



EurekaSelect Web Penetration Test Report

CONFIDENTIAL

DATE: 12/7/2023



Use and Disclosure of Data

This document includes data that shall not be disclosed outside the Intended Recipient and shall not be duplicated, used, or disclosed, in whole or in part, for any purpose without the consent of Bentham Science. The data subject to this restriction are contained in all pages of this proposal.

Outline

EXECUTIVE SUMMARY	4
APPROACH	4
SCOPE	4
METHODOLOGY	4
TOOLS	5
TECHNIQUES	5
SEVERITY RATINGS	6
FINDINGS OVERVIEW	6
FINDINGS DETAIL	8
SESSION HIJACKING	8
CROSS SITE SCRIPTING (STORED XSS)	11
BRUTE FORCE ATTACK	14
ACCOUNT LOCKOUT POLICY IS NOT ENABLED	15
HTTP STRICT TRANSPORT SECURITY (HSTS) ERRORS AND WARNINGS	16
CLICK JACKING	17
CONTENT SECURITY POLICY (CSP) NOT IMPLEMENTED	19
COOKIE NOT MARKED AS HTTPONLY & SECURE	20
OUT-OF-DATE VERSION (JQUERY)	21

MISSING X-XSS-PROTECTION HEADER	22
ROBOTS.TXT FILE DETECTED	23
AUTOCOMPLETE IS ENABLED	24

Executive Summary

This document contains the findings of penetration tests conducted by 10Pearls of EurekaSelect web application. ITS engaged 10Pearls to perform a series of penetration tests, both manual and automated, since Oct 2023. This most recent test was performed from Oct 20th - Nov 24th, 2023.

In this most recent test, 12 findings are highlighted. 2 critical vulnerabilities, 2 high risk vulnerabilities, 5 medium risk vulnerabilities, 1 low risk vulnerability and 2 informational level vulnerability findings.

Within this report are the specific findings along with their associated risk level. Additionally, further details regarding each finding, an evaluation, and remediation information have also been included in this report.

Approach

Vulnerability scans and penetration testing attacks were conducted on the EurekaSelect web application both unauthenticated as well as authenticated using different credentials. Different subscriber accounts were created for SecOps team to conduct the penetration testing during predefined test windows.

Scope

The scope included both Dynamic application security testing (DAST), as well as Static application security testing performed on the EurekaSelect web application. The results listed below are limited to findings of confirmed and potential vulnerabilities. Tests on the EurekaSelect web application were conducted unauthenticated as well as authenticated, using both different accounts

The results listed below are prioritized by criticality and should be remediated once an evaluation of the results has been conducted by ITS.

Methodology

The 10Pearls testing methodology is based on the OWASP Web Application Security Testing methodology. Specifically, it involves a tester leveraging various tools and techniques in a black-box approach to identify and then attempt to confirm potential vulnerabilities. The testing is

CONFIDENTIAL 10PEARLS // EUREKASELECT WEB PENETRATION TEST REPORT // PAGE 4 OF 25

performed in two phases (passive and active), each of which are repeated in both unauthenticated as well as authenticated use cases. The passive testing leverages tools such as Burp Suite, Firebug, and others to gather information and identify entry points which may be vulnerable. The active testing then attempts to prove vulnerability through the use of tools and techniques.

Phases of penetration testing activities include the following:

- Planning – Customer goals are gathered and rules of engagement obtained.
- Discovery – Perform scanning and enumeration to identify potential vulnerabilities, weak areas, and exploits.
- Attack – Confirm potential vulnerabilities through exploitation and perform additional discovery upon new access.
- Reporting – Document all found vulnerabilities and exploits, failed attempts, and company strengths and weaknesses.

Tests on the EurekaSelect web application included authentication testing, session management, access control, malicious input control, cryptography, data protection, communication security as well as other business logic and file analysis.

TOOLS

Vulnerability tests were conducted on EurekaSelect web application using several tools, primarily:

- Burp Suite Pro, ZAP Proxy
- Firefox/Chrome Add-ons – FireBug, Hax Bar, Live HTTP Header, Cookie Manager,
- Nmap, Nikto, Kali Linux OS tools
- Wappalyzer, DirBuster
- Wireshark, Goldeneye, SQLmap,
- and more

TECHNIQUES

The severity assigned to each vulnerability was calculated using OWASP Website security project. Additionally, focus was given to the following areas of manual testing and analysis in the EurekaSelect web application:

- Login mechanism and session manipulation involving passwords, cookies and tokens
- Functionality and flaws that are user and app specific
- Application logic weaknesses that allow for manual manipulation of the business workflow and specific input fields
- Password policy exploitation, including complexity enforcement and intruder lockout capabilities

Severity Ratings

The following table outlines the severity ratings used within this report.

Severity	CVSS v3 Score Range	Definition
Critical	9.0-10.0	Exploitation is straightforward and usually results in system-level compromise. It is advised to form a plan of action and patch immediately.
High	7.0-8.9	Exploitation is more difficult but could cause elevated privileges and potentially a loss of data or downtime. It is advised to form a plan of action and patch as soon as possible.
Medium	4.0-6.9	Vulnerabilities exist but are not exploitable or require extra steps such as social engineering. It is advised to form a plan of action and patch after high-priority issues have been resolved.
Low	0.1-3.9	Vulnerabilities are non-exploitable but would reduce an organization's attack surface. It is advised to form a plan of action and patch during the next maintenance window.
Informational	N/A	No vulnerability exists. Additional information is provided regarding items noticed during testing, strong controls, and additional documentation.

Findings Overview

Severity	Number
Critical	2
High	2
Medium	5
Low	1
Informational	2

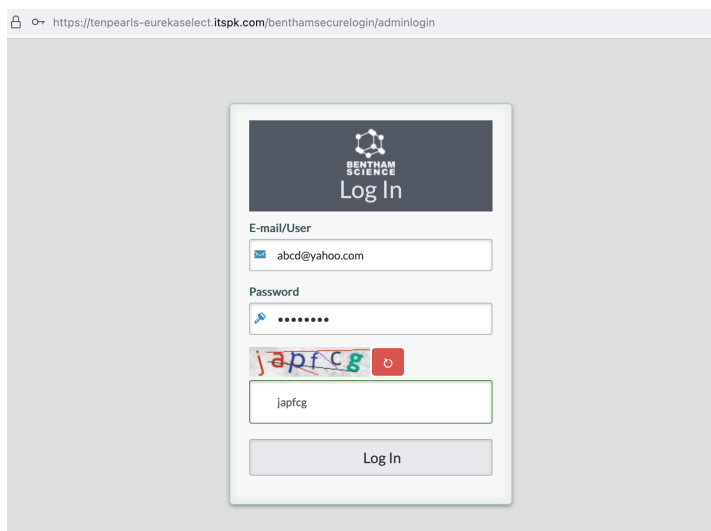
Findings Detail

SESSION HIJACKING

Risk Level	Critical
Finding	Application is vulnerable to session hijacking attack.
Explanation	The Session Hijacking attack consists of the exploitation of the web session control mechanism, which is normally managed for a session token. Because http communication uses many different TCP connections, the web server needs a method to recognize every user's connections. The most useful method depends on a token that the Web Server sends to the client browser after a successful client authentication. A session token is normally composed of a string of variable width and it could be used in different ways, like in the URL, in the header of the http requisition as a cookie, in other parts of the header of the http request, or yet in the body of the http requisition. The Session Hijacking attack compromises the session token by stealing or predicting a valid session token to gain unauthorized access to the Web Server.
References	https://owasp.org/www-community/attacks/Session_hijacking_attack

Proof of Concept

We login from our first user and intercept the requests and responses to the application server.



Intercept

HTTP request

WebSockets history

Proxy settings

Request to https://tenpearls-eurekaselect.itspk.com:443 [18.221.232.97]

Forward

Drop

Intercept is on

Action

Open browser

Pretty

Raw

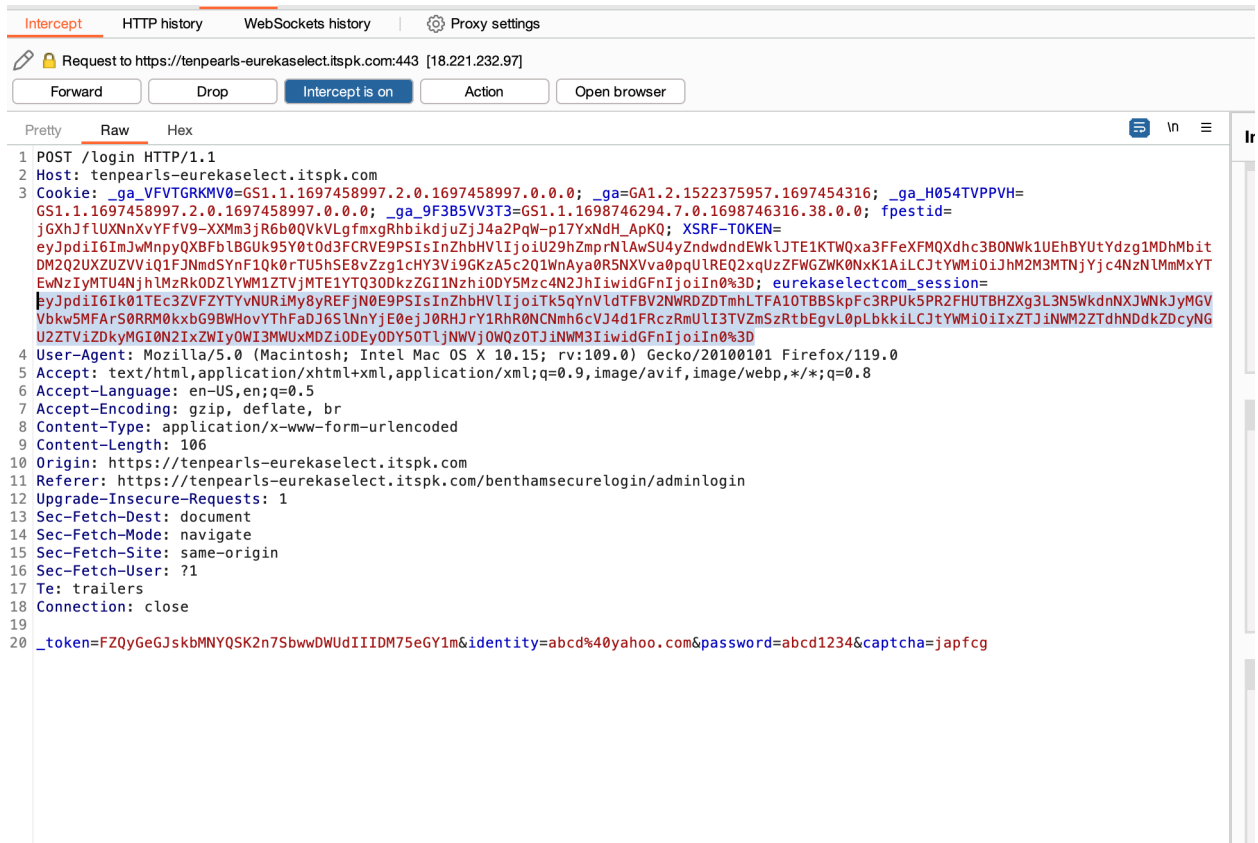
Hex

1 POST /login HTTP/1.1

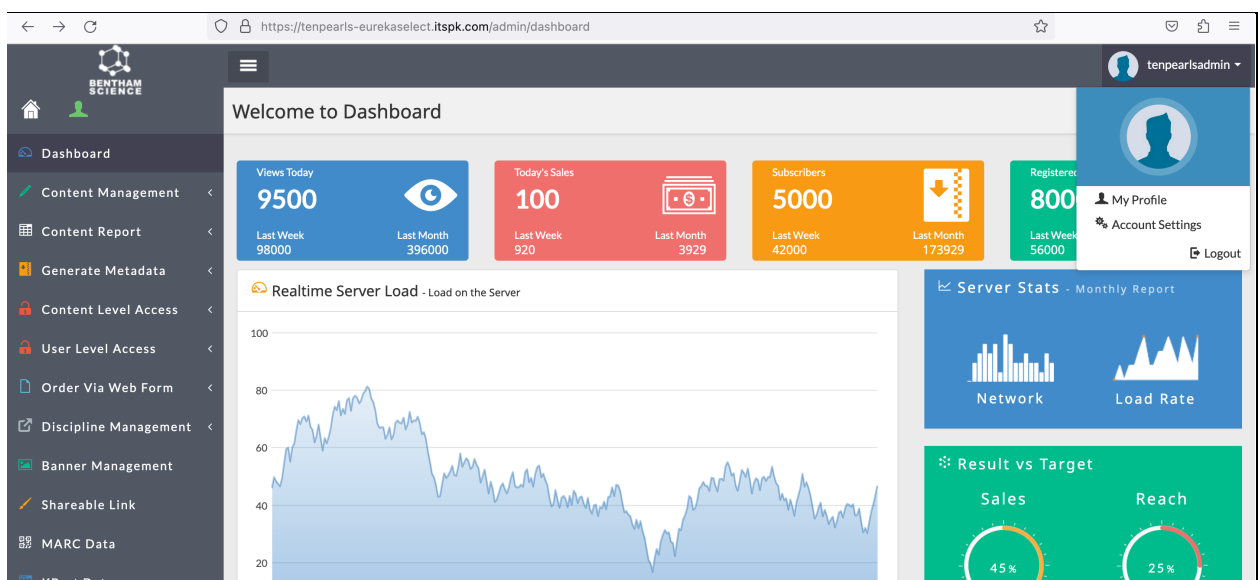
2 Host: tenpearls-eurekaselect.itspk.com

3 Cookie: _ga_VFVTGRKMV0=GS1.1.1697458997.2.0.1697458997.0.0.0; _ga=GA1.2.1522375957.1697454316; _ga_H054TVPPVH=GS1.1.1697458997.2.0.1697458997.0.0.0; _ga_9F3B5VV3T3=GS1.1.1698746294.7.0.1698746316.38.0.0; fpestid=jGhJfLUXNnXvYFfV9-XXMm3jR6b0QVkvLgfmXgRhbkidjuZjJ4a2PqW-p17YxNdH_ApKQ; XSRF-TOKEN=eyJpdii6ImJwMnpYQXBfbLGUk95Y0t0d3FCRVE9PSiInZhbHVlIjoieU29hZmpraWlAwSU4yZndwdndEwkJTE1KTQwxa3FfeFXMQxdhc3BONWk1UEhBYUtYdzg1MDhmMbItDM2Q2UXZUVVQ1FjNm5YnF1k0rTU5hSE8vZzg1cHY3Vi9GKzA5c2Q1WmYya0R5NXVva0pqULREQ2xQzZFWGZWk0Nk1A1LCJtYWMiOiJhM2M3MTNjYjYjY2NzNmMmMxYTUwNmZiYmY0NjhmZkR0ODZlYmM1ZTVjMTE1YTQ3ODkzZGI1NzhiODY5Mzc4N2JhIiwidGFnIjoieU29hZmpraWlAwSU4yZndwdndEwkJTE1KTQwxa3FfeFXMQxdhc3BONWk1UEhBYUtYdzg1MDhmMbItDM2Q2UXZUVVQ1FjNm5YnF1k0rTU5hSE8vZzg1cHY3Vi9GKzA5c2Q1WmYya0R5NXVva0pqULREQ2xQzZFWGZWk0Nk1A1LCJtYWMiOiJhM2M3MTNjYjYjY2NzNmMmMxYTUwNmZiYmY0NjhmZkR0ODZlYmM1ZTVjMTE1YTQ3ODkzZGI1NzhiODY5Mzc4N2JhIiwidGFnIjoieU29hZmpraWlAwSU4yZndwdndEwkJTE1KTQwxa3FfeFXMQxdhc3BONWk1UEhBYUtYdzg1MDhmMbItDM2Q2UXZUVVQ1FjNm5YnF1k0rTU5hSE8vZzg1cHY3Vi9GKzA5c2Q1WmYya0R5NXVva0pqULREQ2xQzZFWGZWk0Nk1A1LCJtYWMiOiJhM2M3MTNjYjYjY2NzNmMmMxYTUwNmZiYmY0NjhmZkR0ODZlYmM1ZTVjMTE1YTQ3ODkzZGI1NzhiODY5Mzc4N2JhIiwidGFnIjoieU29hZmpraWlAwSU4yZndwdndEwkJTE1KTQwxa3FfeFXMQxdhc3BONWk1UEhBYUtYdzg1MDhmMbItDM2Q2UXZUVVQ1FjNm5YnF1k0rTU5hSE8vZzg1cHY3Vi9GKzA5c2Q1WmYya0R5NXVva0pqULREQ2xQzZFWGZWk0Nk1A1LCJtYWMiOiJhM2M3MTNjYjYjY2NzNmMmMxYTUwNmZiYmY0NjhmZkR0ODZlYmM1ZTVjMTE1YTQ3ODkzZGI1NzhiODY5Mzc4N2JhIiwidGFnIjoieU29hZmpraWlAwSU4yZndwdndEwkJTE1KTQwxa3FfeFXMQxdhc3BONWk1UEhBYUtYdzg1MDhmMbItDM2Q2UXZUVVQ1FjNm5YnF1k0rTU5hSE8vZzg1cHY3Vi9GKzA5c2Q1WmYya0R5NXVva0pqULREQ2xQzZFWGZWk0Nk1A1LCJtYWMiOiJhM2M3MTNjYjYjY2NzNmMmMxYTUwNmZiYmY0NjhmZkR0ODZlYmM1ZTVjMTE1YTQ3ODkzZGI1NzhiODY5Mzc4N2JhIiwidGFnIjoieU29hZmpraWlAwSU4yZndwdndEwkJTE1KTQwxa3FfeFXMQxdhc3BONWk1UEhBYUtYdzg1MDhmMbItDM2Q2UXZUVVQ1FjNm5YnF1k0rTU5hSE8vZzg1cHY3Vi9GKzA5c2Q1WmYya0R5NXVva0pqULREQ2xQzZFWGZWk0Nk1A1LCJtYWMiOiJhM2M3MTNjYjYjY2NzNmMmMxYTUwNmZiYmY0NjhmZkR0ODZlYmM1ZTVjMTE1YTQ3ODkzZGI1NzhiODY5Mzc4N2JhIiwidGFnIjoieU29hZmpraWlAwSU4yZndwdndEwkJTE1KTQwxa3FfeFXMQxdhc3BONWk1UEhBYUtYdzg1MDhmMbItDM2Q2UXZUVVQ1FjNm5YnF1k0rTU5hSE8vZzg1cHY3Vi9GKzA5c2Q1WmYya0R5NXVva0pqULREQ2xQzZFWGZWk0Nk1A1LCJtYWMiOiJhM2M3MTNjYjYjY2NzNmMmMxYTUwNmZiYmY0NjhmZkR0ODZlYmM1ZTVjMTE1YTQ3ODkzZGI1NzhiODY5Mzc4N2JhIiwidGFnIjoieU29hZmpraWlAwSU4yZndwdndEwkJTE1KTQwxa3FfeFXMQxdhc3BONWk1UEhBYUtYdzg1MDhmMbItDM2Q2UXZUVVQ1FjNm5YnF1k0rTU5hSE8vZzg1cHY3Vi9GKzA5c2Q1WmYya0R5NXVva0pqULREQ2xQzZFWGZWk0Nk1A1LCJtYWMiOiJhM2M3MTNjYjYjY2NzNmMmMxYTUwNmZiYmY0NjhmZkR0ODZlYmM1ZTVjMTE1YTQ3ODkzZGI1NzhiODY5Mzc4N2JhIiwidGFnIjoieU29hZmpraWlAwSU4yZndwdndEwkJTE1KTQwxa3FfeFXMQxdhc3BONWk1UEhBYUtYdzg1MDhmMbItDM2Q2UXZUVVQ1FjNm5YnF1k0rTU5hSE8vZzg1cHY3Vi9GKzA5c2Q1WmYya0R5NXVva0pqULREQ2xQzZFWGZWk0Nk1A1LCJtYWMiOiJhM2M3MTNjYjYjY2NzNmMmMxYTUwNmZiYmY0NjhmZkR0ODZlYmM1ZTVjMTE1YTQ3ODkzZGI1NzhiODY5Mzc4N2JhIiwidGFnIjoieU29hZmpraWlAwSU4yZndwdndEwkJTE1KTQwxa3FfeFXMQxdhc3BONWk1UEhBYUtYdzg1MDhmMbItDM2Q2UXZUVVQ1FjNm5YnF1k0rTU5hSE8vZzg1cHY3Vi9GKzA5c2Q1WmYya0R5NXVva0pqULREQ2xQzZFWGZWk0Nk1A1LCJtYWMiOiJhM2M3MTNjYjYjY2NzNmMmMxYTUwNmZiYmY0NjhmZkR0ODZlYmM1ZTVjMTE1YTQ3ODkzZGI1NzhiODY5Mzc4N2JhIiwidGFnIjoieU29hZmpraWlAwSU4yZndwdndEwkJTE1KTQwxa3FfeFXMQxdhc3BONWk1UEhBYUtYdzg1MDhmMbItDM2Q2UXZUVVQ1FjNm5YnF1k0rTU5hSE8vZzg1cHY3Vi9GKzA5c2Q1WmYya0R5NXVva0pqULREQ2xQzZFWGZWk0Nk1A1LCJtYWMiOiJhM2M3MTNjYjYjY2NzNmMmMxYTUwNmZiYmY0NjhmZkR0ODZlYmM1ZTVjMTE1YTQ3ODkzZGI1NzhiODY5Mzc4N2JhIiwidGFnIjoieU29hZmpraWlAwSU4yZndwdndEwkJTE1KTQwxa3FfeFXMQxdhc3BONWk1UEhBYUtYdzg1MDhmMbItDM2Q2UXZUVVQ1FjNm5YnF1k0rTU5hSE8vZzg1cHY3Vi9GKzA5c2Q1WmYya0R5NXVva0pqULREQ2xQzZFWGZWk0Nk1A1LCJtYWMiOiJhM2M3MTNjYjYjY2NzNmMmMxYTUwNmZiYmY0NjhmZkR0ODZlYmM1ZTVjMTE1YTQ3ODkzZGI1NzhiODY5Mzc4N2JhIiwidGFnIjoieU29hZmpraWlAwSU4yZndwdndEwkJTE1KTQwxa3FfeFXMQxdhc3BONWk1UEhBYUtYdzg1MDhmMbItDM2Q2UXZUVVQ1FjNm5YnF1k0rTU5hSE8vZzg1cHY3Vi9GKzA5c2Q1WmYya0R5NXVva0pqULREQ2xQzZFWGZWk0Nk1A1LCJtYWMiOiJhM2M3MTNjYjYjY2NzNmMmMxYTUwNmZiYmY0NjhmZkR0ODZlYmM1ZTVjMTE1YTQ3ODkzZGI1NzhiODY5Mzc4N2JhIiwidGFnIjoieU29hZmpraWlAwSU4yZndwdndEwkJTE1KTQwxa3FfeFXMQxdhc3BONWk1UEhBYUtYdzg1MDhmMbItDM2Q2UXZUVVQ1FjNm5YnF1k0rTU5hSE8vZzg1cHY3Vi9GKzA5c2Q1WmYya0R5NXVva0pqULREQ2xQzZFWGZWk0Nk1A1LCJtYWMiOiJhM2M3MTNjYjYjY2NzNmMmMxYTUwNmZiYmY0NjhmZkR0ODZlYmM1ZTVjMTE1YTQ3ODkzZGI1NzhiODY5Mzc4N2JhIiwidGFnIjoieU29hZmpraWlAwSU4yZndwdndEwkJTE1KTQwxa3FfeFXMQxdhc3BONWk1UEhBYUtYdzg1MDhmMbItDM2Q2UXZUVVQ1FjNm5YnF1k0rTU5hSE8vZzg1cHY3Vi9GKzA5c2Q1WmYya0R5NXVva0pqULREQ2xQzZFWGZWk0Nk1A1LCJtYWMiOiJhM2M3MTNjYjYjY2NzNmMmMxYTUwNmZiYmY0NjhmZkR0ODZlYmM1ZTVjMTE1YTQ3ODkzZGI1NzhiODY5Mzc4N2JhIiwidGFnIjoieU29hZmpraWlAwSU4yZndwdndEwkJTE1KTQwxa3FfeFXMQxdhc3BONWk1UEhBYUtYdzg1MDhmMbItDM2Q2UXZUVVQ1FjNm5YnF1k0rTU5hSE8vZzg1cHY3Vi9GKzA5c2Q1WmYya0R5NXVva0pqULREQ2xQzZFWGZWk0Nk1A1LCJtYWMiOiJhM2M3MTNjYjYjY2NzNmMmMxYTUwNmZiYmY0NjhmZkR0ODZlYmM1ZTVjMTE1YTQ3ODkzZGI1NzhiODY5Mzc4N2JhIiwidGFnIjoieU29hZmpraWlAwSU4yZndwdndEwkJTE1KTQwxa3FfeFXMQxdhc3BONWk1UEhBYUtYdzg1MDhmMbItDM2Q2UXZUVVQ1FjNm5YnF1k0rTU5hSE8vZzg1cHY3Vi9GKzA5c2Q1WmYya0R5NXVva0pqULREQ2xQzZFWGZWk0Nk1A1LCJtYWMiOiJhM2M3MTNjYjYjY2NzNmMmMxYTUwNmZiYmY0NjhmZkR0ODZlYmM1Z

Change the responses cookies with the session cookies stolen from another user



You can see after forwarding all requests, we have logged in to the admin user:



Remediation

Following are the different ways to prevent a session hijacking:

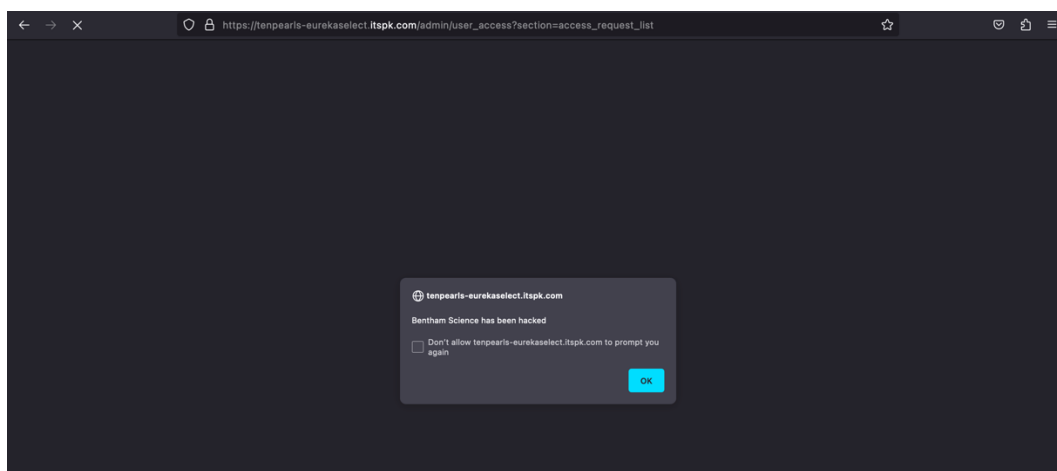
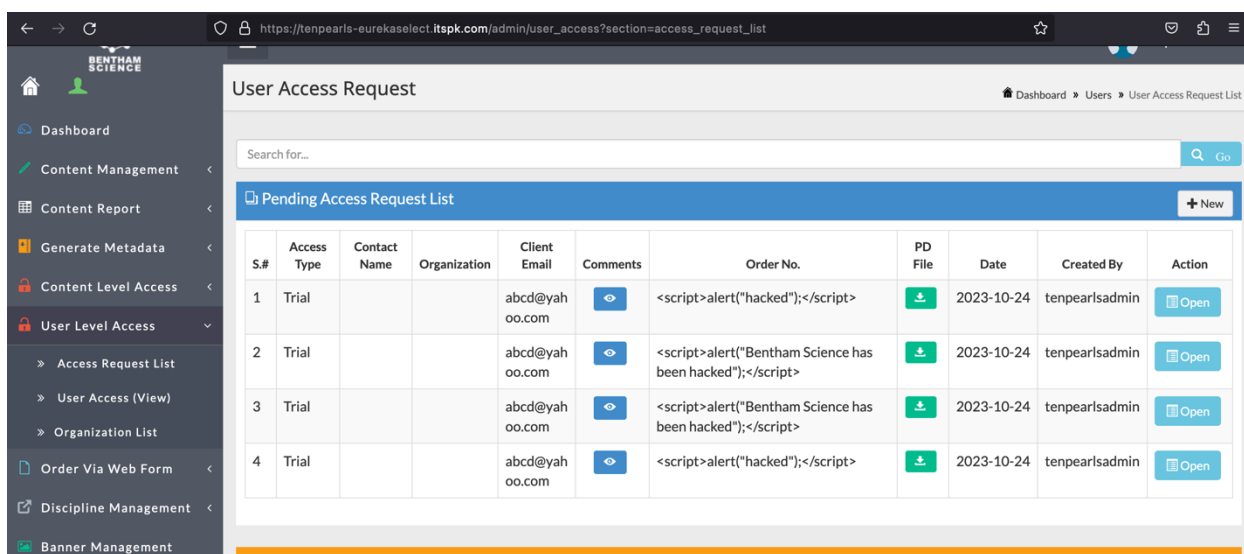
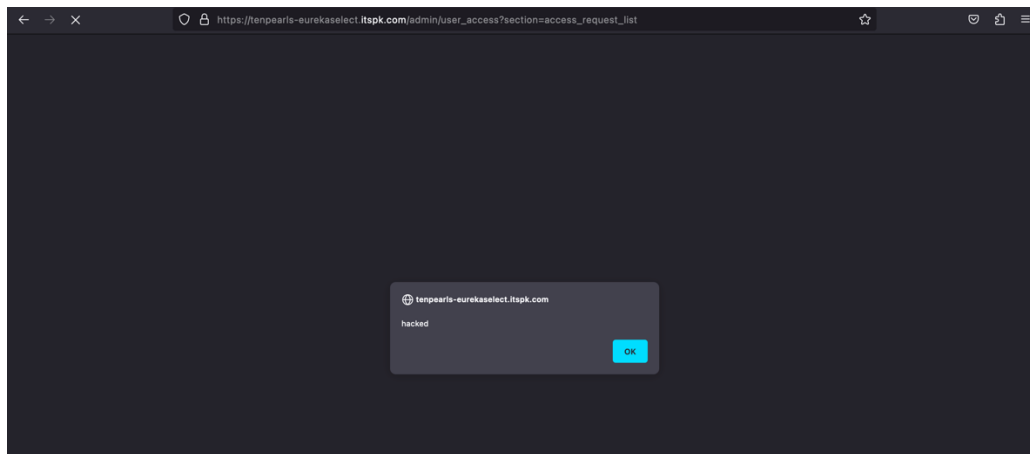
- ☐ Make sure that all session identifiers are transmitted over an encrypted protocol.
- ☐ Terminate/regenerate the session if the session token is transmitted insecurely (either in clear text or as part of the URL), or signal to the application to do so.
- ☐ Ensure that session identifiers are transmitted only using the SSL session where they originated. Track sessions across SSL renegotiations and integrate with framework solutions to support common SSL termination/encryption architectures.

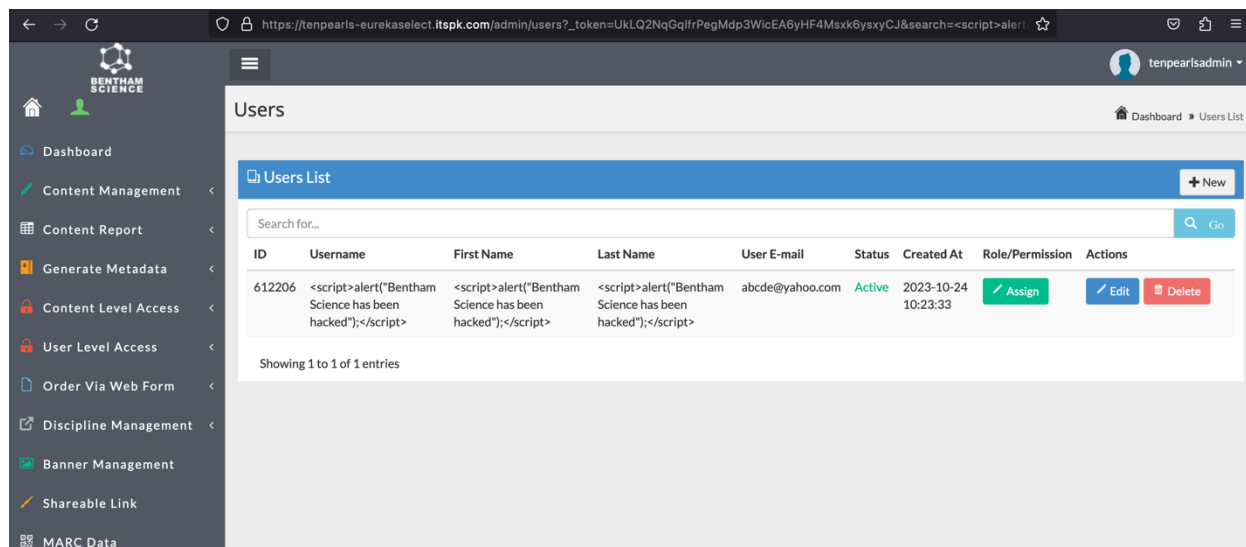
CROSS SITE SCRIPTING (STORED XSS)

Risk Level	Critical
Finding	Application is vulnerable to a stored cross site scripting attack.
Explanation	Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application. It allows an attacker to circumvent the same origin policy, which is designed to segregate different websites from each other. Cross-site scripting vulnerabilities normally allow an attacker to masquerade as a victim user, to carry out any actions that the user is able to perform, and to access any of the user's data. If the victim user has privileged access within the application, then the attacker might be able to gain full control over all of the application's functionality and data. Stored XSS generally occurs when user input is stored on the target server, such as in a database, in a message forum, visitor log, comment field, etc. And then a victim is able to retrieve the stored data from the web application without that data being made safe to render in the browser.
References	https://owasp.org/www-community/attacks/xss/

Proof of Concept

Following are the screenshots of the vulnerability:





Remediation

This issue occurs because the browser interprets the input as active HTML, JavaScript or VBScript. To avoid this, all input and output from the application should be filtered / encoded. Output should be filtered / encoded according to the output format and location.

There are a number of pre-defined, well structured whitelist libraries available for many different environments. Good examples of these include OWASP Reform and Microsoft Anti-Cross-site Scripting libraries.

Additionally, you should implement a strong Content Security Policy (CSP) as a defense-in-depth measure if an XSS vulnerability is mistakenly introduced. Due to the complexity of XSS-Prevention and the lack of secure standard behavior in programming languages and frameworks, XSS vulnerabilities are still common in web applications.

CSP will act as a safeguard that can prevent an attacker from successfully exploiting Cross-site Scripting vulnerabilities in your website and is advised in any kind of application. Please make sure to scan your application again with Content Security Policy checks enabled after implementing CSP, in order to avoid common mistakes that can impact the effectiveness of your policy. There are a few pitfalls that can render your CSP policy useless and we highly recommend reading the resources linked in the reference section before you start to implement one.

BRUTE FORCE ATTACK

Risk Level	High
Finding	Application is vulnerable to brute force attack
Explanation	Brute-force attacks are often used for attacking authentication and discovering hidden content/pages within a web application. These attacks are usually sent via GET and POST requests to the server. In regards to authentication, brute force attacks are often mounted when an account lockout policy is not in place.
References	https://owasp.org/www-community/attacks/Brute_force_attack

Proof of concept

Following evidence showed us that the website was vulnerable to brute force attack, when we uploaded a file payload of thousands of random passwords (automatically generated), and the tool provided results with the correct password redirecting to a new page

The screenshot shows the Burp Suite interface during a brute force attack. The 'Intruder' tab is active, displaying a list of payloads and their corresponding status codes. The 'Results' tab shows a table with columns: Request, Payload, Status code, Error, Timeout, Length, and Comment. The table lists 8 requests, all with a status code of 302. The 'Response' tab shows the HTTP response for the selected request, including headers like Host, Cookie, and Content-Type.

Request	Payload	Status code	Error	Timeout	Length	Comment
0		302			1845	
1	abcd123	302			1845	
2	asasdasd	302			1845	
3	xyz222	302			1845	
4	jhdjdjda	302			1845	
5	jnuhrhbf	302			1845	
6	random	302			1845	
7	tenpearls	302			1845	
8	abcd12345	302			1845	

Remediation

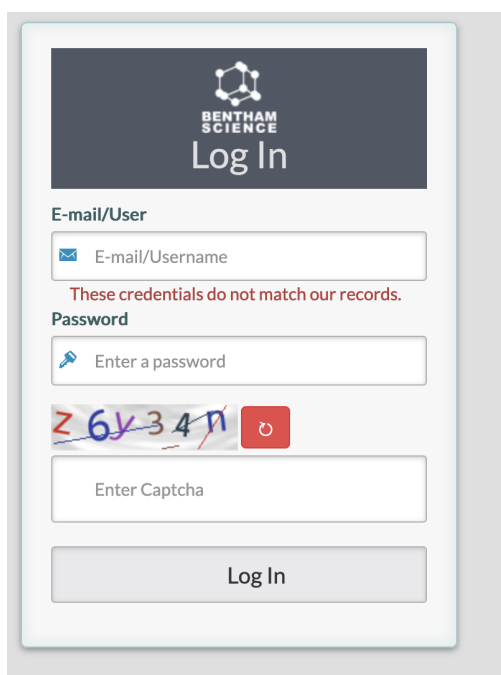
Below are some proven ways for brute force attack prevention:

- ☐ Use strong passwords
- ☐ Limit login attempts
- ☐ Use multifactor authentication
- ☐ Use unique login URLs

ACCOUNT LOCKOUT POLICY IS NOT ENABLED

Risk Level	High
Finding	Account lockout policy is not implemented
Explanation	Upon entering multiple times wrong password, the account of that particular user has not been locked out which leads to the successful attempt of account breach of any user.
References	https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/04-Authentication_Testing/03-Testing_for_Weak_Lock_Out_Mechanism

Proof of concept



Remediation

After one or two failed login attempts, you may want to prompt the user not only for the username and password but also to answer a secret question. This not only causes problems with automated attacks, it prevents an attacker from gaining access, even if they do get the username and password correct.

You could also detect high numbers of attacks system-wide and under those conditions prompt all users for the answer to their secret questions.

For advanced users who want to protect their accounts from attack, give them the option to allow login only from certain IP addresses.

Assign unique login URLs to blocks of users so that not all users can access the site from the same URL.

Use a captcha to prevent automated attacks.

HTTP STRICT TRANSPORT SECURITY (HSTS) ERRORS AND WARNINGS

Risk Level	Medium
Finding	HSTS shows errors and warnings. The preload directive is not set.
Explanation	HTTP Strict Transport Security (HSTS) is a policy mechanism that helps to protect websites against man-in-the-middle attacks such as protocol downgrade attacks and cookie hijacking. It allows web servers to declare that web browsers (or other complying user agents) should automatically interact with it using only HTTPS connections, which provide Transport Layer Security (TLS/SSL), unlike the insecure HTTP used alone. We identified errors during parsing of Strict-Transport-Security header. The HSTS Warning and Error may allow attackers to bypass HSTS, effectively allowing them to read and modify your communication with the website.
References	-

Proof of Concept

```

HTTP/1.1 200 OK
Set-Cookie: XSRF-TOKEN=eyJpdiI6IjNsQzBaTjJpMzNhYmg3ZXRMQ2lNQ1E9PSIsInZhbnVlIjoicDJWZmJqWn15dDFpd0VWQzZuVksuQWVXaw90V0YzNEg2TU
Set-Cookie: eurekaselectcom_session=eyJpdiI6InJ3NFUwWkkrUmUwUDhycUljlMlJIM1E9PSIsInZhbnVlIjoiodDZtZ08ybmlNUTGNENFJEWjMzVEw0angrY
Access-Control-Allow-Headers: Content-Type, X-CSRF-TOKEN
Server: Apache
Transfer-Encoding: chunked
Vary: Accept-Encoding
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=31536000
Content-Type: text/html; charset=UTF-8
Content-Encoding:

```

Remediation

Ideally, after fixing the errors and warnings, you should consider adding your domain to the HSTS preload list. This will ensure that browsers automatically connect your website by using HTTPS, actively preventing users from visiting your site using HTTP. Since this list is hardcoded in users' browsers, it will enable HSTS even before they visit your page for the first time, eliminating the need for Trust on First Use (TOFU) with its associated risks and disadvantages. Unless you fix the errors and warnings your website won't meet the conditions required to enter the browser's preload list.

Browser vendors declared:

- ☐ Serve a valid certificate
- ☐ If you are listening on port 80, redirect all domains from HTTP to HTTPS on the same host. Serve all subdomains over HTTPS:
 - In particular, you must support HTTPS for the www subdomain if a DNS record for that subdomain exists
- ☐ Serve an HSTS header on the base domain for HTTPS requests:
 - The max-age must be at least 31536000 seconds (1 year)
 - The includeSubDomains directive must be specified
 - The preload directive must be specified
 - If you are serving an additional redirect from your HTTPS site, that redirect must have the HSTS header (rather than the page it redirects to).

CLICK JACKING

Risk Level	Medium
Finding	Application is vulnerable to a click jacking attack.
Explanation	Clickjacking refers to any attack where the user is tricked into unintentionally clicking an unexpected web page element. The name was coined from click hijacking, and the technique is most often applied to web pages by overlaying malicious content over a trusted page or by placing a transparent page on top of a visible one. When the user clicks an innocent-looking item on the visible page, they are actually clicking the corresponding location on the overlaid page and the click triggers a malicious action – anything from faking a like or follow-on social media
References	https://owasp.org/www-community/attacks/Clickjacking

Proof of Concept

irls-eurekaselect.itspk.com/admin/dashboard

jr Mobile Applic...  Android Bug Boun...  Android SSL pinning  ios 12.5.4 jailbreak  Secure Java Progr...  Mobexler - Mobile...  AES Ki

- + Toggle transparency Reset Save

Click

irls-eurekaselect.itspk.com/admin/dashboard

jr Mobile Applic...  Android Bug Boun...  Android SSL pinning  ios 12.5.4 jailbreak  Secure Java Progr...  Mobexler - Mobile...  AES Ki

- + Toggle transparency Reset Save

You've been Hacked!

Remediation

X-Frame-Options: Probably the best solution at present is to use the X-Frame-Options (XFO) HTTP response header in server responses. Originally introduced by Microsoft in Internet Explorer 8 and later formalized in RFC 7034, the X-Frame-Options header is used to specify whether a page can be embedded in a <frame>, <iframe>, <embed> or <object> element. The header supports three possible directives: deny to block all framing attempts, sameorigin to allow framing only by pages of the same origin, or allow-from to allow framing by pages from specified

URIs. However, allow-from is unsupported by several browsers (including Chrome and Safari), so if you need to specify sources, you are better off using CSP (see below). For general anti-framing protection, you only need to specify X-Frame-Options: deny or X-Frame-Options: sameorigin in your server's headers.

Content-Security-Policy with frame-ancestors: The Content-Security-Policy HTTP header (CSP) was initially developed to protect against XSS and other data injection attacks. However, it also provides a frame-ancestors directive for specifying sources that are permitted to embed a page (in a <frame>, <iframe>, <object>, <embed>, or <applet> element). The syntax is simple:

Content-Security-Policy: frame-ancestors <source1> <source2> ... <sourceN>;

CONTENT SECURITY POLICY (CSP) NOT IMPLEMENTED

Risk Level	Medium
Finding	CSP policy not implemented on application.
Explanation	If Content Security Policy (CSP) is not implemented in a web application, it means that the application is lacking an important security mechanism that helps mitigate various types of attacks, such as cross-site scripting (XSS) and data injection attacks. Content Security Policy is a security feature supported by modern web browsers. It allows website administrators to define a set of policies that control which resources (e.g., scripts, stylesheets, fonts, etc.) the browser should load and execute. By specifying a Content Security Policy, administrators can restrict the sources from which a webpage can load content, thus reducing the risk of malicious code execution.
References	-

Proof of Concept

Following is the screenshot of this vulnerability:

Enable CSP on your website by sending the Content-Security-Policy in HTTP response headers that instruct the browser to apply the policies you specified.

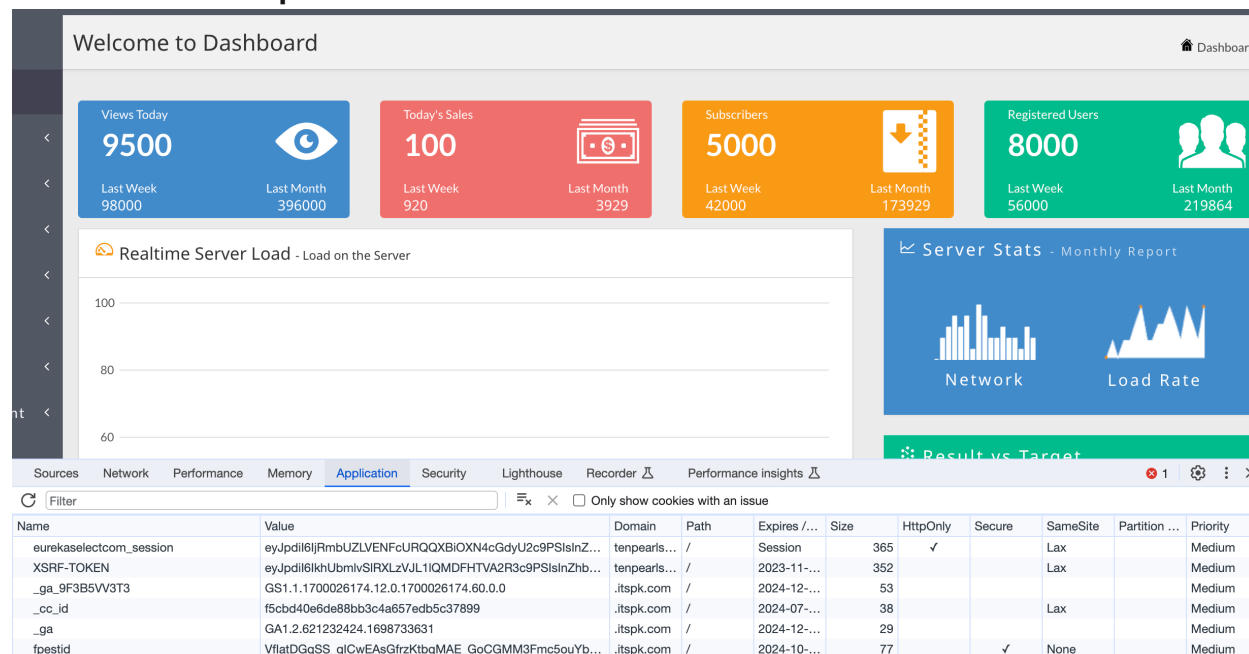
Risk Level	Medium
Finding	Session cookie httponly and secure flag is not set.
Explanation	It was identified a cookie not marked as HTTPOnly & secure, and transmitted over HTTPS. If Secure flag is not set: This means the cookie could potentially be stolen by an attacker who can successfully intercept and decrypt the traffic, or following a successful man-in-the-middle attack. This cookie will be transmitted over a HTTP connection, therefore if this cookie is important (such as a session cookie), an attacker might intercept it and hijack a victim's session. If the attacker can carry out a man-in-the-middle attack, he/she can force the victim to make an HTTP request to steal the cookie. If HTTPOnly flag is not set: HTTPOnly cookies cannot be read by client-side scripts, therefore marking a cookie as HTTPOnly can provide an additional layer of protection against cross-site scripting attacks.

References

During a cross-site scripting attack, an attacker might easily access cookies and hijack the victim's session.

<https://portswigger.net/web-security/information-disclosure>

Proof of Concept



Remediation

Mark all session cookies as HTTPOnly and secure.

OUT-OF-DATE VERSION (JQUERY)

Risk Level	Medium
Finding	Vulnerable version of jquery found.
Explanation	We identified the target web application is using jquery and detected that it is out of date. Since this is an old version of the software, it may be vulnerable to attacks. This version is vulnerable to multiple known and public attacks.
References	-

Proof of Concept

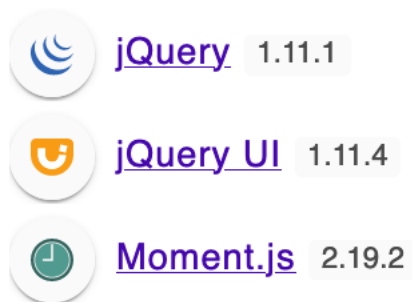
```

/*! jQuery v1.11.1 | (c) 2005, 2014 jQuery Foundation, Inc. | jquery.org/license */
!function(a,b){{"object"==typeof module&&"object"==typeof module.exports?module.exports=a.document?b(a,!0
le(l=l[p])if(h?l.nodeName.toLowerCase()===r:1===l.nodeType)return!1;o=p="only"===a&&o&&"nextSibling"}re
ddEventListener?(y.removeEventListener("DOMContentLoaded",J,!1),a.removeEventListener("load",J,!1)):(y.d

if(k&&j[k]&&(e||j[k].data)||void 0!==d||"string"!==typeof b)return k||(k=i?a[h]=c.pop()||m.guid++:h),j[k]
d,e;if(a&&a.preventDefault&&a.handleObj)return d=a.handleObj,m(a.delegateTarget).off(d.namespace?d.origT
]=Ub(a.style,h)),g=m.cssHooks[b]||m.cssHooks[h],g&&"get"in g&&(f=g.get(a,!0,c)),void 0===f&&(f=Jb(a,b,d)

```

JavaScript libraries



Remediation

Please upgrade to the latest stable version.

MISSING X-XSS-PROTECTION HEADER

Risk Level	Low
Finding	The application is missing x-xss protection header.
Explanation	The X-XSS-Protection header is designed to enable the cross-site scripting (XSS) filter built into modern web browsers. This is usually enabled by default, but using it will enforce it. It is supported by Internet Explorer 8+, Chrome, Edge, Opera, and Safari. The recommended configuration is to set this header to the following value, which will enable the XSS protection and instruct the browser to block the response in the event that a malicious script has been inserted from user input instead of sanitizing.

	It was detected a missing X-XSS-Protectionheader which means that this website could be at risk of a Cross-site Scripting (XSS) attacks.
References	https://www.keycdn.com/blog/x-xss-protection

Proof of Concept

Following is the screenshot of this vulnerability:

```
HTTP/1.1 200 OK
Date: Wed, 25 Oct 2023 10:05:03 GMT
Server: Apache
Last-Modified: Fri, 01 Sep 2023 07:26:31 GMT
Accept-Ranges: bytes
Vary: Accept-Encoding
Strict-Transport-Security: max-age=31536000
Access-Control-Allow-Headers: Content-Type, X-CSRF-TOKEN
X-Frame-Options: SAMEORIGIN
Content-Length: 110
Connection: close
Content-Type: text/plain

User-agent: *
Allow: /
Allow: /$
#Disallow: /

#User-agent: * Disallow: /
Crawl-delay: 10
# Paths (clean URLs)
```

Remediation

Add the X-XSS-Protection header with a value of "1; mode= block".

X-XSS-Protection: 1; mode=block

ROBOTS.TXT FILE DETECTED

Risk Level	Informational
Finding	The web server contains a robots.txt file.
Explanation	The file robots.txt is used to give instructions to web robots, such as search engine crawlers, about locations within the web site that robots are allowed, or not allowed, to crawl and index. The presence of the robots.txt does not in itself present any kind of security vulnerability. However, it is often used to identify restricted or private areas of a site's contents. The information in the file may therefore help an attacker to map out the site's contents, especially if some of the locations identified are not linked from elsewhere in the site. If the application relies on robots.txt to protect

	access to these areas, and does not enforce proper access control over them, then this presents a serious vulnerability.
References	https://cwe.mitre.org/data/definitions/200.html

Proof of Concept

```

← → ↻ https://tenpearls-eurekaselect.itspk.com/robots.txt

User-agent: *
Allow: /
Allow: /$
#Disallow: /

#User-agent: * Disallow: /
Crawl-delay: 10
# Paths (clean URLs)

```

Remediation

The robots.txt file is not itself a security threat, and its correct use can represent good practice for non-security reasons. You should not assume that all web robots will honor the file's instructions. Rather, assume that attackers will pay close attention to any locations identified in the file. Do not rely on robots.txt to provide any kind of protection over unauthorized access.

AUTOCOMPLETE IS ENABLED

Risk Level	Informational
Finding	Autocomplete is Enabled in one or more of the form fields which might contain sensitive information like "username" and "old password"
Explanation	In typical form-based web applications, it is common practice for developers to allow `autocomplete` within the HTML form to improve the usability of the page. With `autocomplete` enabled (default), the browser is allowed to cache previously entered form values.

References

https://owasp.org/www-community/OWASP_Application_Security_FAQ

Proof of Concept

```
<div id="login" class="animate form">
<form action="https://tenpearls-eurekaselect.itspk.com/login" autocomplete="on" method="post" role="form" id="login_form">
<input type="hidden" name="_token" value="hjkZRUqudQsJEXurz4JZDdPrjSM6k3F7QneBa5A7">
...[SNIP]...
</label>
<input type="password" id="password" name="password" placeholder="Enter a password" />
</div class="col-sm-12">
```

Remediation

The `autocomplete` value can be configured in two different locations.

The first and most secure location is to disable the `autocomplete` attribute on the `` HTML tag. This will disable `autocomplete` for all inputs within that form. An example of disabling `autocomplete` within the form tag is ``.

The second slightly less desirable option is to disable the `autocomplete` attribute for a specific `` HTML tag. While this may be the less desired solution from a security perspective, it may be preferred method for usability reasons, depending on size of the form. An example of disabling the `autocomplete` attribute within a password input tag is `