https://e-discovery.ng/

WEB APPLICATION PENETRATION TESTING

Report for:	
Date:	

Table of Contents

Executive Summary

E-Discovery (Provider) was contracted by _____ (Client) to carry out a **web application** penetration test.

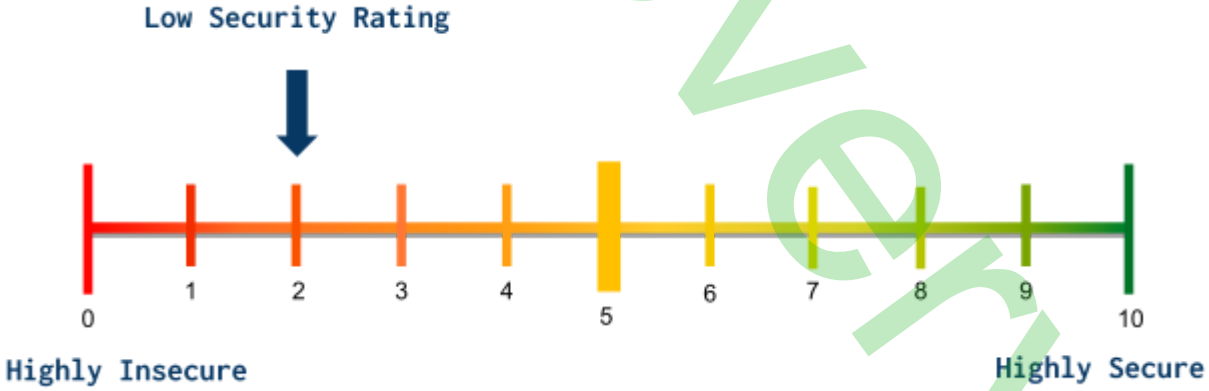
The application provides customers the ability to submit order requests, review design, leave feedback, etc.

The penetration test was conducted between **08.02.2021 - 26.02.2021**.

The penetration test has the following objectives:

- identify technical and functional vulnerabilities
- evaluate a severity level (ease of use, impact on information systems, etc.);
- make a prioritized list of recommendations to address identified weaknesses

According to our research after performing the penetration testing, the security rating of the client's **web application** was identified as **Low**.



Scope

The following list of the information systems was the scope of the penetration testing.

#	Name	Description	Version
1.	client.com www.client.com h5.client.com openws.client.com ws-manager.client.com ws.client.com gitlab.infra.client.com registry.infra.client.com nexus.infra.client.com wiki.infra.client.com	Web	
2.	35.220.000.000 35.240.00.000 35.190.00.000 35.240.00.000 35.220.000.000 130.210.00.00	IP	
3.	api.Client.com openapi.Client.com (https://github.com/Client/Client-official-api-docs)	API	

Methodology

The testing methodology is based on generally accepted industry-wide approaches to perform penetration testing for web applications (OWASP Testing Guide);

Application-level penetration tests include, at the minimum, checking for the following types of vulnerabilities:

- injections, in particular, SQL injections, NoSQL, XPath, etc.;
- Local File Inclusion (LFI), Remote File Inclusion (RFI);
- Cross-Site Scripting (XSS);
- errors in access control mechanisms (for example, unsafe direct links to objects, lack of restriction of access by URL, directory traversal and lack of restriction of user access rights to functions);
- Cross-Site Request Forgery (CSRF);
- web server configuration errors;
- incorrect error handling;
- Counteracting the compromise of authentication mechanisms and session management (Session Management Testing);

Severity Definition

The level of severity of each vulnerability is determined based on the potential impact of loss from successful exploitation as well as ease of exploitation, the existence of exploits in public access and other factors.

Severity	Description
High	High-level vulnerabilities are easy to exploit and may provide an attacker with complete control of the affected systems, leading to significant data loss or downtime. There are exploits or PoC available in public access.
Medium	Medium-level vulnerabilities are much harder to exploit and may not provide the same access to affected systems. Exploits or PoCs aren't available in public access. Exploitation provides only very limited access.
Low	Low-level vulnerabilities exploitation is extremely difficult, or impact is minimal.
Info	Information-level vulnerabilities provide an attacker with information that may assist them in conducting subsequent attacks against target information systems or against other information systems, which belong to an organization.

Key Findings

■■■■ Rate limit bypass via X-Forwarded-For

#1	Description

X-Forwarded-For is a well-established HTTP header used by proxies to pass along other IP addresses in the request. This is often the same as CF-Connecting-IP, but there may be multiple layers of proxies in a request path.

There is dynamically changing value can attackers do brute force 6-digits approve code and other attacks which are based on brute force method.

Evidence

Steps to reproduce:

1. Get request for restore password
2. Input some code
3. Intercept request and set header X-Forwarded-For with something value
4. The count of the number of attempts will be restored to the initial value

Request:

```
POST [redacted].com/api/user_findPwd HTTP/1.1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.12; rv:64.0) Gecko/20100101 Firefox/64.0
Accept: application/json, text/plain, */*
Accept-Language: uk-UA,uk;q=0.8,en-US;q=0.5,en;q=0.3
Referer: [redacted]forget/en
Content-Type: application/x-www-form-urlencoded;charset=UTF-8
X-Forwarded-For: TEST12312X
Content-Length: 150
Connection: keep-alive
[redacted]
Host: [redacted]

loginName=bus[redacted]-mail.net&loginType=1&pwdType=0&emailCode=718315&newPwd=
146d2f289749a5c70b1dbe65ef6[redacted]&reNewPwd=146d2f289749a5c70b1dbe65ef6[redacted]
```

Recommendations

- check the value of headers
- add a “one-time token”

■■■■ Broken Authentication and Session Management

#2 Description

Incorrect logic in the transfer of the session between domains allows the user to intercept another user's session.

The WebSocket application at client.com is responsible for mediating the session for the main casino application, which can be located on one of the mirrors, for example, at client.com and client.com.

This functionality is used to dynamically transfer the session to different mirrors, which allows the user not to log into the system every time when changing such a mirror. Also, the WebSocket of the application on client.com does not have a built-in validation of the domain from which the session request comes, which allows getting a user session for any domain.

An example of such session interception is located at https://ps29.net/client-dwju3726ks/. This page contains the authorization.js (https://www.client.com/files/js/authorization.js) code that pinup uses for authorization.

Evidence

Steps to reproduce:

1. Login to any account on client domain
2. Go to https://ps29.net/client-dwju3726ks/

Recommendations

- Add domain validation

Email disclosure via Forgot password

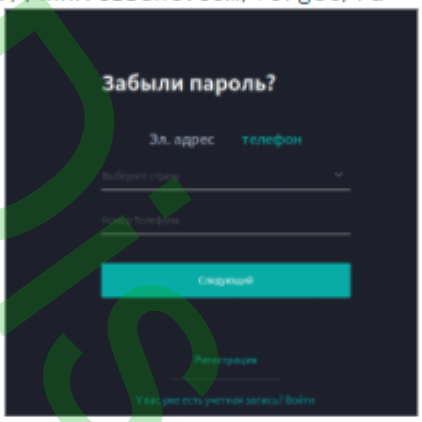
#3	Description

It is possible to get information about registered email.

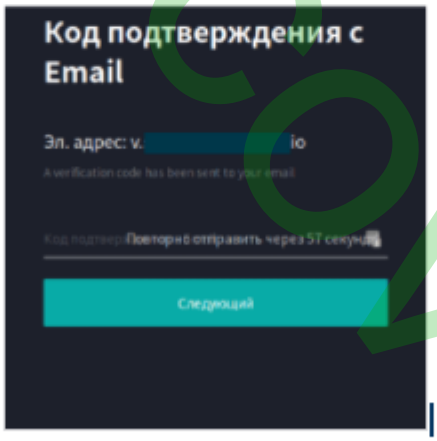
Evidence

Steps to reproduce:

1. Go to page https://www.client.com/forget/ru



Response:



Recommendations

- Shouldn`t show the email address when restoring a password via the phone.

User enumeration

#4 Description

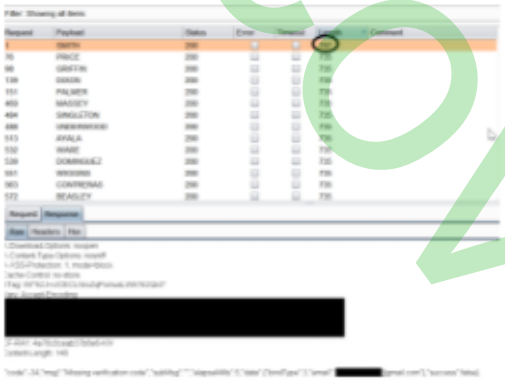
The scope of this test is to verify whether it's possible to collect a set of valid usernames by interacting with the authentication mechanism of the application. This test will be useful for brute force testing, in which we verify if, given a valid username, it's possible to find a corresponding password. Often, web applications reveal when a username exists in a system, either due to a misconfiguration or as a design decision.

For example, sometimes, when we submit wrong credentials, we receive a message stating that either the username is present in the system or the provided password is wrong. The information obtained can be used by an attacker to gain a list of users in the system. This information can be used to attack the web application, for example, through a brute force or default username/password attack.

Evidence

Steps to reproduce:

1. Intercept request POST /api/user_findPwd
2. Send request to Intruder
3. Set payload to loginName=<email>&loginType=1&pwdType=0
4. Run attack



Recommendations

- It's recommended not to show whether the user is logged in the system or not

Vulnerability Lucky13 and BREACH

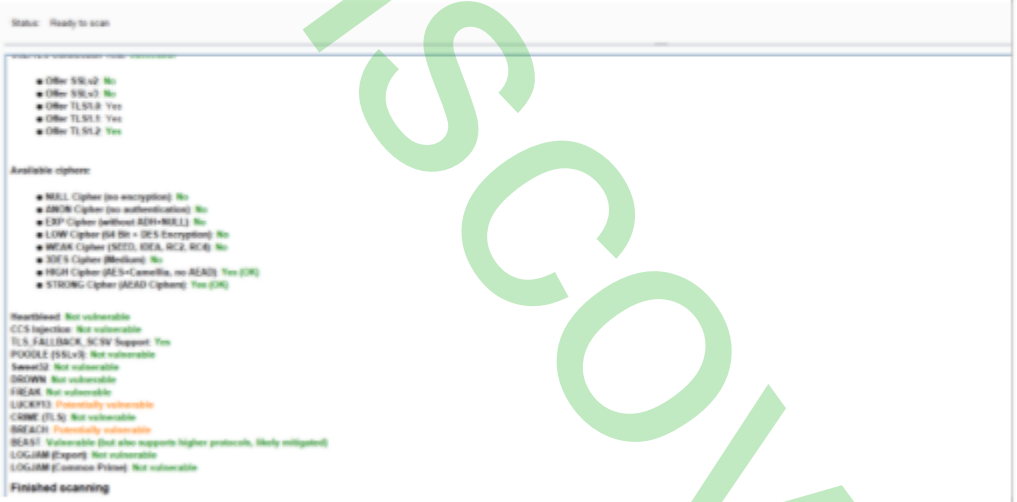
#5	Description

BREACH

Short for Browser Exploit Against SSL/TLS, BREACH is a browser exploit against SSL/TLS that was revealed in late September 2011. This attack leverages weaknesses in cipher block chaining (CBC) to exploit the Secure Sockets Layer (SSL)/Transport Layer Security (TLS) protocol. The CBC vulnerability can enable man-in-the-middle (MITM) attacks against SSL in order to silently decrypt and obtain authentication tokens, thereby providing hackers access to data passed between a web server and the Web browser accessing the server.

Evidence

Scanning https://www.client.com with SSLscan



Recommendations

- Disable TLS 1.0 and make user connections using TLS 1.1 or TLS 1.2 protocols which are immune to the BEAST attack. TLS 1.0 is now considered insecure. Disabling the TLS 1.0 protocol improves the overall security.
- Avoid using TLS in CBC-mode and switch to AEAD algorithms.

Cacheable HTTPS response

#6	Description
Unless directed otherwise, browsers may store a locally cached copy of content received from web servers. Some browsers, including Internet Explorer, cache content accessed via HTTPS. If sensitive information in application responses is stored in the local cache, then this may be retrieved by other users who have access to the same computer at a future time.(Cache-control: no-store, Pragma: no-cache)	
Recommendations	
Add the following headers: – Cache-control: no-store – Pragma: no-cache	

Appendix A. OWASP Testing Checklist

Category	Test Name	Result
Information Gathering		
OTG-INFO-001	Conduct Search Engine Discovery and Reconnaissance for Information Leakage	Tested
OTG-INFO-002	Fingerprint web server	Tested
OTG-INFO-003	Review Webserver Metafiles for Information Leakage	Tested
OTG-INFO-004	Enumerate Applications on Web server	Tested
OTG-INFO-005	Review Webpage Comments and Metadata for Information Leakage	Tested
OTG-INFO-006	Identify application entry points	Tested
OTG-INFO-007	Map execution paths through application	Tested
OTG-INFO-008	Fingerprint Web Application Framework	Tested
OTG-INFO-009	Fingerprint Web Application	Tested
OTG-INFO-010	WAF	Tested
Configuration and Deploy Management Testing		
OTG-CONFIG-001	Test Network/Infrastructure Configuration	Tested
OTG-CONFIG-002	Test Application Platform Configuration	Tested
OTG-CONFIG-003	Test File Extensions Handling for Sensitive Information	Tested
OTG-CONFIG-004	Backup and Unreferenced Files for Sensitive Information	Tested
OTG-CONFIG-005	Enumerate Infrastructure and Application Admin Interfaces	Tested
OTG-CONFIG-006	Test HTTP Methods	Tested
OTG-CONFIG-007	Test HTTP Strict Transport Security	Tested
OTG-CONFIG-008	Test RIA cross domain policy	Tested
Identity Management Testing		
OTG-IDENT-001	Test Role Definitions	N/A
OTG-IDENT-002	Test User Registration Process	Tested
OTG-IDENT-003	Test Account Provisioning Process	N/A
OTG-IDENT-004	Testing for Account Enumeration and Guessable User Account	Tested
OTG-IDENT-005	Testing for Weak or unenforced username policy	Tested
OTG-IDENT-006	Test Permissions of Guest/Training Accounts	N/A
OTG-IDENT-007	Test Account Suspension/Resumption Process	Tested
Authentication Testing		
OTG-AUTHN-001	Testing for Credentials Transported over an Encrypted Channel	Tested
OTG-AUTHN-002	Testing for default credentials	N/A

OTG-AUTHN-003	Testing for Weak lock out mechanism	Tested
OTG-AUTHN-004	Testing for bypassing authentication schema	Tested
OTG-AUTHN-005	Test remember password functionality	Tested
OTG-AUTHN-006	Testing for Browser cache weakness	Tested
OTG-AUTHN-007	Testing for Weak password policy	Tested
OTG-AUTHN-008	Testing for Weak security question/answer	Tested
OTG-AUTHN-009	Testing for weak password change or reset functionalities	Tested
OTG-AUTHN-010	Testing for Weaker authentication in alternative channel	Tested
Authorization Testing		
OTG-AUTHZ-001	Testing Directory traversal/file include	Tested
OTG-AUTHZ-002	Testing for bypassing authorization schema	Tested
OTG-AUTHZ-003	Testing for Privilege Escalation	Tested
OTG-AUTHZ-004	Testing for Insecure Direct Object References	Tested
Session Management Testing		
OTG-SESS-001	Testing for Bypassing Session Management Schema	Tested
OTG-SESS-002	Testing for Cookies attributes	Tested
OTG-SESS-003	Testing for Session Fixation	Tested
OTG-SESS-004	Testing for Exposed Session Variables	Tested
OTG-SESS-005	Testing for Cross Site Request Forgery	Tested
OTG-SESS-006	Testing for logout functionality	Tested
OTG-SESS-007	Test Session Timeout	Tested
OTG-SESS-008	Testing for Session puzzling	Tested
Data Validation Testing		
OTG-INPVAL-001	Testing for Reflected Cross Site Scripting	Tested
OTG-INPVAL-002	Testing for Stored Cross Site Scripting	Tested
OTG-INPVAL-003	Testing for HTTP Verb Tampering	Tested
OTG-INPVAL-004	Testing for HTTP Parameter pollution	Tested
OTG-INPVAL-005	Testing for SQL Injection	Tested
OTG-INPVAL-006	Testing for LDAP Injection	Tested
OTG-INPVAL-007	Testing for ORM Injection	Tested
OTG-INPVAL-008	Testing for XML Injection	Tested
OTG-INPVAL-009	Testing for SSI Injection	Tested
OTG-INPVAL-010	Testing for XPath Injection	Tested
OTG-INPVAL-011	IMAP/SMTP Injection	Tested
OTG-INPVAL-012	Testing for Code Injection	Tested
OTG-INPVAL-013	Testing for Command Injection	Tested
Error Handling		
OTG-ERR-001	Analysis of Error Codes	Tested
OTG-ERR-002	Analysis of Stack Traces	Tested

Cryptography		
OTG-CRYPST-001	Testing for Weak SSL/TSL Ciphers, Insufficient Transport Layer Protection	Tested
OTG-CRYPST-002	Testing for Padding Oracle	Tested
OTG-CRYPST-003	Testing for Sensitive information sent via unencrypted channels	Tested
Business Logic Testing		
OTG-BUSLOGIC-001	Test Business Logic Data Validation	Tested
OTG-BUSLOGIC-002	Test Ability to Forge Requests	Tested
OTG-BUSLOGIC-003	Test Integrity Checks	Tested
OTG-BUSLOGIC-004	Test for Process Timing	Tested
OTG-BUSLOGIC-005	Test Number of Times a Function Can be Used Limits	Tested
OTG-BUSLOGIC-006	Testing for the Circumvention of Work Flows	Tested
OTG-BUSLOGIC-007	Test Defenses Against Application Mis-use	Tested
OTG-BUSLOGIC-008	Test Upload of Unexpected File Types	Tested
OTG-BUSLOGIC-009	Test Upload of Malicious Files	Tested
Client Side Testing		
OTG-CLIENT-001	Testing for DOM based Cross Site Scripting	Tested
OTG-CLIENT-002	Testing for JavaScript Execution	Tested
OTG-CLIENT-003	Testing for HTML Injection	Tested
OTG-CLIENT-004	Testing for Client Side URL Redirect	Tested
OTG-CLIENT-005	Testing for CSS Injection	Tested
OTG-CLIENT-006	Testing for Client Side Resource Manipulation	Tested
OTG-CLIENT-007	Test Cross Origin Resource Sharing	Tested
OTG-CLIENT-008	Testing for Cross Site Flashing	Tested
OTG-CLIENT-009	Testing for Clickjacking	Tested
OTG-CLIENT-010	Testing WebSockets	Tested
OTG-CLIENT-011	Test Web Messaging	Tested
OTG-CLIENT-012	Test Local Storage	Tested

Appendix B. Pentesting Tools

Scope	Tools Used
Application Security	Acunetix 11 BurpSuite 1.7.30 Owasp-zap Maltego Classic Detectify Sqlmap
Network Security	Nmap Recon-ng Nessus Nexpose