOW

High Risk Findings

Two **high**-severity issues were found in the application, as described below.

H1. Missing function-level access control

Risk Rating: ~~HIGH~~ REMEDIATED

CVSS: 8.1 (AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N)

Remediation Efforts: MEDIUM

Summary

During the assessment, DataArt identified that the application did not properly implement function-level access controls for a part of functionality. Some of exposed operations did not properly verify (or did not verify at all) that passed parameters belonged to objects of the current user as well as the requested functionality was allowed for execution within granted permissions. For example, as a possible way of exploiting this vulnerability, a malicious user without appropriate permissions could transfer money from other accounts, list transactions processed by other accounts, and view information related to other companies.

Affected Functionality

The issue affected the following functionalities:

1. Knowing the accounts` IDs, a malicious user without appropriate permissions can list transactions processed within any other accounts:
   a. **GET** https://api.[redacted]com/banking/[redacted]transactions?accountId=*accountID*
2. Knowing the business keys, a malicious user without appropriate permissions can view information related to other companies:
   a. **GET** https://api.[redacted]com/banking/[redacted]dashboard
      `businesskey: business key`
   b. **GET** https://api.[redacted]com/dashboard/banking
      `businesskey: business key`
   c. **GET** https://api.[redacted]com/banking/[redacted]wire
      `businesskey: business key`
   d. **GET** https://api.[redacted]com/banking/[redacted]ach
      `businesskey: business key`
   e. **GET** https://api.[redacted]com/banking/[redacted]check_deposit
      `businesskey: business key`
   f. **GET** https://api.[redacted]com/banking/[redacted]counterparty
      `businesskey: business key`
   g. **GET** https://api.[redacted]com/dashboard/banking/expenses
      `businesskey: business key`
   h. **GET** https://api.[redacted]com/banking/[redacted]counterparty
      `businesskey: business key`

3. Knowing a victim`s business key, a malicious user without appropriate permissions can create transfers from/to any account:
   a. **POST** https://api[redacted]com/banking/[redacted]ach_credit
   ```
   businesskey: business key
   uuid: 0
   {"amount": "", "accountId": "", "counterpartyName": "", "counterpartyId": "",
   "email": "", "direction": "", "counterpartyType": "", "sendEmail": false}
   ```
   b. **POST** https://api[redacted]com/banking/[redacted]ach_debit
   ```
   businesskey: business key
   uuid: 0
   {"amount": "", "accountId": "", "counterpartyName": "", "counterpartyId": "",
   "email": "", "direction": "", "counterpartyType": ""}
   ```
   c. **POST** https://api[redacted]com/banking/[redacted]wire
   ```
   businesskey: business key
   uuid: 0
   {"amount": "", "accountId": "", "counterpartyName": "", "counterpartyId": "",
   "email": "", "direction": "", "counterpartyType": "", "sendEmail": false}
   ```
   d. **POST** https://api[redacted]com/banking/[redacted]check_deposit
   ```
   businesskey: business key
   uuid: 0
   {"amount": "", "accountId": "", "osName": "", "frontImage": "", "osVersion": "",
   "backImage": "", "personId": ""}
   ```
4. Knowing the business keys, an ordinary user without appropriate permissions can perform different actions on behalf of other accounts.
   a. Update attributes:

   POST https://api[redacted]com/attributes
   ```
   {"parentDocument":"","data":{"LastSeen":},"collection":""}
   ```
   b. Process user`s registration:

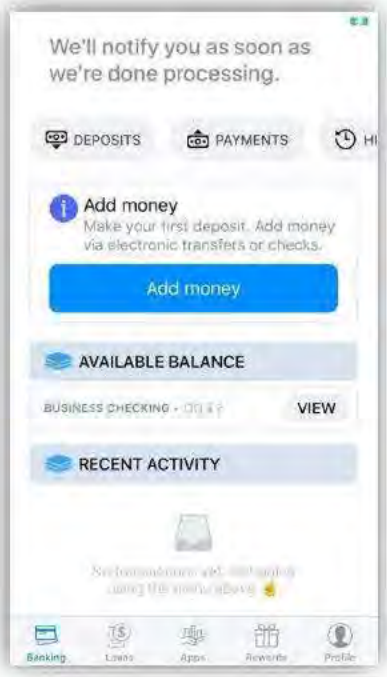
   POST https://api[redacted]com/banking/[redacted]application
   ```
   {"coOwnerInfo": [], "primaryApplicantInfo": "", "businessInfo": "{}"}
   ```
   c. Upload banking checks:

   POST https://api[redacted]com/banking/[redacted]file
   ```
   <Binary data>
   ```

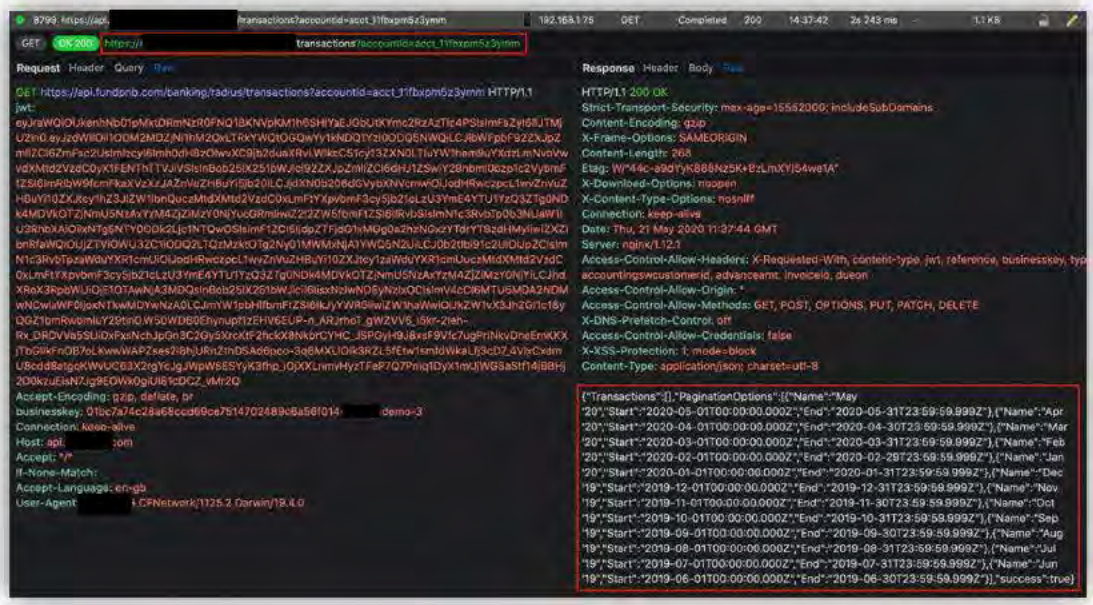
Proof of Concept

The workflow below provides the evidence how an ordinary user without appropriate permissions can create money transfers from other accounts:

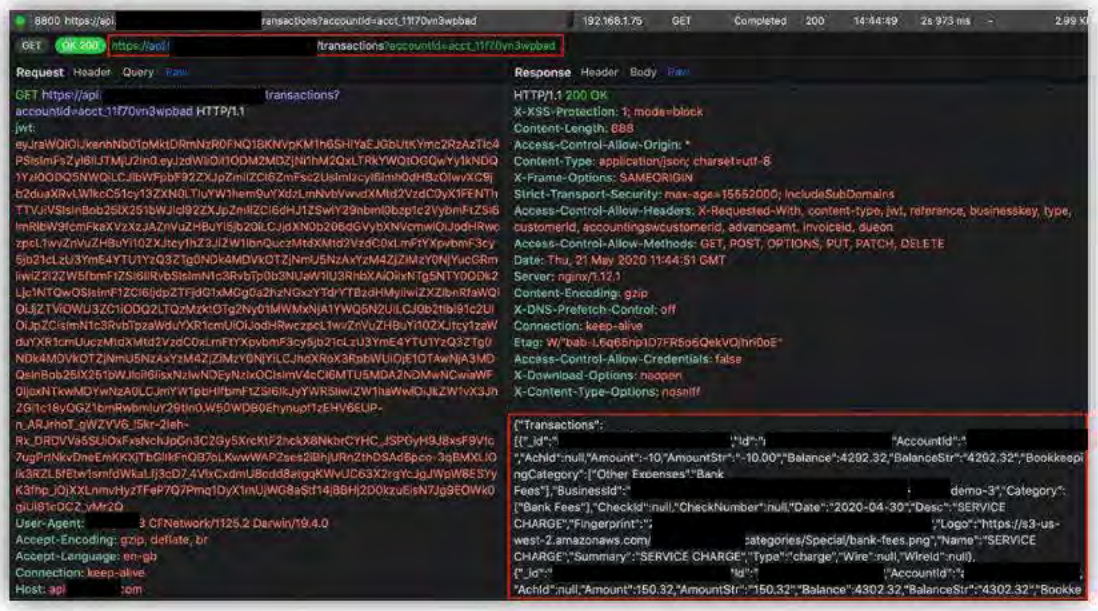
1. Login as demo_[redacted]2@[redacted]com which has *acct_11fbxpm5z3ymm* account ID.



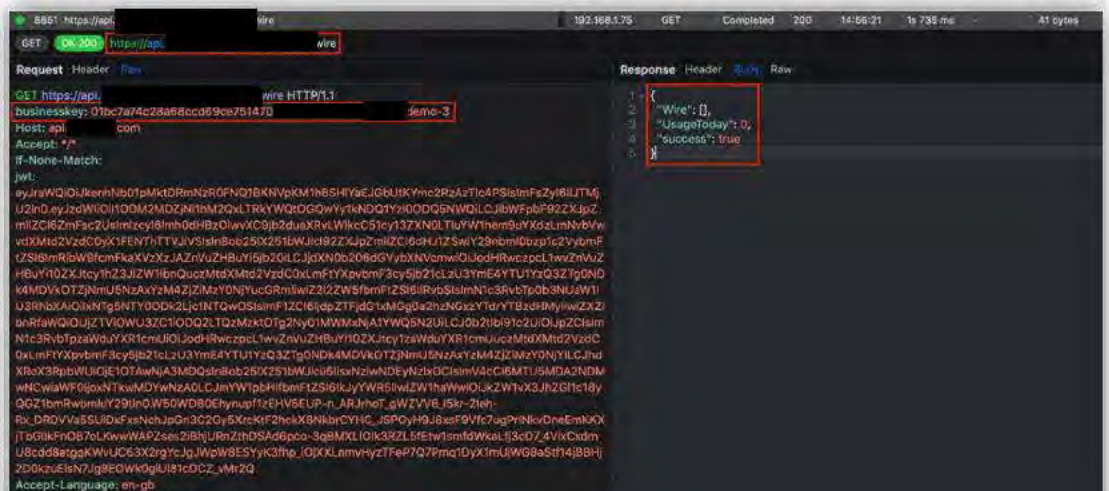
2. Send the request which lists all the transactions processed by this account.



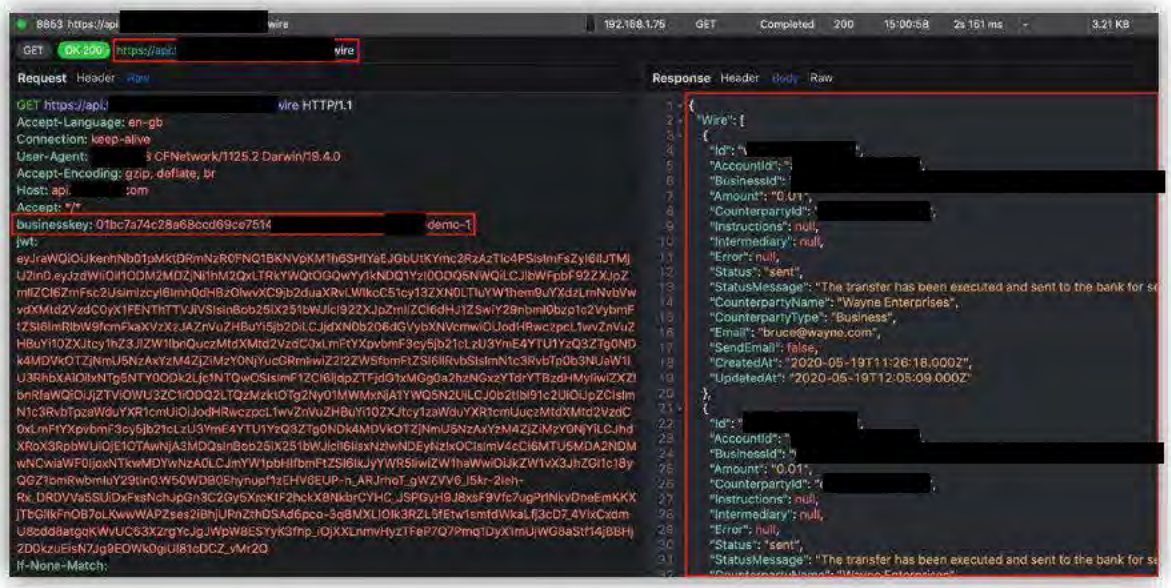
3. On behalf of this user send another request which lists all the transactions processed by another account demo@[redacted]com with *acct_11f70vn3wpbad* account ID by modifying the *accountId* parameter in URL.



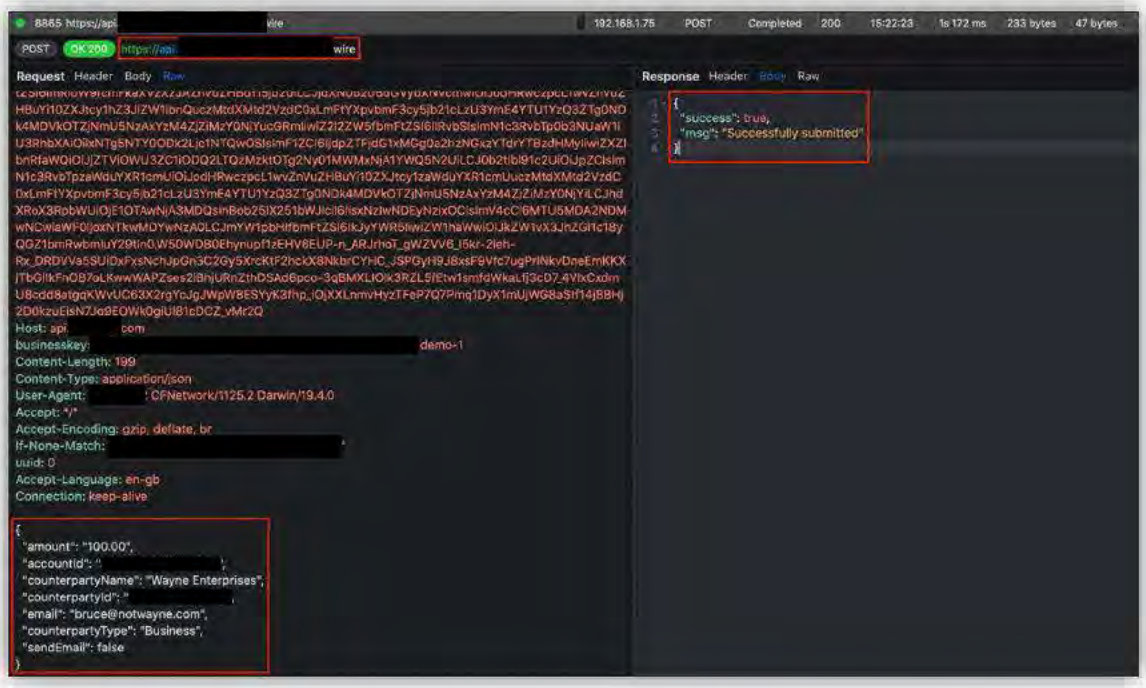
4. In order to list wire transactions of the victim account demo@[redacted]com, we need to somehow guess the business key.
5. List wire transactions of the current user demo_[redacted]2@[redacted]com:



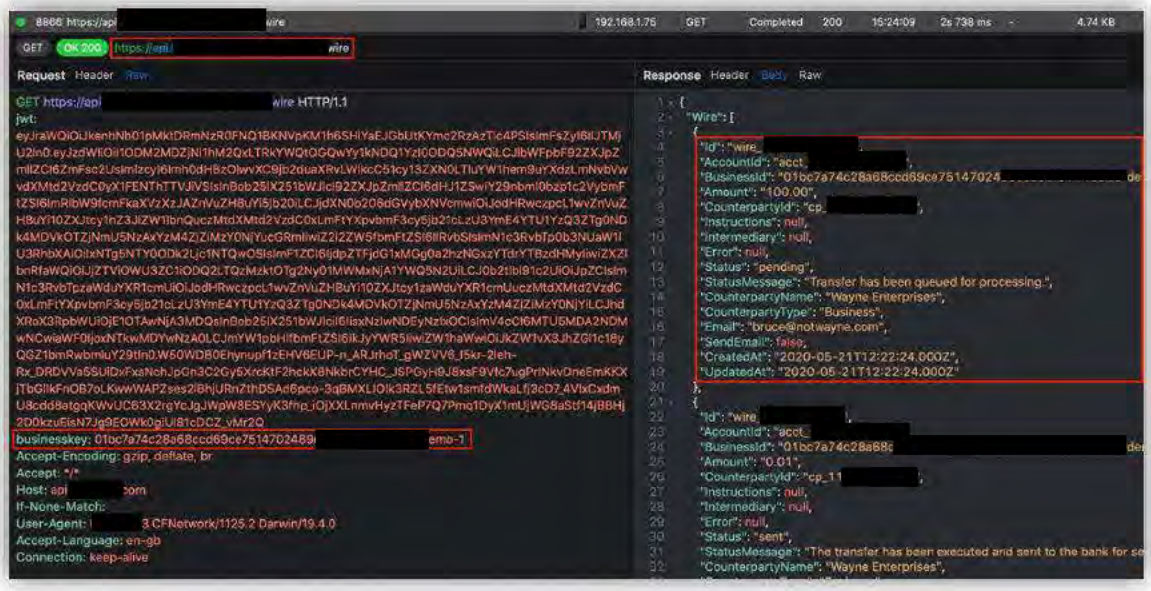
6. Modify the last part of the *businesskey* header from **3** to **1** and obtain wire transactions of the victim user demo@[redacted]com. The latest transaction amount was **0.01 USD** and was made to bruce@wayne.com.



7. On behalf of the same user initiate the wire transaction. Header *businesskey* has to be set as in the previous request. Set the *uuid* header to **0**. Set the transaction initiator ID to the victim`s *acct_11f70vn3wpbad* and the counterparty account to attackers one.



8. List wire transactions of the victim`s account. Identify the attacker`s transaction in the list.



Recommendations

With the goal of minimizing the possible security risks connected with improper handling of user permissions, DataArt recommends introducing proper access controls to all application`s resources and functionality. Effective access control mechanisms should follow the best practices listed below:

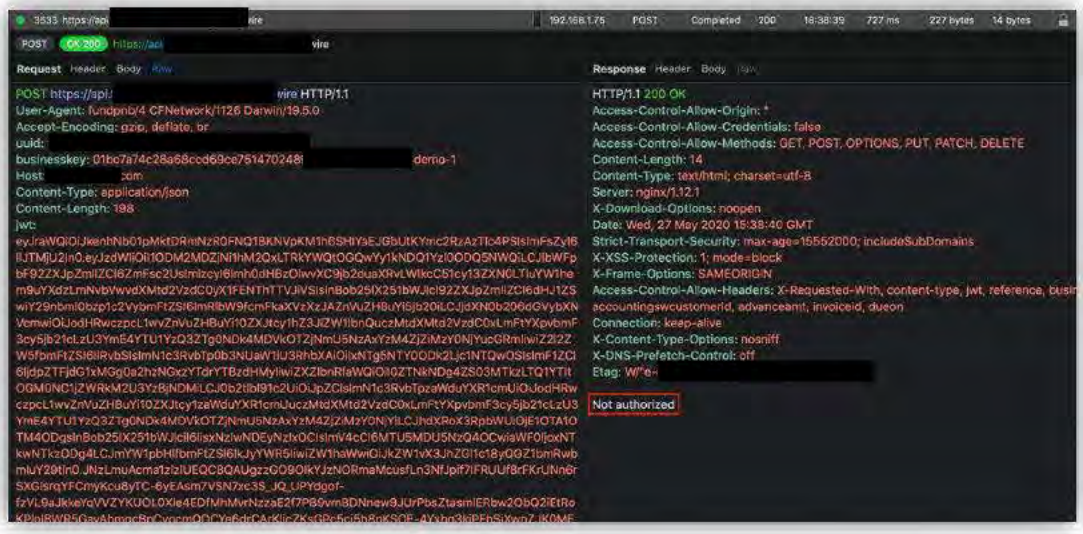
- Use a central application component to check access controls.
- Process every client request via this component to validate that the user making the request is permitted to access the functionality and resources being requested.
- Use programmatic techniques to ensure that there are no exceptions to the previous point: developers must be forced to explicitly embed access control logic into every page and resource.
- The access to all functionalities and resource available in the system should be denied by default.

The detailed information could be found in the articles below:

- https://owasp.org/www-project-proactive-controls/v3/en/c7-enforce-access-controls
- https://cheatsheetseries.owasp.org/cheatsheets/Access_Control_Cheat_Sheet.html
- https://www.packetlabs.net/broken-access-control/

Remediation

As of **May 27, 2020**, the issue was successfully **remediated**. DataArt confirms that currently the application properly validates access control in all cases mentioned above:



H2. Missing multi-factor authentication

Risk Rating: HIGH

CVSS: 9.1 (AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N)

Remediation Efforts: HIGH

Summary

During the testing authorization workflow, DataArt noticed that the application did not implement multi-factor authentication for functionality that operates with highly sensitive data (money transfer, login to the application).

Utilization of multi-factor verifications could protect against potential security risks connected with the cases when a user account is compromised via weak, re-used or stolen passwords. Despite any technical security controls implemented on the application, users are liable to choose weak passwords, or to use the same password on different applications. Application's developers should assume that users` passwords will be compromised as some point, so the system should be designed in order to defend user accounts from attacks using well-known, leaked and stolen credentials.

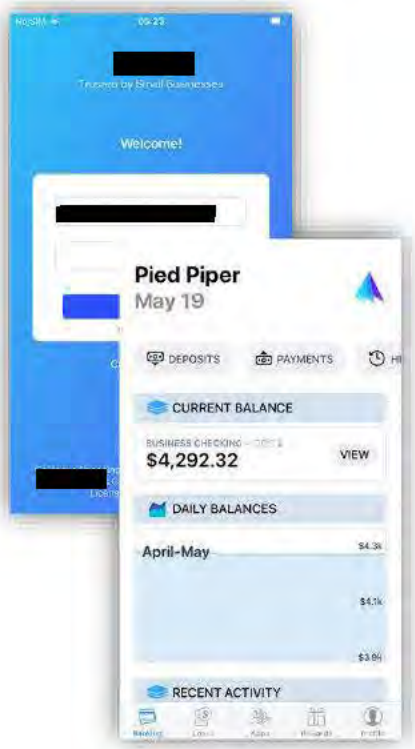
Multi-factor login mechanisms are designed to provide enhanced security over the simple model based on username and password. It is recognized as the best defense against the majority of password-related attacks, including brute-force, credential stuffing and password spraying.

Affected Functionality

The issue affected application`s functionality operating with sensitive data.

Proof of Concept

The following screenshots demonstrate the user authentication workflow.



As shown above, no multi-factor authentication was implemented in the application for user login.

Recommendations

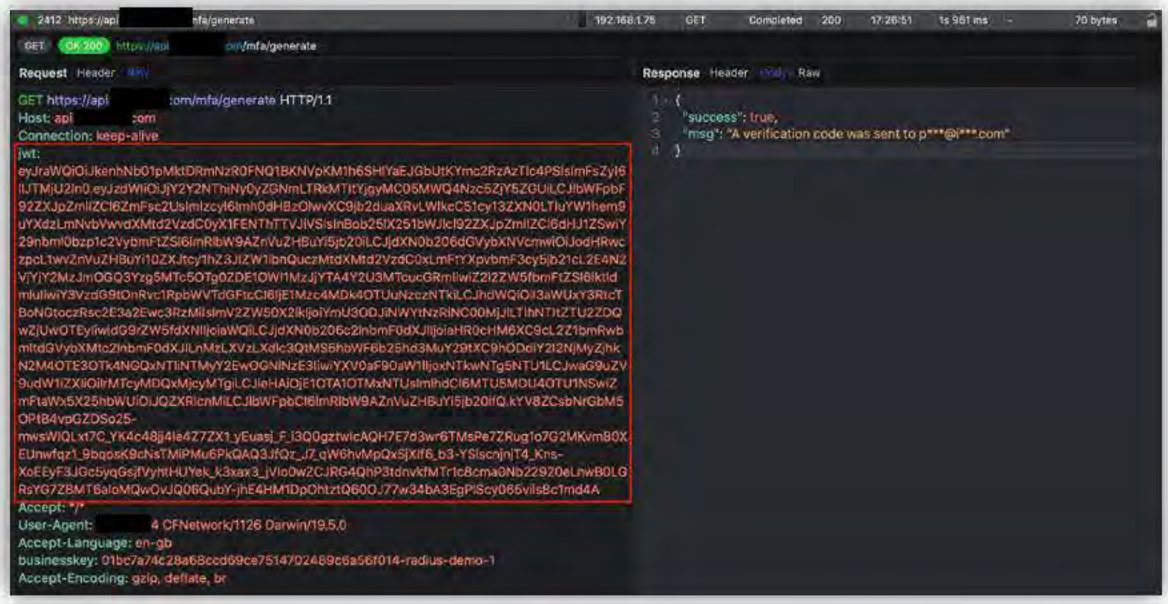
DataArt recommends using multi-factor authentication as an additional layer of security for user accounts. It has to combine something user knows: like email and password, with something user has: the code that is sent to user`s email, to user's mobile phone via a signed push notification or generated by the special code generating application (e.g. Google Authenticator). Other methods of delivering the one-time code can be used, but it should be noted, that SMS-based multi-factor authentication is considered deprecated.

Remediation

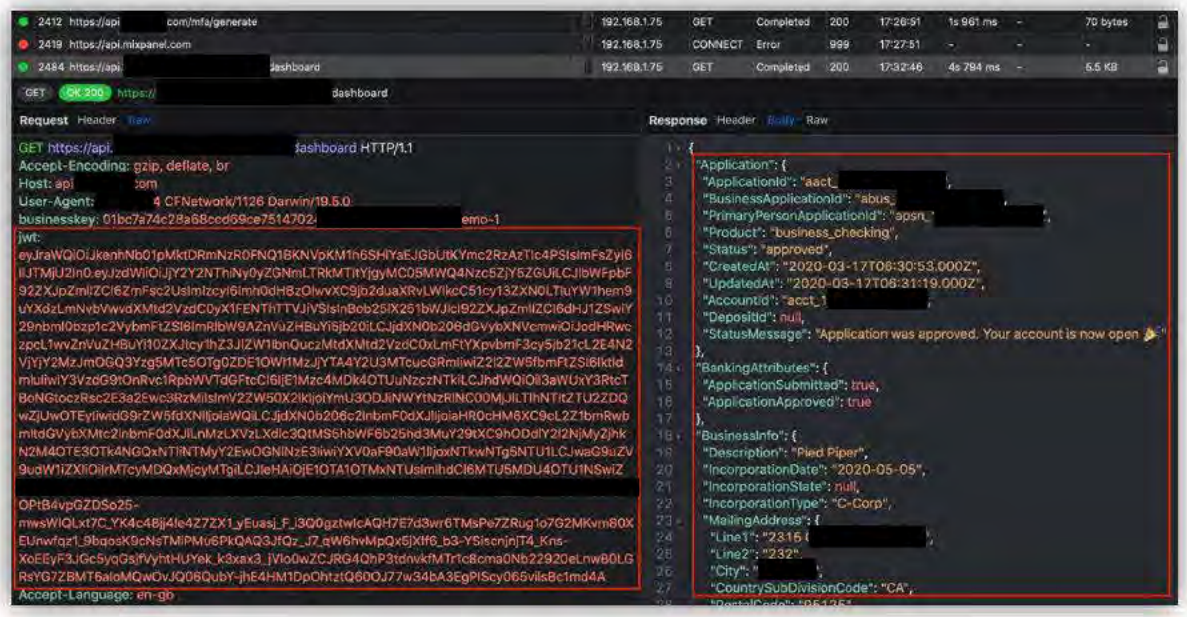
As of **May 27, 2020**, the issue was **still actual**. Despite the fact that the two-factor authentication has been implemented, it is possible to access a user`s account without the second factor.

The workflow below provides the evidence how the implemented multi-factor authentication can be bypassed by a malefactor:

1. An attacker who knows user`s credentials sends the authentication request using the login form and receives the access token.



2. Without the one-time code sent to the email, attacker can access user`s data and perform transactions from the account using this access token.



The two-factor verification during the transactions processing can be bypassed in the same way.

Additionally, it has to be noted that one-time codes were generated using the predictable Random Number Generating algorithm. Below is the screenshot which contains codes that were generated within the period of 20 minutes.

As can be seen, some codes are similar to each other. Under certain conditions easy-guessable codes may jeopardize the security of users' accounts.

Medium Risk Findings

Six **medium**-severity issues were found in the application, as described below.

M1. Responses caching is enabled

Risk Rating: MEDIUM (PARTLY REMEDIATED)

CVSS: 5.1 (AV:L/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N)

Remediation Efforts: LOW

Summary

During the testing, it appeared that the application stored sensitive data such as cookies, transactions, and users` data in the URL and cookies cache files. Furthermore, these files were not wiped after the user logs out of the application. Since these files have no encryption, the issue can cause sensitive data leakage or even account hijacking. The directory which contains URL cache files is only accessible for the application and it is forbidden to be backed up (via iTunes, Finder or iCloud). However, this protection can be easily bypassed using a jailbroken device. Also, it has to be noted that the folder containing cookies cache files was allowed to be backed up.

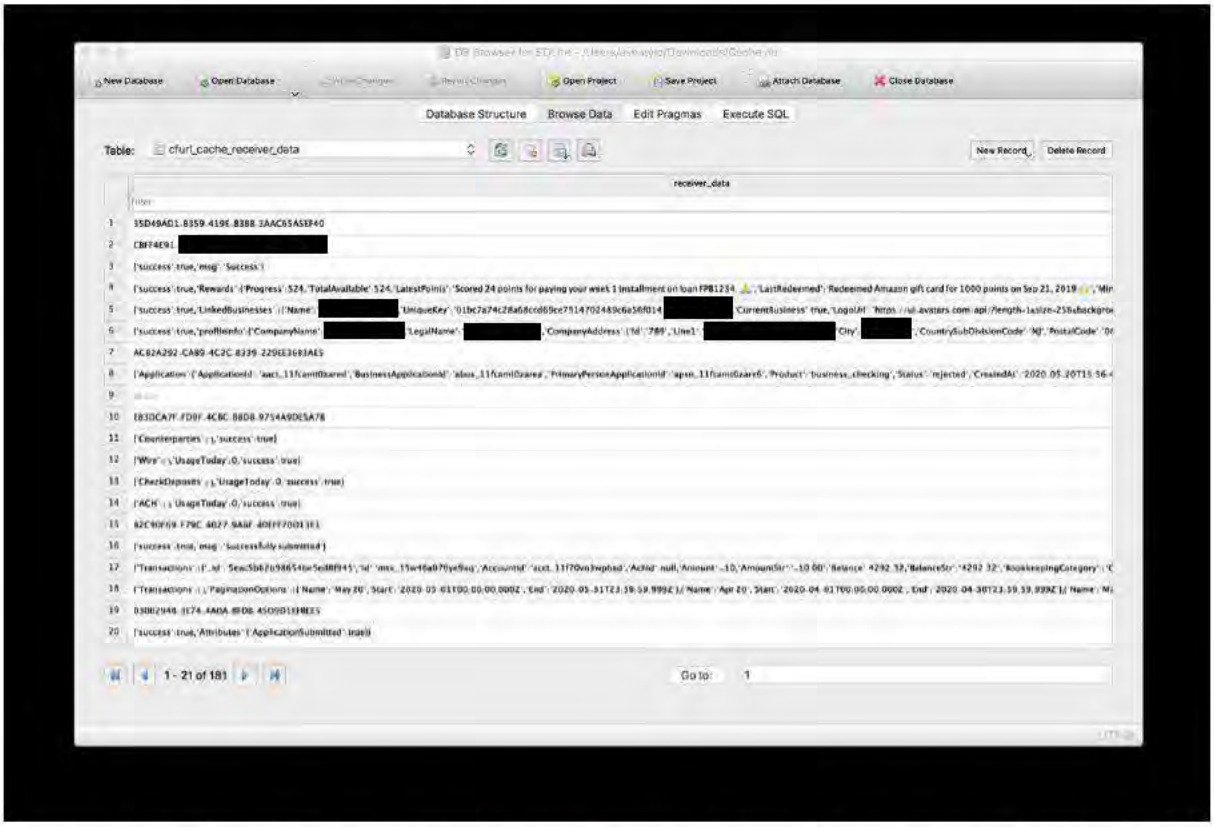
Affected Functionality

The issue affected the iOS application, URLCache and HTTPCookieStorage.

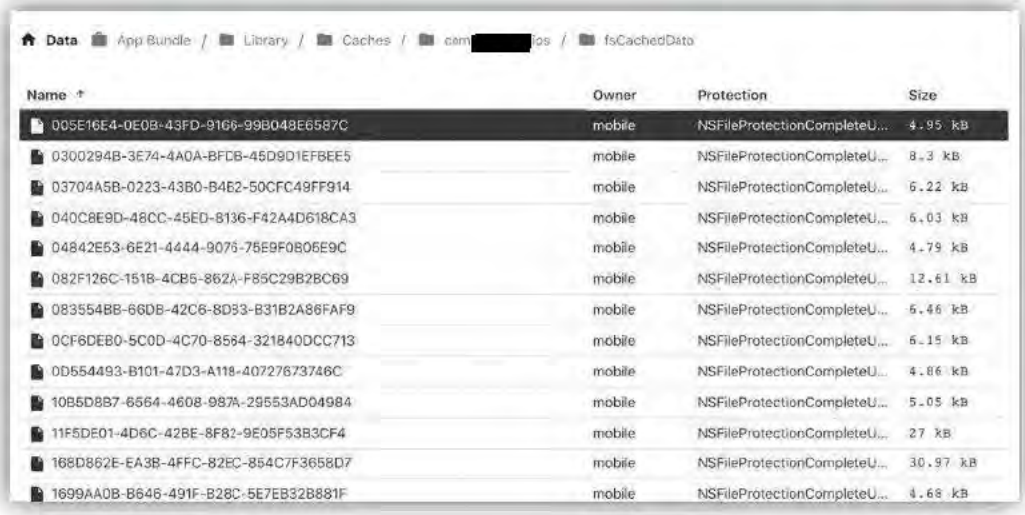
Proof of Concept

Cookies could be found cached in the *'Cookies.binarycookies'* file of the *Application/Library/Cookies* directory even after a user logs out. An attacker could read this file using any text viewing software as shown in the screenshot:

Responses with sensitive data could be found cached in the *'Cache.db'* file, obtained from the *Application/Library/Caches/com[redacted]ios* directory even after the logout. An attacker can read this file using any database viewing software as shown in the screenshot below:



Other pieces of application data, such as location and company information that was used during the registration process could be found in *Application/Library/Caches/com[redacted]ios/fsCachedData* directory:



Recommendations

DataArt would recommend eliminating caching of authentication-related requests by tuning HTTP *Cache-Control* directive on the server side. The following server response headers could be used to prevent caching in all browsers:

- Cache-Control: no-cache, no-store, must-revalidate (applicable for all modern browsers);
- Pragma: no-cache (required for outdated browsers compatibility);
- Expires: 0 (required for outdated browsers compatibility).

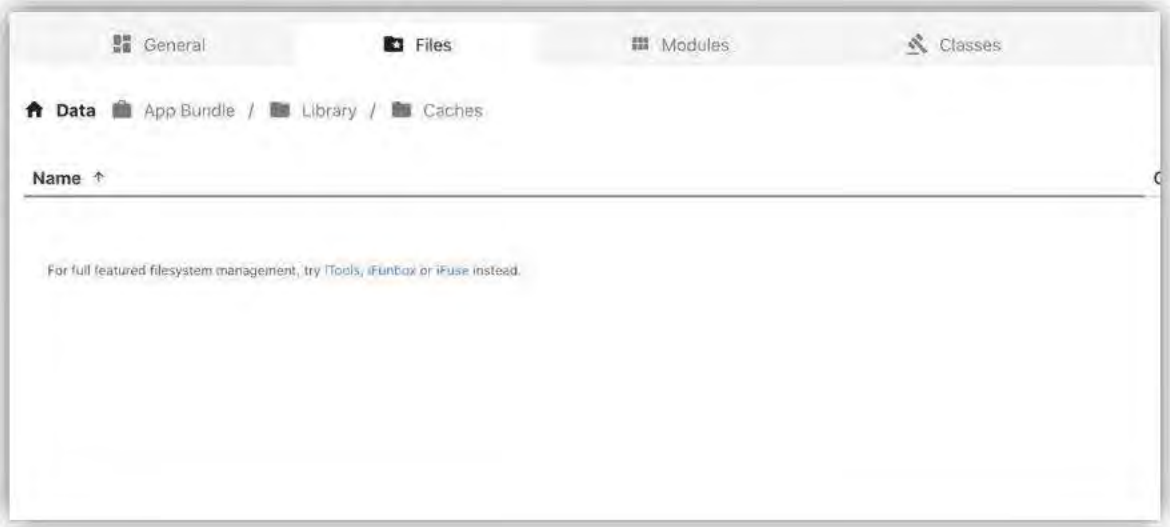
Please refer to the following link for more details:

- https://www.nginx.com/blog/nginx-caching-guide/
- https://stackoverflow.com/questions/1046966/whats-the-difference-between-cache-control-max-age-0-and-no-cache
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control
- https://stackoverflow.com/questions/49547/how-to-control-web-page-caching-across-all-browsers/2068407#2068407

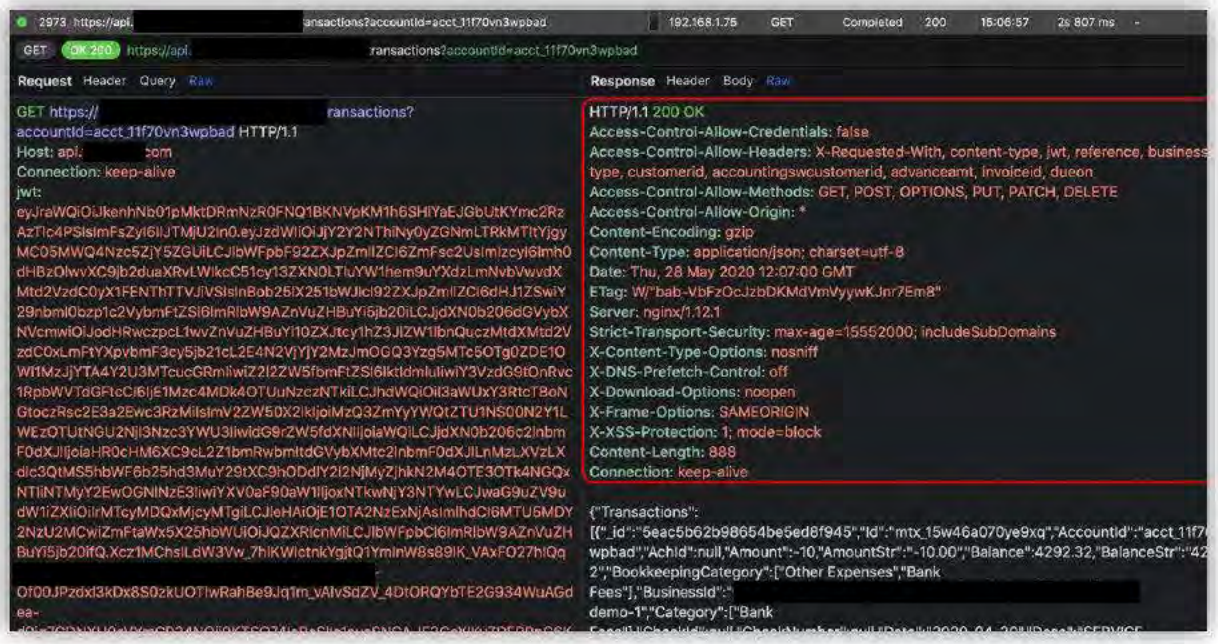
Alternatively, it is possible to configure cache policy while creating respective instance of *URLRequest*. Also, the application should erase all the data that was generated during the user`s session after the logout or its expiration.

Remediation

As of **May 27, 2020**, the issue was **partially remediated**. DataArt confirms that now the application does not store any sensitive data in the URL and cookies cache files:



However, DataArt also noticed that the server side still has not implemented any caching directives preventing intermediate proxies to store server responses:



M2. Unencrypted storage of sensitive data

Risk Rating: ~~MEDIUM~~ (PARTLY REMEDIATED)

CVSS: 5.1 (AV:L/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N)

Remediation Efforts: MEDIUM

Summary

DataArt identified that the tested application did not encrypt sensitive data such as *KeyChain* items, photos, and property list items while storing them on user`s device. Furthermore, photos which contain highly-sensitive user`s data were not wiped after the user logs out. Also, weak pre-defined protection classes were set for these files, which made it possible for malefactors to access them after booting the device. If the attacker gets physical access to the device (for instance in case of theft), it would be possible to extract valuable information using these issues.

Affected Functionality

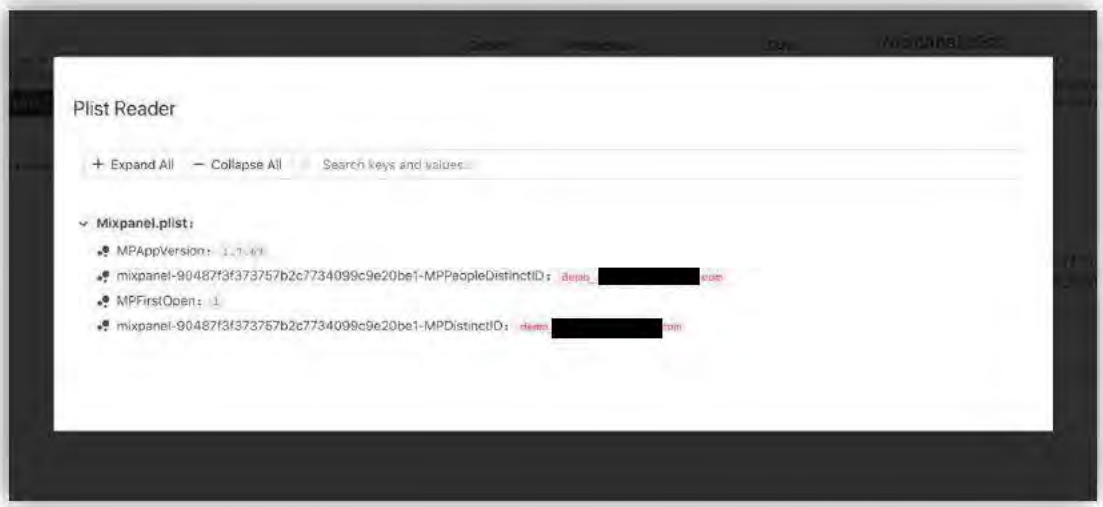
The following components were affected by the issue:

1. Fields which contained unencrypted user`s login were set as values to *MPPeopleDistinctID* and *MPDistinctID* keys in the *Application/Library/Preferences/Mixpanel.plist*
2. Images which contained a user`s driver license that was used during the registration process and banking checks that were used for deposits while performing transactions could be found in the */Library/Caches/com.apple.DocumentCamera/Scans/* directory
3. Items which were stored in the iOS *KeyChain*, such as authentication token and username

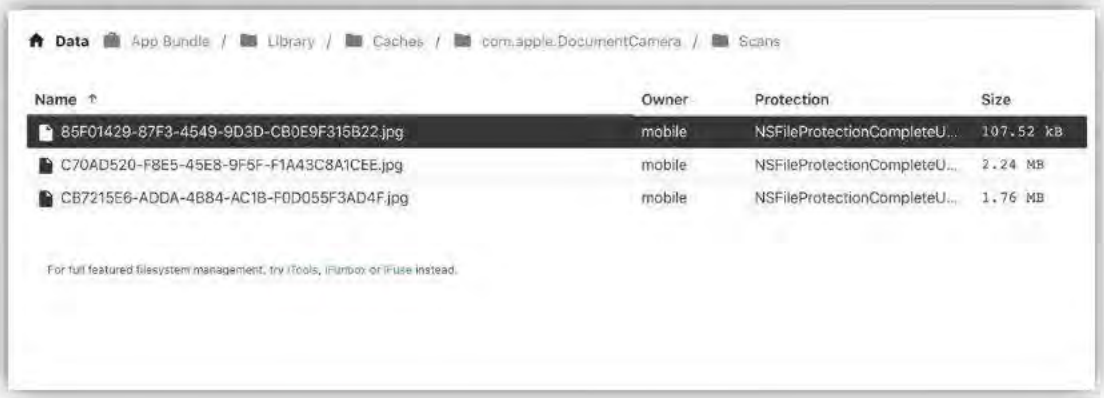
Both files had *NSFileProtectionCompleteUntilFirstUserAuthentication* protection flag set which is not enough for highly sensitive data

Proof of Concept

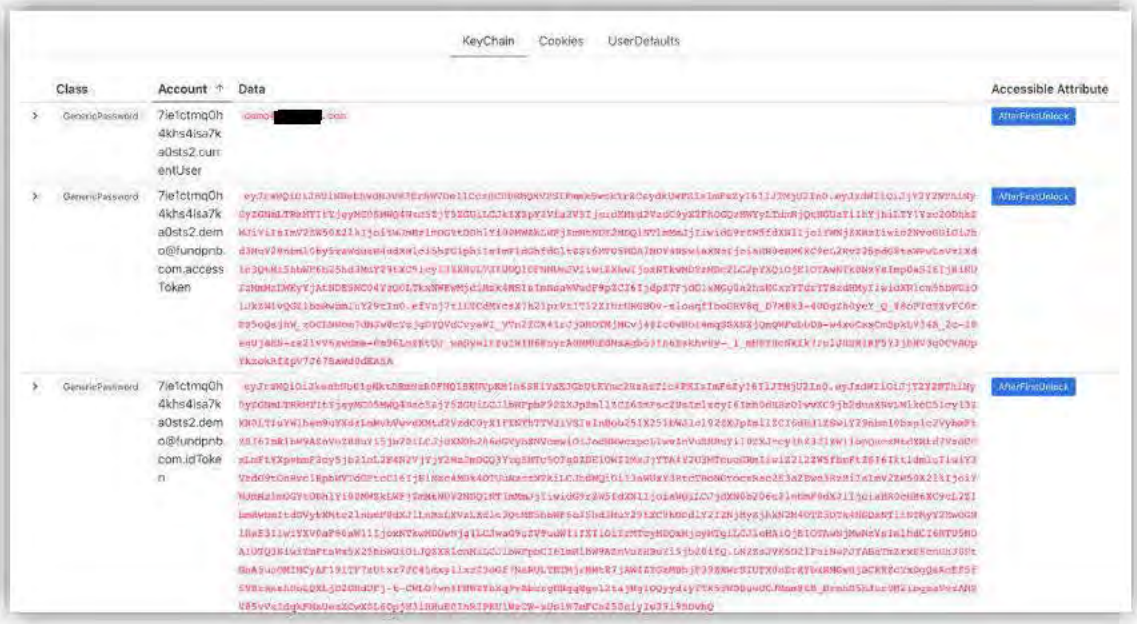
User`s email could be found in cleartext in the *Application/Library/Preferences/Mixpanel.plist* even after user`s logout:



The driver license and banking checks could be found in the */Library/Caches/com.apple.DocumentCamera/Scans/* directory as shown below:



The items which were stored in the *KeyChain* could be obtained in cleartext:



Recommendations

It's recommended to avoid storing sensitive data on a user`s device until it is absolutely necessary for the proper functioning of the application. Otherwise, such data should be encrypted with help of strong key creation and derivation mechanisms.

For example, it's possible to use two or more different sources of data to derive data encryption keys (DEKs).

- The encryption key should be user-specific and device-specific at the same time, so one piece of key material could be derived from a user's unique identifier (like login) while another may use device's characteristics;
- At least one piece of key material should not be kept on the device, so only those users who have successfully authenticated online will be able to access their data cached on the device. Otherwise, it should be created and stored in the device's secure cryptographic module (Secure Enclave / Secure Element), if possible.

A good example of such key derivation scheme is *KDF(H(device_id):H(user_id), H(secret))*, where *H* is a hash function (*SHA1+*), *KDF* is a key derivation function (*PBKDF2*, *bcrypt* or *Argon*) and secret is an independent piece of key material which is associated with the user's account but not stored on the device.

It's also recommended to erase all the data which was generated by the application user after logout or session expiration and set strong pre-defined protection classes (*Complete* or *CompleteUnlessOpen*) for protecting the stored data from being stolen when a malefactor attempts to copy files from a device which is locked or powered off.

Remediation

As of **May 27, 2020**, the issue was **partially remediated**. It was proved that now the application does not store images and sensitive data in property list files. However, the application still uses *KeyChain* for storage of the unencrypted login and the authentication token. In case a malefactor gains the physical access over victim`s device, he will be able to easily compromise this data:

M3. Missing certificate validation

Risk Rating: ~~MEDIUM~~ (PARTLY REMEDIATED)

CVSS: 5.9 (AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N)

Remediation Efforts: MEDIUM

Summary

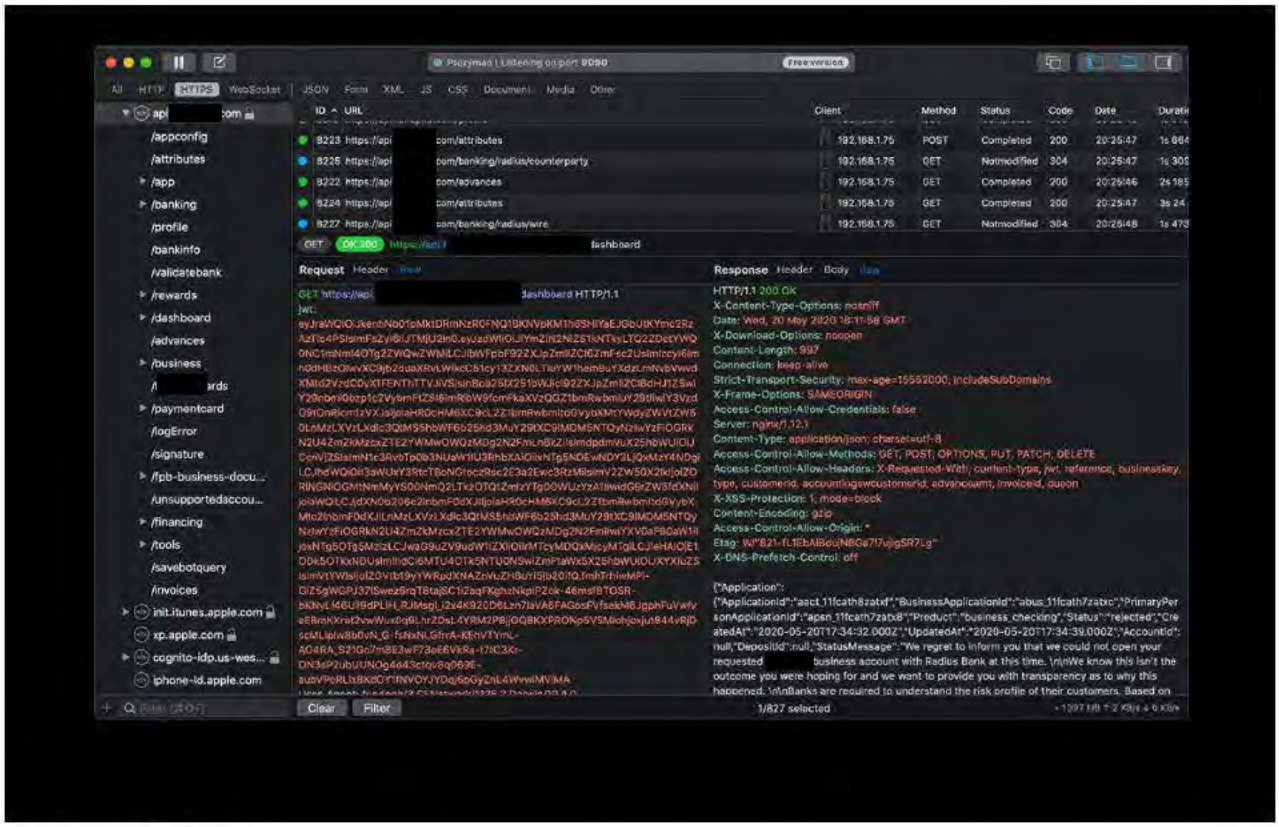
It was discovered that the application did not implement certificate validation while establishing secure network connections. An attacker could acquire a valid certificate through a compromised Certificate Authority or by installing a trusted user certificate on the device. Once the certificate is trusted, an attacker can perform Man-In-The-Middle attacks on encrypted communications and capture security-critical data.

Affected Functionality

The issue affected the iOS application, network module and certificate validation.

Proof of Concept

The following screenshot demonstrates the intercepted workflow of the authentication:



Recommendations

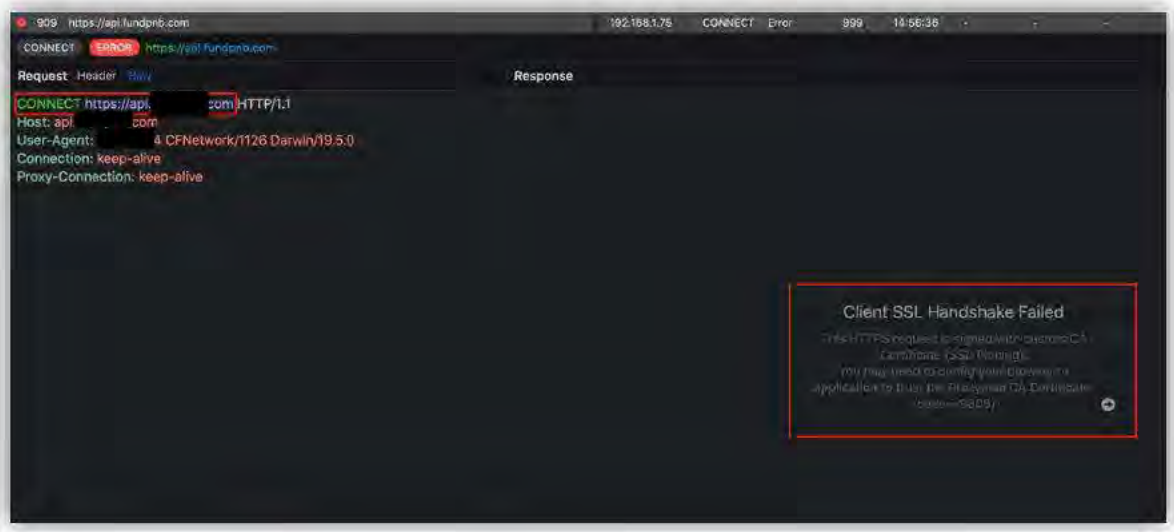
The application should implement certificate or public key pinning mechanisms or perform certificate revocation checks (OCSP Stapling) in order to implement additional layer of protection and exclude any possibility of interception attacks, which can lead to disclosure of sensitive information.

Regardless of the choice, it's necessary to find an optimal balance (for both technical teams and end users) between security and usability. The options are listed below:

- Test OCSP Stapling in a situation when a device has a self-signed root certificate installed and is connecting to a server via a MitM proxy. If SSL connection is dropped by the system, OCSP Stapling works as a complete substitution for classic pinning.
- If OCSP Stapling fails to prevent a connection which is listened by a malicious proxy, then it could not be a proper substitution for pinning. In such case, it's recommended to implement both pinning (as the primary method of verification) and OCSP Stapling as a fallback (when a pin expires and should be rotated). This option gives some extra time to prepare and release an update with new pins without a hurry. Another option is to implement Certificate Transparency only and accept the risk of a potential MitM attack.
- It's also possible to go on with pinning and implement a mechanism to push a new pin onto the device when a new certificate is released and an old one is about to expire. Such pin could be saved in a secure storage on a device, but it could be either extracted from a rooted device or intercepted and altered by a malefactor while being pushed.

Remediation

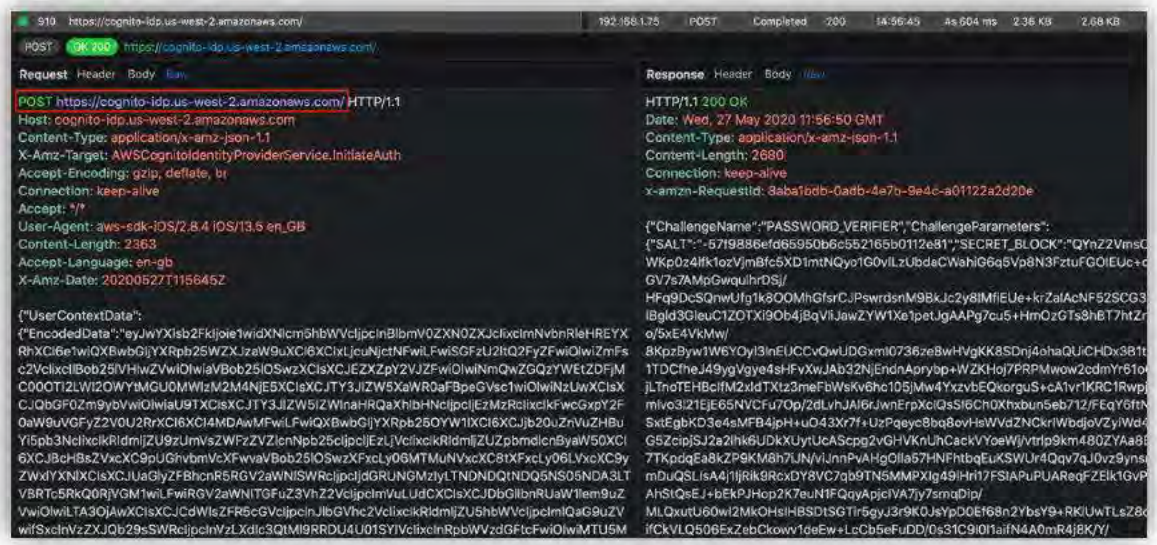
As of **May 27, 2020**, the issue was **partially remediated**. DataArt confirms that currently the certificate validation is implemented for all connections between the mobile application and api [redacted] com server:



However, the issue is still actual for interaction between the application and SSO endpoint:

- cognito-idp.us-west-2.amazonaws.com

The screenshot below shows the intercepted authentication process:



M4. Weak password requirements

Risk Rating: ~~MEDIUM~~ REMEDIATED

CVSS: 5.9 (AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N)

Remediation Efforts: LOW

Summary

DataArt observed that the strong password policy was not employed by the application, which makes it easier for attackers to compromise users` accounts. The target application recommended users to use passwords with 6 characters long. However, the recommended minimal password length is 12 characters long. Furthermore, the application did not recommend using uppercase and lowercase characters, numbers, or special characters in the password, and did not check whether the password is the same as email or widely used one.

An authentication mechanism is only as strong as its credentials. For this reason, it is important to require users to have strong passwords. Lack of password complexity significantly reduces the search space when trying to guess user`s passwords, making brute-force attacks easier. Also, if the passwords` hashes were compromised, it would be easier for an attacker to crack them.

Affected Functionality

The issue affected the iOS application and the authentication flow.

Proof of Concept

The screenshot below demonstrates that password with 6 symbols length was recommended by the application.

Recommendations

DataArt recommends enforcing the following minimum set of password quality requirements:

- Password minimum length: at least twelve (12) characters long.
- Password complexity: at least 3 combination of the following – digits, lowercase and uppercase letters, digits and/or special characters.
- Password matching: at least not equal or contain username; should optionally be checked against common dictionary words and names.

As an additional countermeasure, the application could display graphical control, which reflects complexity of the currently typed password. It will motivate users to create strong passwords.

Remediation

As of **May 27, 2020**, the issue was **remediated**. DataArt confirms that currently the application properly validates complexity of user passwords:

M5. Enumeration of registered users

Risk Rating: ~~MEDIUM~~ (PARTLY REMEDIATED)

CVSS: 5.3 (AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N)

Remediation Efforts: MEDIUM

Summary

DataArt identified that the application notified a non-authorized actor about existence of the user with requested email. Such behavior provides an ability to a potential malefactor to enumerate all users registered in the system.

Collected information could be used for further attacks like brute-force or social engineering (for instance for spreading phishing emails or other social engineering attacks).

Affected Functionality

The issue affected the following endpoints:

1. **Login:**

   POST https://cognito-idp.us-west-2.amazonaws.com/

   {"UserContextData":{"EncodedData":""},"ClientMetadata":{"cognito:deviceName":"","cognito:bundleShortV":"","cognito:idForVendor":"","cognito:bundleVersion":"","cognito:bundleId":"com.[redacted]ios","cognito:model":"","cognito:systemName":"","cognito:iOSVersion":""},"AuthParameters":{"SRP_A":"","SECRET_HASH":"","USERNAME":""},"AuthFlow":"USER_SRP_AUTH","ClientId":""}

   a. The response in case the requested user exists:

   {"__type":"NotAuthorizedException","message":"Incorrect username or password."}

   b. The response in case the requested user does not exists:

   {"__type":"UserNotFoundException","message":"User does not exist."}

2. **Registration:**

   POST https://cognito-idp.us-west-2.amazonaws.com/

   {"SecretHash":"","Username":"","ClientId":"","ValidationData":[{"Value":"","Name":"cognito:deviceName"},{"Value":"","Name":"cognito:bundleShortV"},{"Value":"","Name":"cognito:idForVendor"},{"Value":"","Name":"cognito:bundleVersion"},{"Value":"com.[redacted]ios","Name":"cognito:bundleId"},{"Value":"","Name":"cognito:model"},{"Value":"","Name":"cognito:systemName"},{"Value":"","Name":"cognito:iOSVersion"}],"Password":"","UserAttributes":[{"Value":"","Name":"phone_number"},{"Value":"","Name":"given_name"},{"Value":"","Name":"family_name"},{"Value":"","Name":"email"},{"Value":"","Name":"custom:tosTimeStamp"},{"Value":"","Name":"custom:signature"},{"Value":"","Name":"custom:termsUrl"}],"UserContextData":{"EncodedData":""}}

   a. The response in case the requested user exists:

   {"__type":"UsernameExistsException","message":"User already exists"}

   b. The response in case the requested user does not exists:

   {"CodeDeliveryDetails":{"AttributeName":"phone_number","DeliveryMedium":"SMS","Destination":""},"UserConfirmed":false,"UserSub":""}

3. **Password recovery:**

   POST https://cognito-idp.us-west-2.amazonaws.com/

   {"UserContextData":{"EncodedData":""},"SecretHash":"","Username":"","ClientId":""}

   a. The response in case the requested user exists:

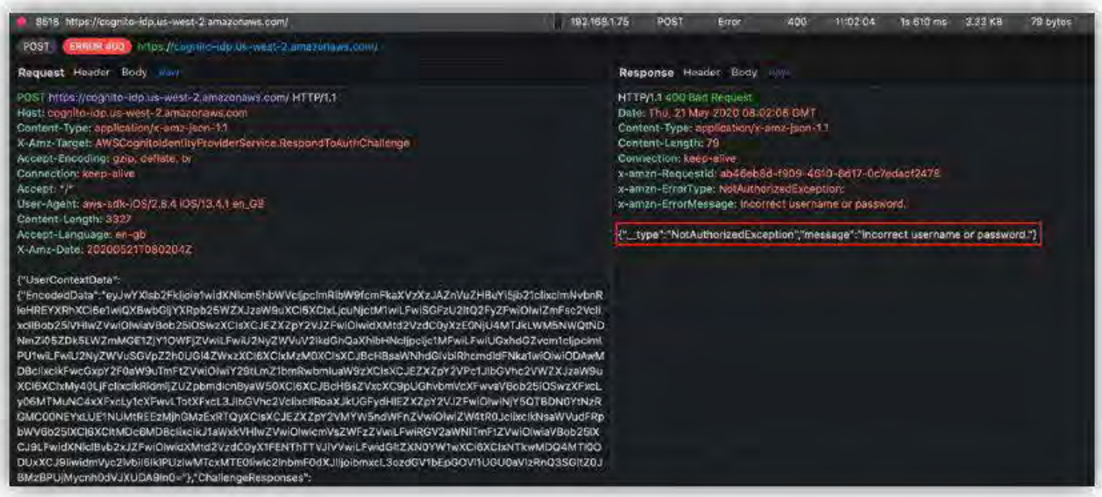
   {"CodeDeliveryDetails":{"AttributeName":"phone_number","DeliveryMedium":"SMS","Destination":""}}

   b. The response in case the requested user does not exists:

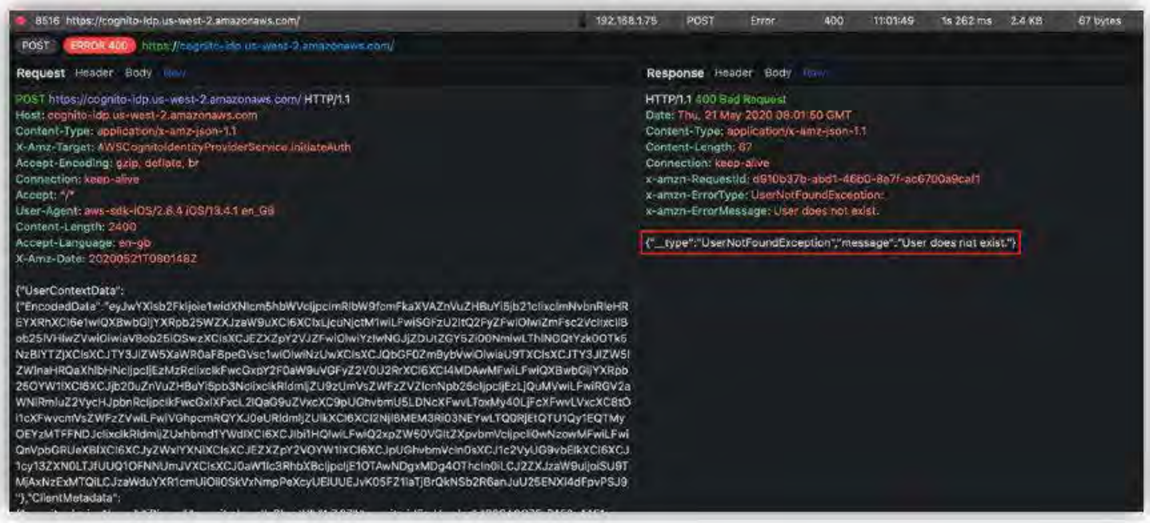
   {"__type":"UserNotFoundException","message":"Username/client id combination not found."}

Proof of Concept

The following screenshot shows the server response in case the requested user existed in the system:



Another screenshot shows the server response in case the requested user did not exist in the system:



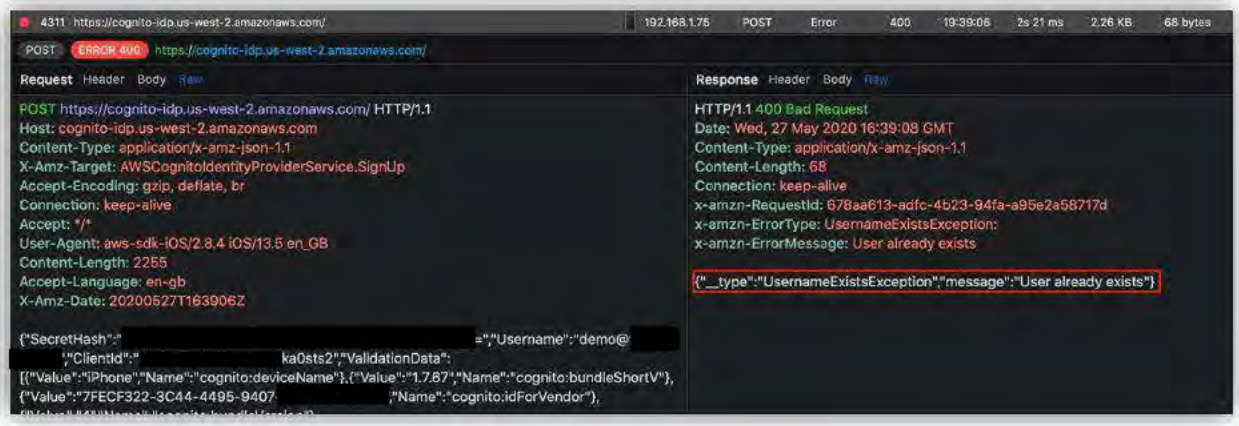
Recommendations

In order to avoid automated enumeration of valid e-mails, DataArt suggests using the following techniques:

1. Do not show an error message providing information about existence or non-existence of the entered email. In both cases, the application should return a generic message that the provided information is incorrect.
2. Add a CAPTCHA challenge after the series of failed attempts to prevent automated email enumeration.
3. For the forgot password functionality, the application should return a generic message that password recovery information has been sent to the specified email address.
4. Configure the server so that it will respond with a random amount of time for any request, throwing off the noticeable difference.

Remediation

As of **May 27, 2020**, the issue was **partially remediated**. DataArt confirms that the application does not disclose sensitive information for login and password recovery processes. However, it is still possible to enumerate user existence through the registration functionality:



M6. Insecure versions of TLS protocol are supported

Risk Rating: ~~MEDIUM~~ (PARTLY REMEDIATED)

CVSS: 4.8 (AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:N)

Remediation Efforts: LOW

Summary

During the testing, DataArt determined that the application server supported TLS v1.1 and TLS v.1.0. These versions of security protocol are affected by multiple cryptographic flaws. An attacker can exploit them to conduct man-in-the-middle attacks or to decrypt communications between the server and clients.

Affected Functionality

The following server-side components supported weak TLS versions:

1. TLS 1.0:
   a. cognito-idp.us-west-2.amazonaws.com
2. TLS 1.1:
   a. cognito-idp.us-west-2.amazonaws.com
   b. [redacted]com

Proof of Concept

Command line utility *testssl* was used to test for deprecated cryptographic protocols support. Below is a fragment of its output with highlighted successful connections using TLS v1.0 and TLS v1.1.

```
Testing protocols via sockets except NPN+ALPN

SSLv2      not offered (OK)
SSLv3      not offered (OK)
TLS 1      offered (deprecated)
TLS 1.1    offered (deprecated)
TLS 1.2    offered (OK)
TLS 1.3    not offered and downgraded to a weaker protocol
NPN/SPDY   h2, http/1.1 (advertised)
ALPN/HTTP2 h2, http/1.1 (offered)
```

Recommendations

Due to possible CBC Chaining and Padding Oracle attacks, it is recommended to disable utilization of this obsolete version of the TLS protocol (TLS v1.0) and allow utilization only the latest ones such as TLS v1.2 and v1.3. PCI standards recommend using TLS 1.2 as a default.

Please note that the two main mobile platforms (Android & iOS), have been supporting TLS v1.2 from Android 5 Level 20 (released Oct 2014, Build 71) and iOS 9 (June 2015).

It should be noted that TLS v1.1 version is currently considered as secure however under certain circumstances it could also be also compromised. Moreover, please note that after March 2020 all modern browsers will stop supporting TLS v1.1, see the referenced link below for more details:

- https://blog.qualys.com/ssllabs/2018/11/19/grade-change-for-tls-1-0-and-tls-1-1-protocols

The detailed information about the issue can be found in the supporting links below:

- https://www.keycdn.com/blog/deprecating-tls-1-0-and-1-1
- https://www.ssllabs.com/ssltest/analyze.html?d=[redacted]com&s=13.35.126.114&hideResults=on
  Due to the fact that the affected applications utilize Amazon CloudFront, the issue could be fixed by applying an up-to-date TLS security policy for the corresponding AWS resources. Detailed information regarding this can be found in the links below:
- https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/distribution-web-values-specify.html#DownloadDistValues-security-policy
- https://www.cloudconformity.com/knowledge-base/aws/CloudFront/security-policy.html

Remediation

As of May 27, 2020, the issue was **partially remediated**. DataArt confirms that currently the main API server [redacted]com does not support outdated versions of TLS:

```
Testing SSL server [redacted].com on port 443 using SNI name [redacted].com

  SSL/TLS Protocols:
SSLv2     disabled
SSLv3     disabled
TLSv1.0   disabled
TLSv1.1   disabled
TLSv1.2   enabled
TLSv1.3   disabled
```

However, the issue is still actual for AWS-based cognito-idp.us-west-2.amazonaws.com:

```
Testing SSL server cognito-idp.us-west-2.amazonaws.com

  SSL/TLS Protocols:
SSLv2     disabled
SSLv3     disabled
TLSv1.0   enabled
TLSv1.1   enabled
TLSv1.2   enabled
TLSv1.3   disabled
```

Note: it has to be noted that [redacted] development team has no direct control of the configuration of AWS Cognito servers and remediation of the remaining part of the issue is dependent on AWS providing a resolution.

Low Risk Findings

Nine low-severity issues were found in the application, as described below.

L1. The application server allows unencrypted communications

Risk Rating: ~~LOW~~ REMEDIATED

CVSS: 3.7 (AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N)

Remediation Efforts: LOW

Summary

DataArt noticed that the target server did not implement server-side redirection to the encrypted communication channel. As the header redirection can be possibly bypassed, it could potentially allow transmitting the security-critical data in cleartext over the communication channel. An attacker could use this situation to compromise or eavesdrop on the HTTP communication between client and server using a Man-in-the-Middle attack to get an access to sensitive data like authentication token.

As the application properly implemented the *Strict-Transport-Security* header, the risk of this vulnerability was decreased to LOW level.

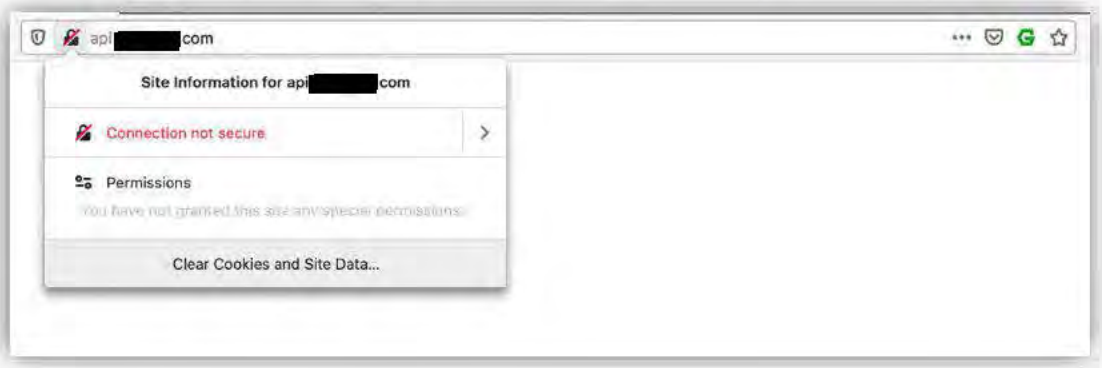
Affected Functionality

The issue affected the following server-side component:

- http://api[redacted]com

Proof of Concept

The screenshot below demonstrates that the affected host was successfully accessed via unencrypted channel:



Recommendations

The application should never use unencrypted connection for transfer of sensitive data. Well configured encryption will mitigate security risks connecting with intercepting data in transit during man-in-the-middle attacks.

Remediation

As of **May 27, 2020**, the issue was **successfully remediated**. It was proved that now the API server is not accessible in an unencrypted way.

L2. Missing password changing functionality

Risk Rating: LOW

CVSS: 3.7 (AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N)

Remediation Efforts: MEDIUM

Summary

DataArt identified that the application did not provide a clear way (except “forgot password” functionality) for users to change their passwords. However, such functionality is necessary for a well-designed authentication mechanism for at least two reasons:

- Periodic password change mitigates the threat of password compromise by reducing the window in which a given password can be targeted in a guessing attack.
- Users who suspect that their passwords may have been compromised need to be able to quickly change their password to reduce the threat of unauthorized use.

Affected Functionality

The issue affected all user accounts.

Recommendations

A separate password change function should always be implemented, to allow periodic password expiration or to allow users to change passwords if they want to for any reason. An easily understandable approach for password changing could save time and help to mitigate potential risks connected with compromising of a user`s password.

Remediation

As of **May 27, 2020**, the issue was **still actual**. Applications` users still do not have the ability to change their passwords in a clear way.

L3. Lack of binary protection

Risk Rating: LOW

CVSS: 3.7 (AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N)

Remediation Efforts: MEDIUM

Summary

DataArt found out that the binary obfuscation was not implemented in the application.

Obfuscation is the process of transforming code and data to make it more difficult to comprehend. It is an integral part of every software protection scheme. Programs can be made incomprehensible, in whole or in part, in many ways and to different degrees. For applications which require higher level of security, this reverse engineering countermeasures must be implemented. Without it, it would be possible for an attacker to harvest sensitive

information such as hard coded secrets, logical flaws, information about developers, etc. Information, collected during this phase can be used for further attacks.

Affected Functionality

The issue affected the iOS application.

Proof of Concept

For example, it's possible to obtain strings from the IPA file using *strings* utility and extract developer`s full name. This information could be used for attacks targeted on the developer:

Recommendations

DataArt recommends implementing the binary obfuscation technique which allows to generate irreversible, encrypted names for all the project`s objects.

In order to simplify the implementation, the following open-source tools could be considered:

- https://github.com/rockbruno/swiftshield
- https://www.polidea.com/blog/open-source-code-obfuscation-tool-for-protecting-ios-apps/

Remediation

As of **May 27, 2020**, the issue was **still actual**. DataArt noticed the binary obfuscation still has not been implemented in the application:

L4. Usage of weak hash algorithms

Risk Rating: LOW

CVSS: 3.7 (AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N)

Remediation Efforts: MEDIUM

Summary

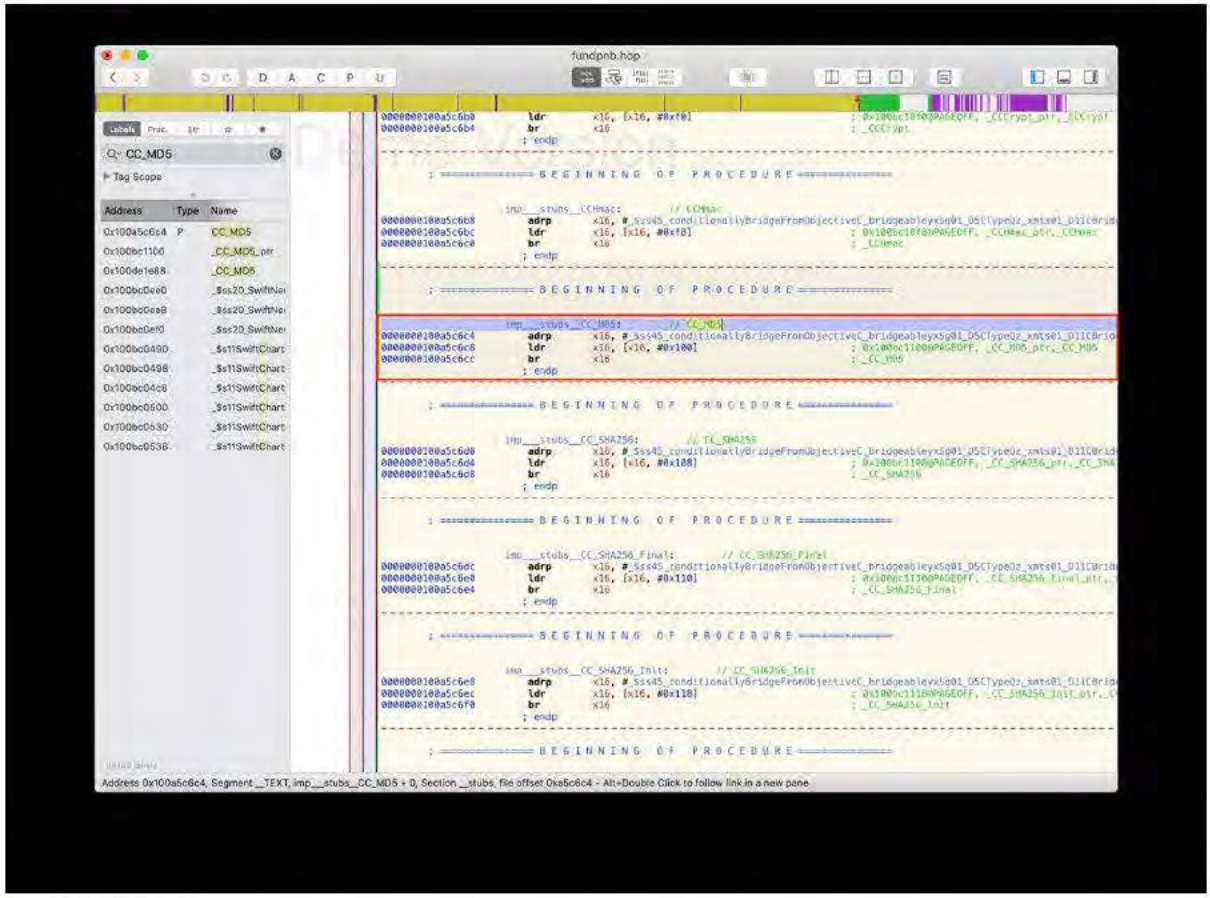
DataArt identified that the application used weak hashing algorithm, namely *MD5*. Weak hashing algorithms (e.g. *MD2*, *MD4*, *MD5* or *SHA-1*) can be vulnerable to different type of collisions attacks and other security weaknesses and should not be used when reliable hashing of data is required.

Affected Functionality

The issue affected the iOS application.

Proof of Concept

Below is a screenshot with the application source code which contains *'CC_MD5'* hashing method:



Recommendations

DataArt recommends replacing the weak hashing algorithm in the application workflow with strong ones. For example, use SHA-256 (*'CC_SHA256'* from the *CommonCrypto* iOS library) or any other from *SHA-2* set.

Remediation

As of **May 27, 2020**, the issue was **still actual**. DataArt identified the application still uses weak hashing algorithms:

L5. Verbose error messages

Risk Rating: ~~LOW~~ REMEDIATED

CVSS: 3.7 (AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N)

Remediation Efforts: MEDIUM

Summary

DataArt noticed that in case of some errors occurred the application returned verbose information about the system including stack trace of the backend logic, internal paths and other sensitive information. All details provided within error descriptions (absolute paths, utilized libraries, etc.) will be useful for understanding the internal logic and structure of the application during preparation of further attacks.

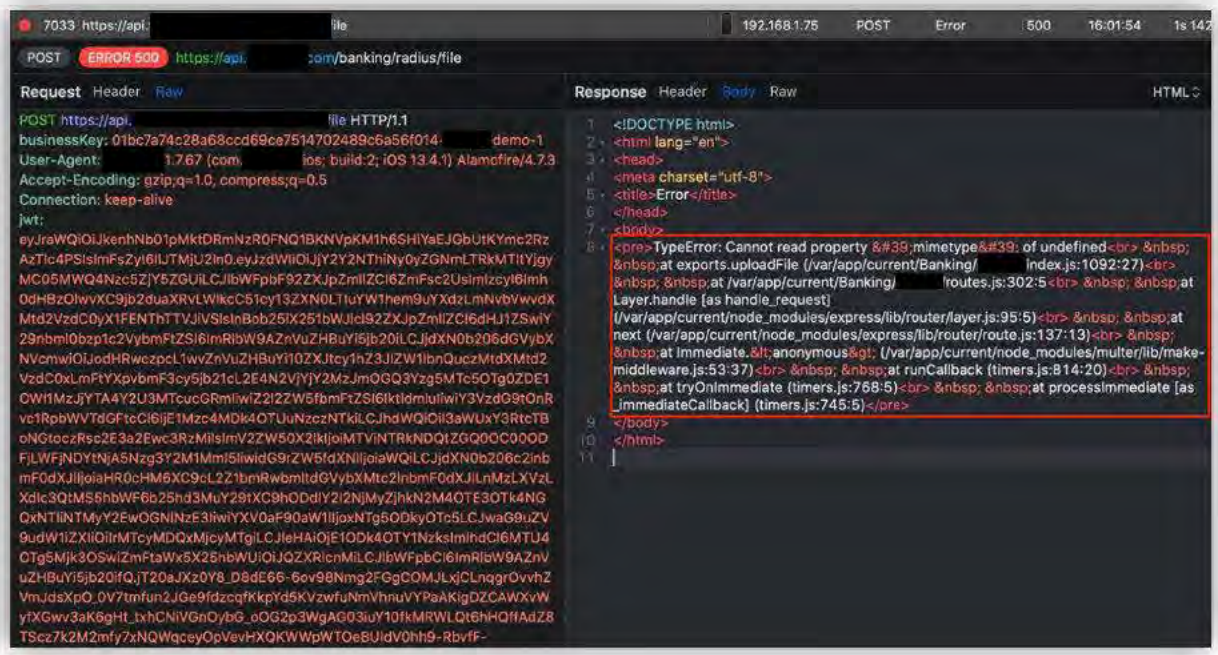
Affected Functionality

The issue affected the file uploading functionality in case a user submits incorrect input data:

- **POST** https://api[redacted]com/banking/[redacted]file

Proof of Concept

Another screenshot demonstrates the detailed error while uploading an incorrect file (the sent request contains empty body):

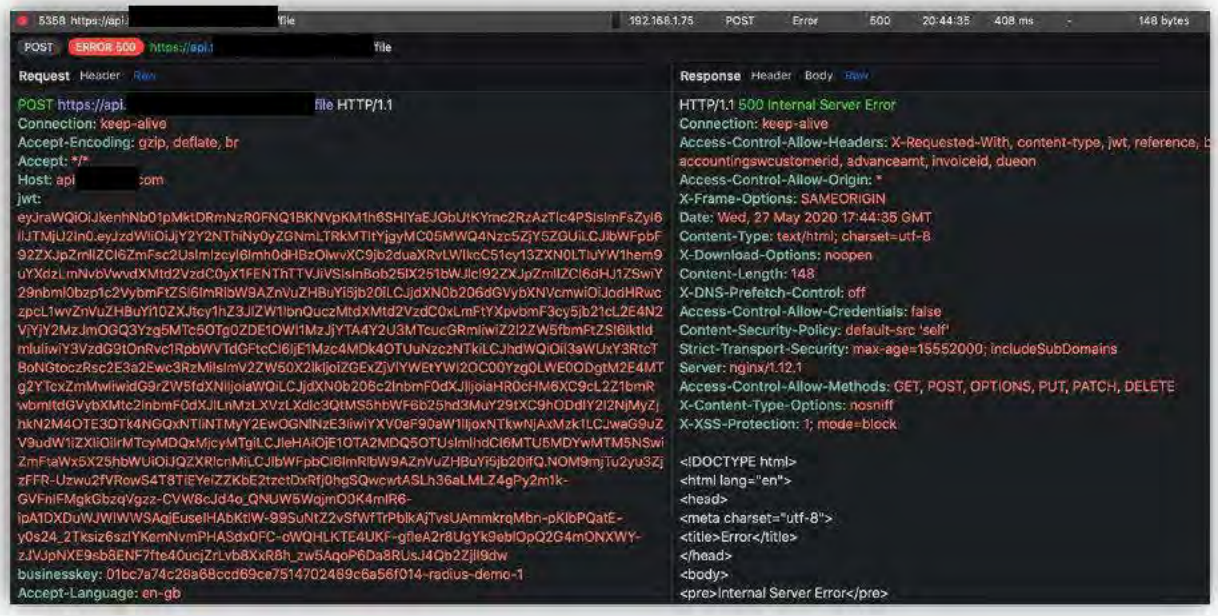


Recommendations

The application should never return internal information, verbose error messages or debug information back to a user. When an unexpected event occurs, the application should return the same generic message informing the user that an error occurred.

Remediation

As of **May 27, 2020**, the issue was **successfully remediated**. DataArt proved that currently the application server does not provide verbose information in cases where an error occurs:



L6. Platform information is disclosed in server's responses

Risk Rating: LOW

CVSS: 3.7 (AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N)

Remediation Efforts: LOW

Summary

DataArt noticed that the application server provided verbose information about the version of the utilized web server. Attackers can often use this type of information to more effectively target the system.

A typical scenario is the attacker accesses the application and discovers information about web server type by viewing the HTTP response headers. The attacker then looks up vulnerabilities that are known to exist within that version of the software and exploits an unpatched vulnerability.

- http://nginx.org/en/security_advisories.html
- https://www.cvedetails.com/vulnerability-list/vendor_id-10048/product_id-17956/version_id-267418/Nginx-Nginx-1.12.1.html

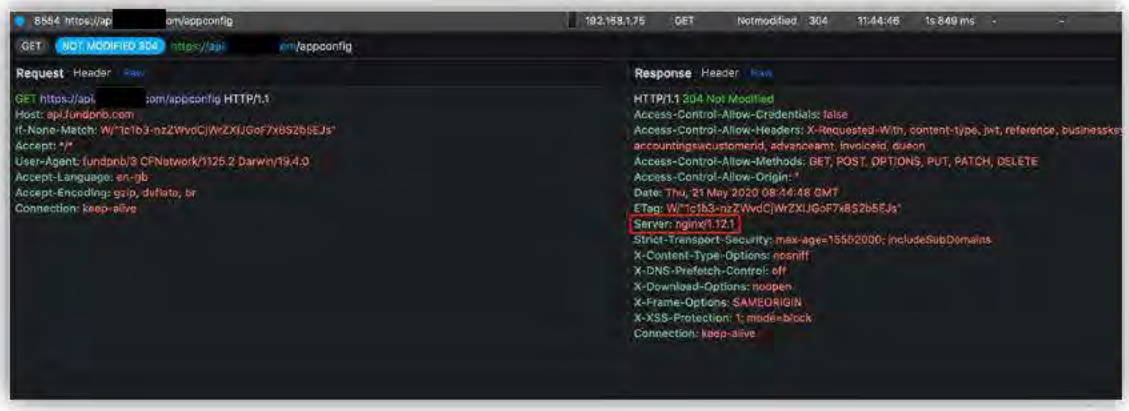
Affected Functionality

The issue affected the following server and the appropriate response header:

- https://api. com
  - *Server: nginx/1.12.1*

Proof of Concept

The images below show the example of server's response disclosing the detailed version of the utilized web server:



Recommendations

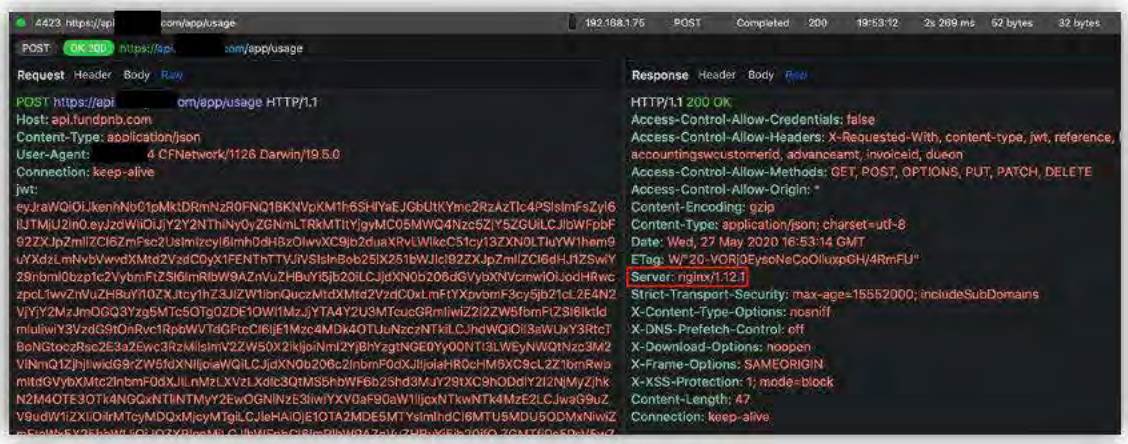
DataArt urges to remove any platform-related information from headers and bodies of server responses, including name of the utilized web framework.

Detailed information about possible solution for the issue in the context of nginx server can be found in the following article:

- https://www.getpagespeed.com/server-setup/nginx/how-to-remove-the-server-header-in-nginx

Remediation

As of **May 27, 2020**, the issue was **still actual**. The application server still provides verbose information about the utilized web server software.



L7. Using components with known vulnerabilities

Risk Rating: LOW

CVSS: 3.7 (AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N)

Remediation Efforts: MEDIUM

Summary

During the testing DataArt identified utilization of outdated version of the web server. The utilized version had publicly disclosed security issues which were remediated in the later releases. A potential malefactor could utilize these issues to attack the server.

Despite the fact that DataArt was not able to find a way to exploit the known security issues of vulnerable components of the system, using software components with known vulnerabilities could reduce the overall security of the solution.

Affected Functionality

The issue affected the following component:

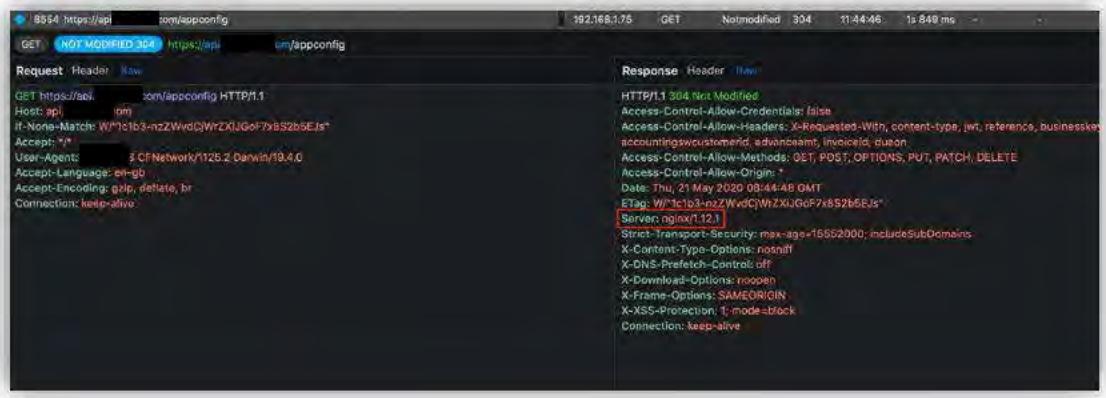
- https://api[redacted]com
  - *Nginx web server:*
    The utilized version: 1.12.1
    The latest version: 1.18.0

Known vulnerabilities for the utilized version of *Nginx* could be found via the links below:

- http://nginx.org/en/security_advisories.html
- https://www.cvedetails.com/vulnerability-list/vendor_id-10048/product_id-17956/version_id-267418/Nginx-Nginx-1.12.1.html

Proof of Concept

The screenshot below provides the details of the utilized version of the web server:

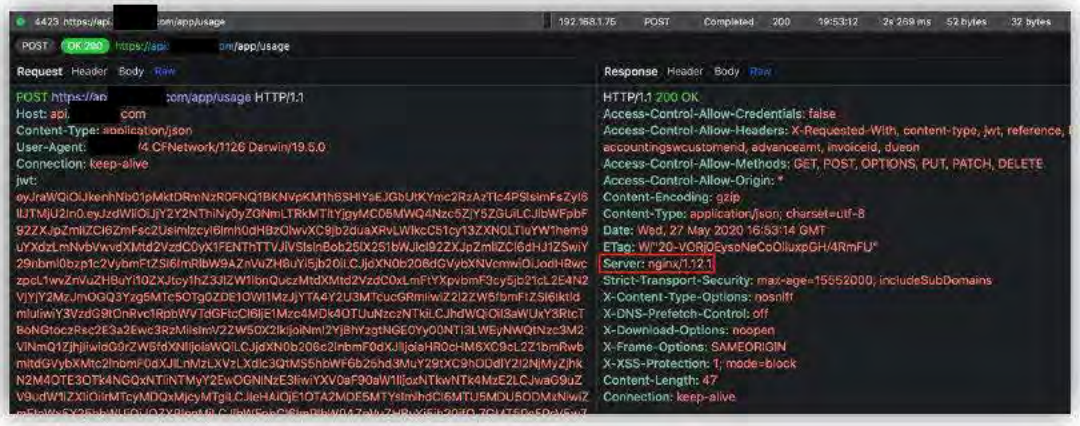


Recommendations

DataArt recommends implementing the security policy which requires utilization only the latest versions of software. Such approach could help to protect the solution against exploitation of publicly known vulnerabilities.

Remediation

As of **May 27, 2020**, the issue was **still actual**. DataArt identified that the outdated software is still utilized on the application server:



L8. Support of TLS ciphers suites without forward secrecy

Risk Rating: LOW

CVSS: 3.7 (AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N)

Remediation Efforts: LOW

Summary

DataArt noticed that all application servers supported TLS cipher suites that used RSA encryption for key exchange. Though the encryption algorithm was considered secure, a part of them did not provide Forward Secrecy (FS) feature protecting against the case when the intercepted traffic between client/server could be decrypted if a private key is stolen.

Affected Functionality

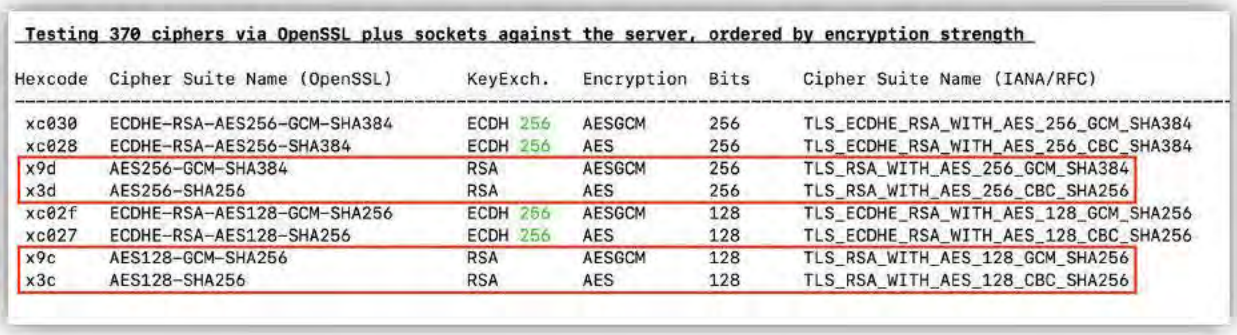
The issue affected all cipher suites except the ECDHE ones allowed by the application server (applicable for supported TLS versions):

- api [redacted] com
- cognito-idp.us-west-2.amazonaws.com
- [redacted] com

  AES128-SHA256
  AES256-SHA256
  AES128-GCM-SHA256
  AES128-GCM-SHA384

Proof of Concept

The images below show all cipher suites allowed by the application server. The weak ones are marked red:



Testing 370 ciphers via OpenSSL plus sockets against the server, ordered by encryption strength

Hexcode	Cipher Suite Name (OpenSSL)	KeyExch.	Encryption	Bits	Cipher Suite Name (IANA/RFC)
xc030	ECDHE-RSA-AES256-GCM-SHA384	ECDH 256	AESGCM	256	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
xc028	ECDHE-RSA-AES256-SHA384	ECDH 256	AES	256	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
x9d	AES256-GCM-SHA384	RSA	AESGCM	256	TLS_RSA_WITH_AES_256_GCM_SHA384
x3d	AES256-SHA256	RSA	AES	256	TLS_RSA_WITH_AES_256_CBC_SHA256
xc02f	ECDHE-RSA-AES128-GCM-SHA256	ECDH 256	AESGCM	128	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
xc027	ECDHE-RSA-AES128-SHA256	ECDH 256	AES	128	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
x9c	AES128-GCM-SHA256	RSA	AESGCM	128	TLS_RSA_WITH_AES_128_GCM_SHA256
x3c	AES128-SHA256	RSA	AES	128	TLS_RSA_WITH_AES_128_CBC_SHA256

Recommendations

It is strongly recommended to utilize only cipher suites allowing perfect forward secrecy features. Such approach should minimize possible security risks connected with decryption of previously intercepted encrypted traffic in case a private key has been compromised.

Detailed information about perfect forward secrecy could be found in the article below:

- https://www.keycdn.com/blog/perfect-forward-secrecy
- https://robotattack.org/
- https://blog.cloudflare.com/padding-oracles-and-the-decline-of-cbc-mode-ciphersuites/

Since the affected application utilizes Amazon CloudFront, the issue could be fixed by applying up-to-date TLS security policies for the corresponding AWS resources:

- CloudFront – 2018 is recommended in AWS documentation. The Cloud Conformity page explains where the corresponding TLS policy settings can be found. Please note that there's no way to deselect particular TLS cipher suites for CloudFront, as these sets of cipher suites are predefined by AWS-supplied policies;
- AWS S3 – it's necessary to verify that Secure Transfer is enforced for all buckets (including those which serve as storage for CloudFront distributions).

Remediation

As of **May 27, 2020**, the issue was **still actual**. DataArt identified that outdated cipher suites are still allowed by the server:

1. cognito-idp.us-west-2.amazonaws.com:

```
  Supported Server Cipher(s):
Preferred TLSv1.2  128 bits  ECDHE-RSA-AES128-GCM-SHA256   Curve P-256 DHE 256
Accepted  TLSv1.2  128 bits  ECDHE-RSA-AES128-SHA256       Curve P-256 DHE 256
Accepted  TLSv1.2  128 bits  ECDHE-RSA-AES128-SHA          Curve P-256 DHE 256
Accepted  TLSv1.2  256 bits  ECDHE-RSA-AES256-GCM-SHA384   Curve P-256 DHE 256
Accepted  TLSv1.2  256 bits  ECDHE-RSA-AES256-SHA384       Curve P-256 DHE 256
Accepted  TLSv1.2  256 bits  ECDHE-RSA-AES256-SHA          Curve P-256 DHE 256
Accepted  TLSv1.2  128 bits  AES128-GCM-SHA256
Accepted  TLSv1.2  128 bits  AES128-SHA256
Accepted  TLSv1.2  128 bits  AES128-SHA
Accepted  TLSv1.2  256 bits  AES256-GCM-SHA384
Accepted  TLSv1.2  256 bits  AES256-SHA256
Accepted  TLSv1.2  256 bits  AES256-SHA
Preferred TLSv1.1  128 bits  ECDHE-RSA-AES128-SHA          Curve P-256 DHE 256
Accepted  TLSv1.1  256 bits  ECDHE-RSA-AES256-SHA          Curve P-256 DHE 256
Accepted  TLSv1.1  128 bits  AES128-SHA
Accepted  TLSv1.1  256 bits  AES256-SHA
Preferred TLSv1.0  128 bits  ECDHE-RSA-AES128-SHA          Curve P-256 DHE 256
Accepted  TLSv1.0  256 bits  ECDHE-RSA-AES256-SHA          Curve P-256 DHE 256
Accepted  TLSv1.0  128 bits  AES128-SHA
Accepted  TLSv1.0  256 bits  AES256-SHA
```

2. [redacted]com:

```
  Supported Server Cipher(s):
Preferred TLSv1.2  128 bits  ECDHE-RSA-AES128-GCM-SHA256   Curve P-256 DHE 256
Accepted  TLSv1.2  128 bits  ECDHE-RSA-AES128-SHA256       Curve P-256 DHE 256
Accepted  TLSv1.2  256 bits  ECDHE-RSA-AES256-GCM-SHA384   Curve P-256 DHE 256
Accepted  TLSv1.2  256 bits  ECDHE-RSA-AES256-SHA384       Curve P-256 DHE 256
Accepted  TLSv1.2  128 bits  AES128-GCM-SHA256
Accepted  TLSv1.2  256 bits  AES256-GCM-SHA384
Accepted  TLSv1.2  128 bits  AES128-SHA256
```

L9. Missing debugger detection

Risk Rating: LOW

CVSS: 3.7 (AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N)

Remediation Efforts: MEDIUM

Summary

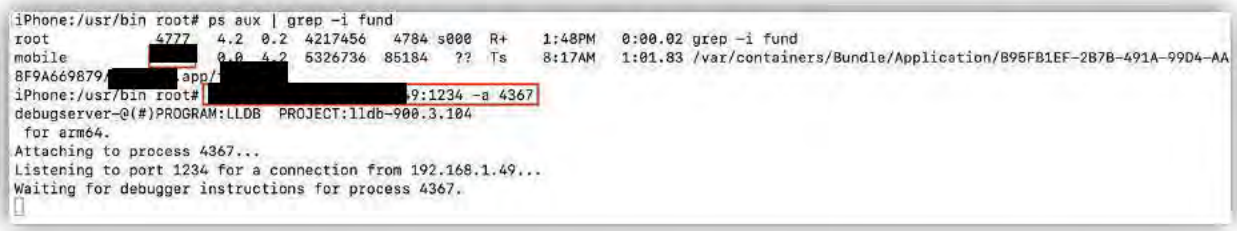
DataArt noticed that the application did not implement security measures to protect against debugging. Debugging and exploring applications are significantly helpful during reversing of the application source code. Using a debugger, a reverse engineer can not only track critical variables but also read and modify memory. The disclosed information is often used for further tailored attacks.

Affected Functionality

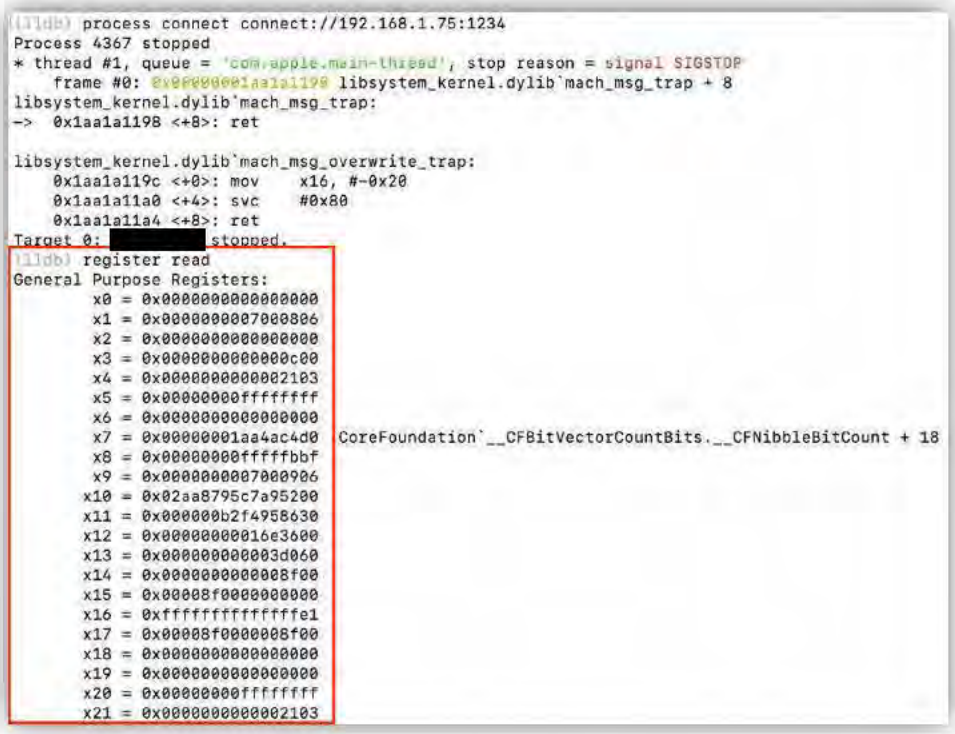
The issue affected the iOS application.

Proof of Concept

The image below shows the evidence that the debug server was successfully started for the application`s process:



The LLDB debugger was attached to the server which allowed to read application's registers, as shown below.



Recommendations

DataArt recommends implementing anti-debugging security measures. The detailed information can be found via the following link:

- https://developer.apple.com/library/archive/qa/qa1361/_index.html

Remediation

As of **May 27, 2020**, the issue was **still actual**. DataArt noticed the application still has not implemented security measures to protect against debugging.

Conclusion

DataArt completed the penetration testing of the [redacted] iOS mobile application (version 1.7.67). This testing was based on the technologies and known threats as of the date of this document. All the security issues discovered during that exercise were analyzed and described in this report. DataArt recommends that all modifications suggested in this document be performed in order to ensure the overall security of the application.

Please note that as technologies and risks change over time, the vulnerabilities associated with the operation of systems described in this report, as well as the actions necessary to reduce the exposure to such vulnerabilities, will also change.