UnderDefense
CyberSecurity solutions protecting your business

Web and Infrastructure Penetration Testing Report for [CLIENT]

Executive summary

This report presents the results of the "Grey Box" penetration testing for [CLIENT] Infrastructure and Web Application. The recommendations provided in this report structured to facilitate remediation of the identified security risks. This document serves as a formal letter of attestation for the recent [CLIENT] Infrastructure and Web Application Penetration Testing.
Evaluation ratings compare information gathered during the course of the engagement to "best in class" criteria for security standards. We believe that the statements made in this document provide an accurate assessment of [CLIENT] current security as it relates to infrastructure and network perimeter..
We highly recommend to review section of Summary of business risks and High-Level Recommendations for better understanding of risks and discovered security issues.

Scope	Security level	Grade
Web application and infrastructure perimeter	D	Poor

UnderDefense Grading Criteria:

Grade	Security	Criteria Description
A	Excellent	The security exceeds "Industry Best Practice" standards. The overall posture was found to be excellent with only a few low-risk findings identified.
B	Good	The security meets with accepted standards for "Industry Best Practice." The overall posture was found to be strong with only a handful of medium- and low- risk shortcomings identified.
C	Fair	Current solutions protect some areas of the enterprise from security issues. Moderate changes are required to elevate the discussed areas to "Industry Best Practice" standards
D	Poor	Significant security deficiencies exist. Immediate attention should be given to the discussed issues to address exposures identified. Major changes are required to elevate to "Industry Best Practice" standards.

F	Inadequate	Serious security deficiencies exist. Shortcomings were identified throughout most or even all of the security controls examined. Improving security will require a major allocation of resources.

Assumptions & Constraints

As the environment changes, and new vulnerabilities and risks are discovered and made public, an organization's overall security posture will change. Such changes may affect the validity of this letter. Therefore, the conclusion reached from our analysis only represents a "snapshot" in time.

Objectives & Scope

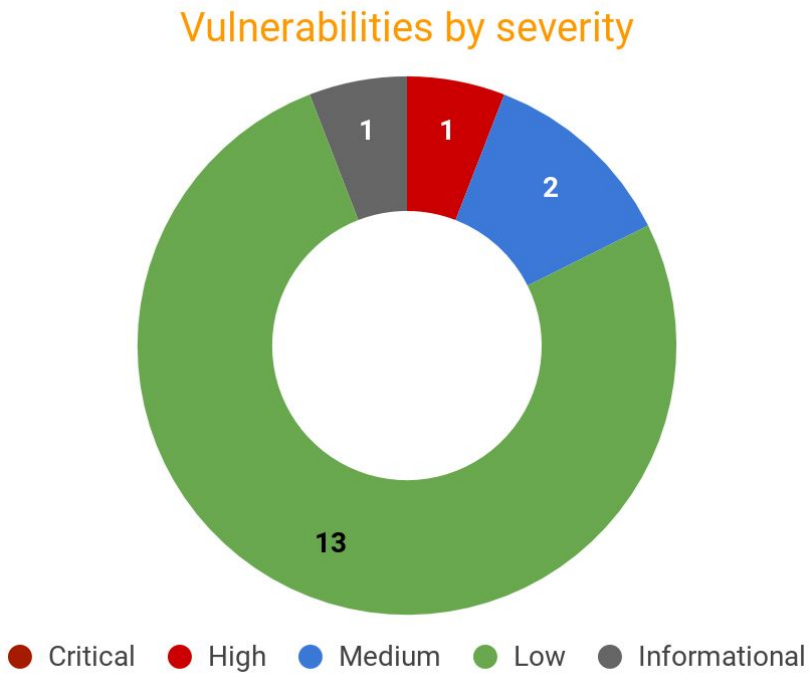
Organization	[CLIENT ORGANISATION]
Audit type	Grey Box Infrastructure and Web Application Penetration Testing
Asset URL	APPENDIX A - Scope
Audit period	Feb. 19 - Mar. 11

Consultants performed discovery process to gather information about the target and searched for information disclosure vulnerabilities. With this data in hand, we conducted the bulk of the testing manually, which consisted of input validation tests, impersonation (authentication and authorization) tests, and session state management tests. The purpose of this penetration testing is to illuminate security risks by leveraging weaknesses within the environment that lead to the obtainment of unauthorized access and/or the retrieval of sensitive information. The shortcomings identified during the assessment were used to formulate recommendations and mitigation strategies for improving the overall security posture.

Results Overview

The test uncovered several vulnerabilities that may cause users credentials stealing, session hijacking, sensitive data leakage, broken confidentiality, integrity and availability of the resource. Security testing activities showed that there are a lot of misconfigurations on cloud infrastructure.

Identified vulnerabilities are easily exploitable and the risk posed by these vulnerabilities can cause significant damage to the company.



Security experts performed manual security testing, which demonstrate the following results.

Severity	Critical	High	Medium	Low	Informational
# of issues	0	1	2	13	1

Severity scoring:

- **Critical** - Immediate threat to key business processes.
- **High** – Direct threat to key business processes.
- **Medium** – Indirect threat to key business processes or partial threat to business processes.
- **Low** – No direct threat exists. Vulnerability may be exploited using other vulnerabilities.
- **Informational** – This finding does not indicate vulnerability, but states a comment that notifies about design flaws and improper implementation that might cause a problem in the long run.

Summary of business risks

High severity issues make direct threat to the business as they can be used to:

- Using XSS attack it is possible to steal user session or credentials and get full access to users account. It will lead to client data leakage as this vulnerability is present on application which is used by company clients. Successful exploitation of this vulnerability will cause big reputational and financial damage for the company.

Medium and low severity issues can lead to:

- Attacks on communication channels and as a result on sensitive data leakage and possible modification, in other words it affects the integrity and confidentiality of data transferred.
- Information leakage about system components which may be used by attackers for further malicious actions.
- Attacks on old and not patched system components with bunch of publicly known vulnerabilities.
- Enumerating existing users emails/usernames and brute forcing their passwords. Easy access to their session after exploitation of high level risks, as session token is not invalidated after logout.
- Combination of few issues can be used for successful realisation of attacks.

Informational severity issues do not carry direct threat but they can be used to gather useful information for an attacker.

High-Level Recommendations

Taking into consideration all issues that have been discovered, we highly recommend to:

- Conduct current vs. future IT/Security program review;
- Establish Secure SDLC best practices, assign Security Engineer to a project to monthly review code, conduct SAST & DAST security testing;
- Review Architecture of application;
- Deploy Web Application Firewall solution to detect any malicious manipulations;
- Continuously monitor logs for anomalies to detect abnormal behaviour and fraud transactions. Dedicate security operations engineer to this task;
- Implement Patch Management procedures for whole IT infrastructure and endpoints of employees and developers;
- Continuously Patch production and development environments and systems on regular bases with latest releases and security updates;
- Conduct annual Penetration test and quarterly Vulnerability Scanning against internal and external environment;
- Conduct security coding training for Developers;
- Develop and Conduct Security Awareness training for employees and developers;
- Develop Incident Response Plan in case of Data breach or security incidents;
- Analyse risks for key assets and resources;

- Update codebase to conduct verification and sanitization of user input on both client and server side;
- Use only encrypted channels for communications;
- Improve server and application configuration to meet security best practises;
- Also we recommend to conduct remediation testing of both infrastructure and web applications.

Performed tests

- All set of applicable OWASP Top 10 Security Threats
- All set of applicable SANS 25 Security Threats

Criteria Label	Status
A1:2017-Injection	Meets criteria
A2:2017-Broken Authentication	Fails criteria
A3:2017-Sensitive Data Exposure	Meets criteria
A4:2017-XML External Entities (XXE)	Meets criteria
A5:2017-Broken Access Control	Meets criteria
A6:2017-Security Misconfiguration	Fails criteria
A7:2017-Cross-Site Scripting (XSS)	Fails criteria
A8:2017-Insecure Deserialization	Meets criteria
A9:2017-Using Components with Known Vulnerabilities	Fails criteria
A10:2017-Insufficient Logging&Monitoring	N/A

Performed Tests	Status
Host and service enumeration	Fails criteria
Weak passwords attack and brute-force	Fails criteria
Identification of misconfigurations	Fails criteria
Vulnerability identification and system exploitation	Meets criteria
Search Engine Discovery and Reconnaissance for Information Leakage	Fails criteria
Weak Authorization Mechanisms testing	Meets criteria
Database compromising, sensitive information stealing	Meets criteria
Outdated services	Fails criteria
S3 bucket enumeration	Fails criteria

Security tools used

- Burp Suite Pro [Commercial Edition]
- Nmap
- TestSSL
- Harvester
- Sublister
- ScoutSuite
- CS Suite
- weirdAAL

Methodology

Our Penetration Testing Methodology grounded on following guides and standards:

- Penetration Testing Execution Standard
- OWASP Top 10 Application Security Risks – 2017
- OWASP Testing Guide
- SANS: Conducting a Penetration Test on an Organization
- The Open Source Security Testing Methodology

Open Web Application Security Project (OWASP) is an industry initiative for web application security. OWASP has identified the 10 most common attacks that succeed against web applications. These comprise the OWASP Top 10.

Application penetration test includes all the items in the OWASP Top 10 and more. The penetration tester remotely tries to compromise the OWASP Top 10 flaws. The flaws listed by OWASP in its most recent Top 10 and the status of the application against those are depicted in the table below.

Web Application Findings Details

Stored Cross Site Scripting (XSS) in multiple pages

SEVERITY: **High**

LOCATION:

- https://app.example.co/settings/templates/mailbox/all/template/1
- https://app.example.co/mail/all/drafts/conversation/1?assignedTo=all&tag=all
- https://app.example.co/settings/preferences
- https://api.example.co/api/v1/note

ISSUE DESCRIPTION:

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.

PROOF OF VULNERABILITY:

The attacker can inject javascript code in template and when somebody opens template javascript will be executed.

https://app.example.co/settings/templates/mailbox/all/template/1

```
POST /api/v1/template HTTP/1.1
Host: api.example.co
…..
{"title":"<.ASDfasDF>ASD>FASDf","body":"<div><h1><script>alert(1)</script><h1/></div>","
mailboxUuid":null}
```

The attacker can inject javascript code in drafts and when somebody opens draft javascript will be executed.

https://app.example.co/mail/all/drafts/conversation/1?assignedTo=all&tag=all

```
PUT /api/v1/mail-meta/1 HTTP/1.1
Host: api.example.co
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://app.example.co/
Content-Type: application/json
X-Requested-With: XMLHttpRequest
Content-Length: 293
Origin: https://app.example.co
Connection: close
Cookie: xxx

{"draftUuid":"1","draftCursorPosition":"[{\"start\":[1,0],\"end\":null,\"startOffset\":2
,\"endOffset\":2,\"collapsed\":true,\"is2\":true}]","to":[],"cc":[],"bcc":[],"subject":"
","messageType":"SEND","body":"<div><img src=x onerror=alert('XSS');>>a</div>"}
```

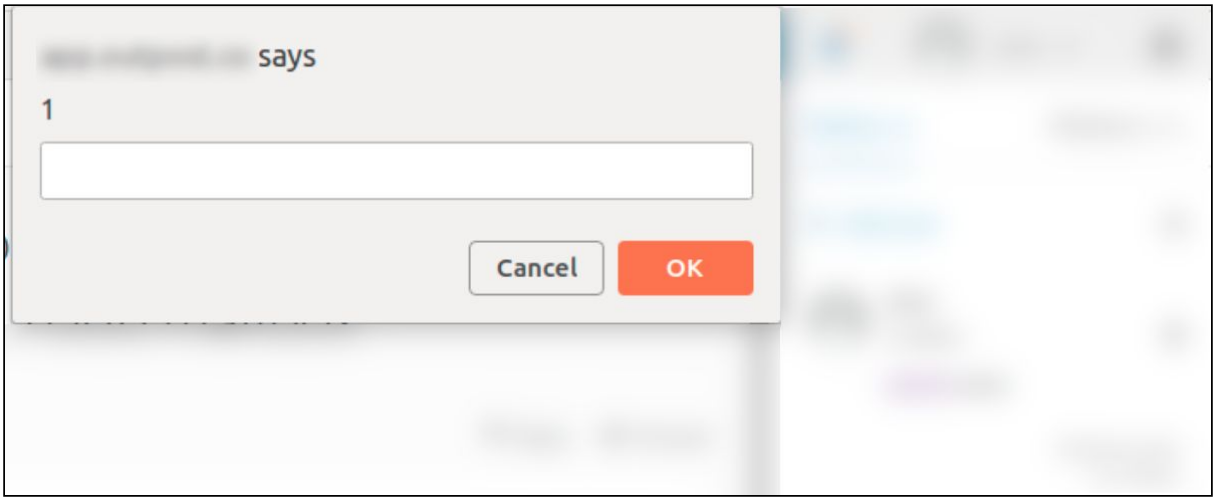
The attacker can inject javascript code in email signature and when somebody opens notes javascript will be executed.

```
PUT /api/v1/user/preferences/1 HTTP/1.1
Host: api.example.co
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://app.example.co/
Content-Type: application/json
X-Requested-With: XMLHttpRequest
Content-Length: 366
Origin: https://app.example.co
Connection: close
Cookie: xxx

{"firstName":"Test'","lastName":"Test","email":"test@example.com","role":"USER","mailbox
Uuids":["xxx","yyy"],"defaultMailboxUuid":"","showMessageSaved":true,"showMessageSent":t
rue,"signature":"<div><svg
onload=alert(1)>a</div>","timezone":"Europe/Kiev","defaultSendAndResolve":true}
```

The attacker can inject javascript code in notes and when somebody opens notes javascript will be executed.

```
PUT /api/v1/note/1 HTTP/1.1
Host: api.example.co
Connection: close
Content-Length: 244
Origin: https://app.example.co
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Ubuntu Chromium/71.0.3578.98 Chrome/71.0.3578.98 Safari/537.36
Content-Type: application/json
Accept: */*
Referer: https://app.example.co/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

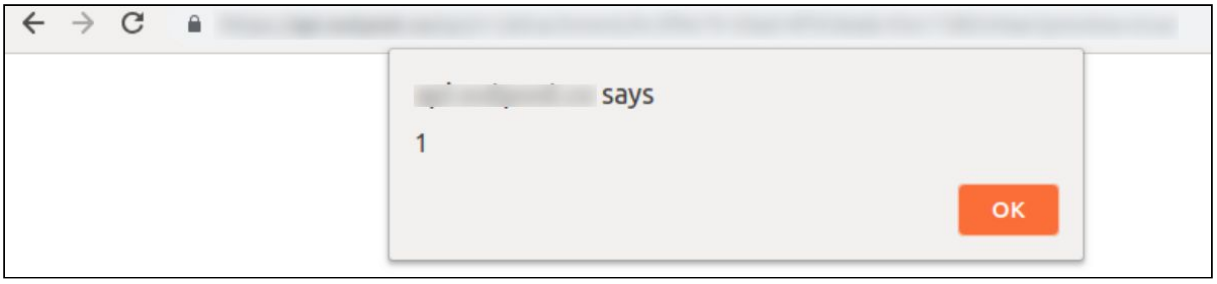
```
Cookie: xxx

{"body":"<div><span contenteditable=\"false\" data-user-uuid=\"1\" class=\"mention
tokenized\">@John</span>d<img src=\"#\"
onerror=alert(1)>ddd</div>","mentionUuids":["1"]}
```



We were able to send email messages containing XSS payload. If users mail account is registered on **teamexample.co** domain it will be not sanitized on backend.

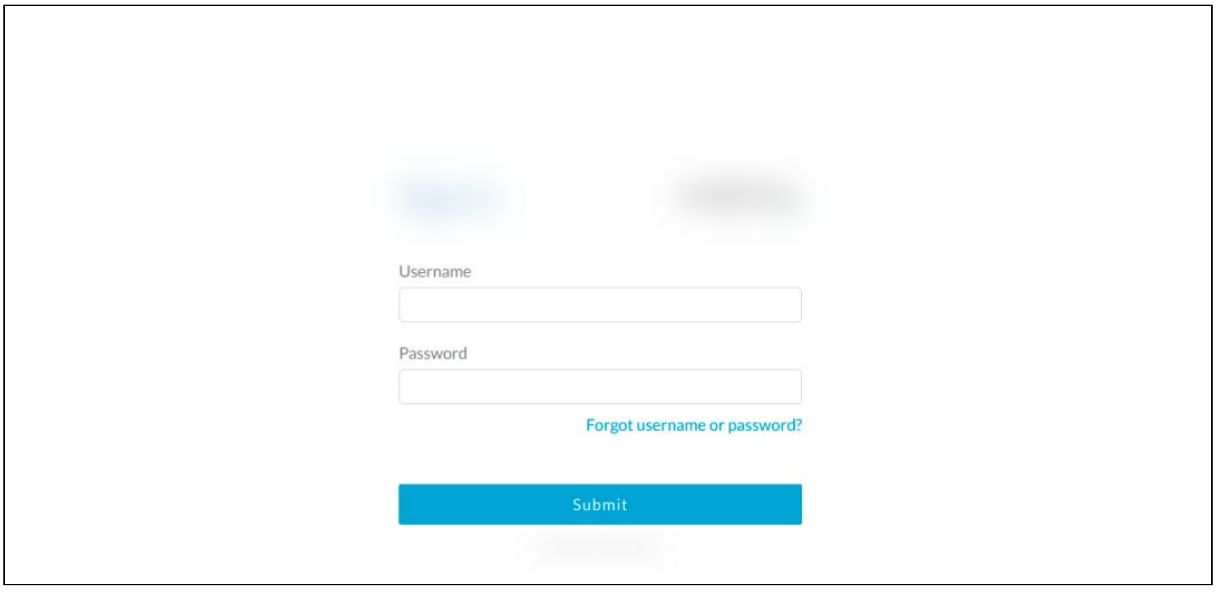
```
POST /api/v1/send-mail?resolve=false HTTP/1.1
Host: api.example.co
Connection: close
Content-Length: 635
Origin: https://app.example.co
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Ubuntu Chromium/71.0.3578.98 Chrome/71.0.3578.98 Safari/537.36
Content-Type: application/json
Accept: */*
Referer: https://app.example.co/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: xxx

{"draftUuid":"1","draftCursorPosition":"[{\"start\":[1,0,0],\"end\":null,\"startOffset\"
:3,\"endOffset\":3,\"collapsed\":true,\"is2\":true}]","to":[{"name":"","email":"test@tes
t.teamexample.com"}],"cc":[],"bcc":[],"subject":"TEST3","messageType":"SEND","body":"<di
v><div><a target=\"_blank\" href=\"http://kflk\">New one 2</a><br><img src=\"#\"
onerror=alert(\"XSS\") /> </div></div><div> </div><div id=\"op_sig_block\"><div
id=\"op_user_sig_1\"><div>John</div></div><div id=\"op_mb_sig_1\"> </div></div>"}
```

We can upload file with malicious code and when you open preview link the javascript code will be executed.



Example of XSS exploitation where attacker can send email with malicious code and create fake login page. An inexperienced user could not understand what just happened and will try to login again. Request which contains users credentials will be sent to the malicious actor. As a result username and password will be leaked.

Example of fake malicious login page:



RECOMMENDATIONS:

Preventing XSS requires separation of untrusted data from active browser content. This can be achieved by:

- Using frameworks that automatically escape XSS by design, such as the latest Ruby on Rails, React JS. Learn the limitations of each framework's XSS protection and appropriately handle the use cases which are not covered.
- Escaping untrusted HTTP request data based on the context in the HTML output (body, attribute, JavaScript, CSS, or URL) will resolve Reflected and Stored XSS vulnerabilities.

To filter user input sufficiently, consider XSS Prevention Cheat Sheet.

It is also needed to provide users input validation and sanitization on backend part.

Session Fixation

SEVERITY: **Medium**

LOCATION:

- https://api.example.co
- https://app.example.co

ISSUE DESCRIPTION:

Session tokens are not invalidated after logout. User can use the same session token after logout. In the reason it is possible for attacker to repeat requests as user with any previous token.

PROOF OF VULNERABILITY:

Request for logout.

```
POST /api/v1/logout HTTP/1.1
Host: api.example.co
Connection: close
Content-Length: 2
Origin: https://app.example.co
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/70.0.3538.77 Safari/537.36
Content-Type: application/json
Accept: */*
Referer: https://app.example.co/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie:auth=eyJhbGciOiJIUzM4NCJ9.eyJpc3MiOiJodHRwczovL2FwcC5leGFtcGxlLmNvbSwic3ViIjoidXN
lcnMvcmVkYWN0ZWQiLCJleHAiOjE1NTI0ODEyNzZmUQ.FyR2KNV8Ia8qwrFoLSgz1gGZcZnamKVYgxuWCRWdhybc
S_HgsaN9kMhfRLn4vshZ;

{}
```

Response

```
HTTP/1.1 204 No Content
Date: Thu, 07 Mar 2019 07:10:30 GMT
Connection: close
Access-Control-Allow-Origin: https://app.example.co
Vary: Origin
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers: X-Cache-Date,X-Atmosphere-tracking-id
Set-Cookie: auth=;Path=/;Expires=Thu, 01-Jan-1970 00:00:00 GMT;Max-Age=0;Secure
Expires: Thu, 01 Jan 1970 00:00:00 GMT
```

Request made after Logout with the same value of cookie.

```
GET /api/v1/mailbox HTTP/1.1
Host: api.example.co
```

```
Connection: close
Origin: https://app.example.co
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/70.0.3538.77 Safari/537.36
Accept: */*
Referer: https://app.example.co/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie:auth=eyJhbGciOiJIUzM4NCJ9.eyJpc3MiOiJodHRwczovL2FwcC5leGFtcGxlLmNvbSwic3ViIjoidXN
lcnMvcmVkYWN0ZWQiLCJleHAiOjE1NTI0ODEyNzZmUQ.FyR2KNV8Ia8qwrFoLSgz1gGZcZnamKVYgxuWCRWdhybc
S_HgsaN9kMhfRLn4vshZ
```

Response

```
HTTP/1.1 200 OK
Date: Thu, 07 Mar 2019 07:10:40 GMT
Content-Type: application/json
Content-Length: 3445
Connection: close
Access-Control-Allow-Origin: https://app.example.co
Vary: Origin
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers: X-Cache-Date,X-Atmosphere-tracking-id
Vary: Accept-Encoding

{"mailboxes":[{"uuid":"cf5c954e-852f-4a5c-a9f4-5cf1fe4a8bcd","name":"123","mailboxUsers"
:[{"firstName":"test","lastName":"test","displayName":"test","username":"testtest","emai
l":"stone@stone.teamexample.com","role":"ADMIN",....
```

RECOMMENDATIONS:

The logout function should be prominently visible to the user, explicitly invalidate a user's session and disallow reuse of the session token.

Weak Account Preferences Update Functionality

SEVERITY: **Medium**

LOCATION:

- https://api.example.co
- https://app.example.co

ISSUE DESCRIPTION:

Security-sensitive actions should ask for an additional authentication attempt. Mere logging in to the service should not enable the attacker to perform sensitive actions.

This application allows authenticated user to change email to the new one without additional security checks and proper validation of the account owner. So it is possible for attacker to change password without knowing old one.

PROOF OF VULNERABILITY:

User can update email address for password recovery without any authorization whether user is account owner. If attacker gained users account he is able to change email address and change its password over recovery link next step.

Request

```
PUT /api/v1/user/preferences/1 HTTP/1.1
Host: api.example.co
Connection: close
Content-Length: 306
Origin: https://app.example.co
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/70.0.3538.77 Safari/537.36
Content-Type: application/json
Accept: */*
Referer: https://app.example.co/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: xxx

{"firstName":"Jonh_Test","lastName":"Stone_Test","email":"123@gmail.com","role":"ADMIN",
"mailboxUuids":["1"],"defaultMailboxUuid":"","showMessageSaved":true,"showMessageSent":t
rue,"signature":"","timezone":"Europe/Uzhgorod","defaultSendAndResolve":false,"password"
:"123"}
```

Edited request

```
PUT /api/v1/user/preferences/1 HTTP/1.1
Host: api.example.co
Connection: close
Content-Length: 289
Origin: https://app.example.co
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/70.0.3538.77 Safari/537.36
Content-Type: application/json
Accept: */*
Referer: https://app.example.co/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie:auth=eyJhbGciOiJIUzM4NCJ9.eyJpc3MiOiJodHRwczovL2FwcC5leGFtcGxlLmNvbSwic3ViIjoidXN
lcnMvcmVkYWN0ZWQiLCJleHAiOjE1NTI0ODEyNzZmUQ.FyR2KNV8Ia8qwrFoLSgz1gGZcZnamKVYgxuWCRWdhybc
S_HgsaN9kMhfRLn4vshZ;

{"firstName":"Jonh_Test","lastName":"Stone_Test","email":"123@gmail.com","role":"ADMIN",
"defaultMailboxUuid":"","showMessageSaved":true,"showMessageSent":true,"signature":"","t
imezone":"Europe/Uzhgorod","defaultSendAndResolve":false}
```

Response

```
HTTP/1.1 200 OK
Date: Fri, 01 Mar 2019 15:15:32 GMT
Content-Type: application/json
Content-Length: 527
```

```
Connection: close
Access-Control-Allow-Origin: https://app.example.co
Vary: Origin
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers: X-Cache-Date,X-Atmosphere-tracking-id

{"success":true,"user":{"firstName":"Jonh_Test","lastName":"Stone_Test","displayName":"J
onh_Test","username":"1231231235","email":"123@gmail.com","role":"ADMIN","uuid":"1","acc
ountUuid":"1","gravatarHash":"1","isOwner":true,"mailboxUuids":["1"],"defaultMailboxUuid
":"","showMessageSaved":true,"showMessageSent":true,"signature":"","timezone":"Europe/Uz
hgorod","defaultSendAndResolve":false}}
```

RECOMMENDATIONS:

Implement additional check for sensitive actions of user. Such as password change and email address update. Most common solution is to ask for additional authentication for account settings change.

The additional authentication step can be:

- Give the password again.
- Email confirmation.
- SMS confirmation.
- Give another two-factor authentication token.

Insufficient session expiration mechanism

SEVERITY: **Low**

LOCATION:

- https://app.example.co
- https://api.example.co

ISSUE DESCRIPTION:

Session is active after more than 50 hours of user inactivity. Insufficient session expiration weakness is a result of poorly implemented session management. This weakness can arise on design and implementation levels and can be used by attackers to gain an unauthorized access to the application.

When handling sessions, web developers can rely either on server tokens or generate session identifiers within the application. Each session should be destroyed after the user clicks the Log off button, or after a certain period of time (called timeout). Unfortunately, coding errors and server misconfigurations may influence session handling process, which can result in an unauthorized access.

Session expiration is comprised of two timeout types:

- Inactivity – such timeout is the amount of idle time allowed before the session is invalidated.

- Absolute – such timeout is defined by the total amount of time a session can be valid without re-authentication.

The lack of proper session expiration may increase the likelihood of success of certain attacks. Long expiration time increases an attacker's chance of successfully guessing a valid session ID. The longer the expiration time, the more concurrent open sessions will exist at any given time. The larger the pool of sessions, the more likely it will be for an attacker to guess one at random. Although a short session inactivity timeout does not help if a token is immediately used, the short timeout helps to insure that the token is harder to capture while it is still valid.

RECOMMENDATIONS:

A Web application should invalidate a session after a predefined idle time has passed (a timeout) and provide the user the means to invalidate their own session (log out); this helps to keep the lifespan of a session ID as short as possible and is necessary in a shared computing environment, where more than one person has unrestricted physical access to a computer. More information:

- https://www.owasp.org/index.php/Session_Timeout

Weak Password Policies

SEVERITY: **Low**

LOCATION:

- https://app.example.co
- https://api.example.co

ISSUE DESCRIPTION:

The weakness occurs when the application does not check complexity or minimum length of the provided passwords. Entire security of application depends on its authentication mechanism. Weak password requirements allow users to create weak passwords, susceptible to a verify of attacks.

There should be sufficient validation on both front- and backend parts of application.

PROOF OF VULNERABILITY

It is possible to change password to less than 9 chars while intercepting requests. We were able to set zero length one.

RECOMMENDATIONS:

To mitigate the risk of easily guessed passwords facilitating unauthorized access there are two solutions: introduce additional authentication controls (i.e. two-factor authentication) or introduce a strong password policy. The simplest and cheapest of these is the introduction of a strong password policy that ensures password length, complexity, reuse and aging.

Weak Validation of Origin

SEVERITY: Low

LOCATION:

- https://app.example.co

ISSUE DESCRIPTION:

Cross-domain access should be restricted to a minimal set of domains that are trusted and will require access. An access policy is considered weak or insecure when a wildcard character is used especially in the value of the "uri" attribute.

PROOF OF VULNERABILITY

No origin validation for GET method.

Request

```
GET /api/v1/user/current HTTP/1.1
Host: app.example.co
Connection: close
Origin: https://attacker.com
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/70.0.3538.77 Safari/537.36
Accept: */*
Referer: https://app.example.co/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie:auth=eyJhbGciOiJIUzM4NCJ9.eyJpc3MiOiJodHRwczovL2FwcC5leGFtcGxlLmNvbSwic3ViIjoidXN
lcnMvcmVkYWN0ZWQiLCJleHAiOjE1NTI0ODEyNzZmUQ.FyR2KNV8Ia8qwrFoLSgz1gGZcZnamKVYgxuWCRWdhybc
S_HgsaN9kMhfRLn4vshZ
```

Response

```
HTTP/1.1 200 OK
Date: Thu, 07 Mar 2019 11:00:55 GMT
Content-Type: application/json
Content-Length: 5535
Connection: close
Vary: Accept-Encoding

{"success":true,"user":{"firstName":"John_Test","lastName":"Iron_Test","displayName":"Jo
hn_Test","username":"1231231234","email":"123@gmail.com","role":"ADMIN","uuid":"1","acco
untUuid": ...
```

RECOMMENDATIONS:

- 'Access-Control-Allow-Origin' should be never set to * if the resource contains sensitive information.
- The mitigation is simple and just a proper configuration. Configure the Access-Control-Allow-Origin header to allow requests only from the domains that you trust.
- Make sure that in server side validation for checking the origin header value, you are comparing with absolute value and NOT with regular expression.

  For example: The following code does a comparison with regular expression:

  RegEx("^https://mail.example.com$")

  In the above validation, dots (.) mean any character. So, an attacker can bypass it by making the CORS request origin from following domain: https://mailxexample.com

  The patched code will be:

```
if($_SERVER["HTTP_ORIGIN"] == "https://mail.example.com") {
header("Access-Control-Allow-Origin: https://mail.example.com");
}
```

- Client should not trust the received content without sanitization because that will result in client side code execution. For example: If website abc.com trusts and fetches cross domain data from example.com. example.com has a malicious intent and starts sering malicious javascript to abc.com, then abc.com can protect its users from cross site scripting by sanitizing the received data and then presenting it to its users.

Missing X-Frame-Options Header (Clickjacking)

SEVERITY: **Low**

LOCATION:

- https://app.example.co
- https://example.co

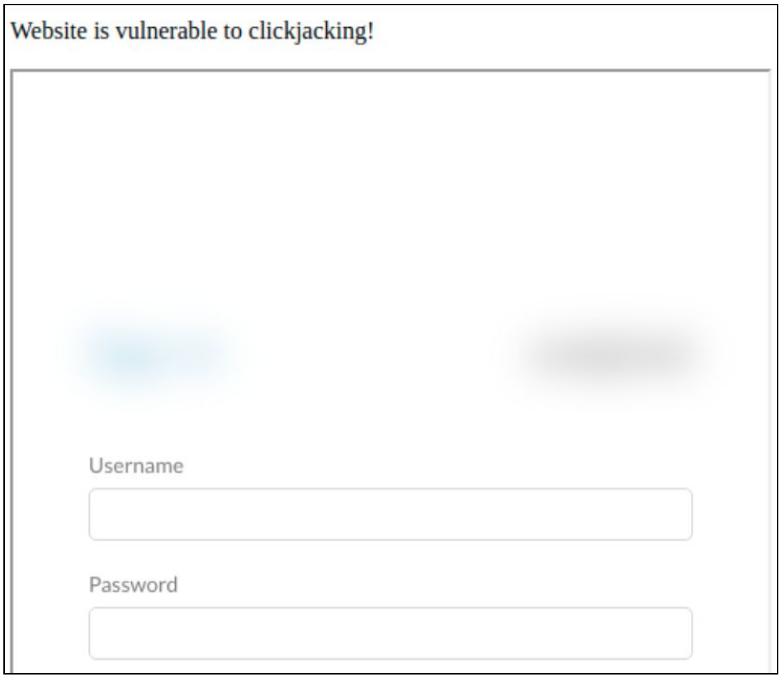
ISSUE DESCRIPTION:

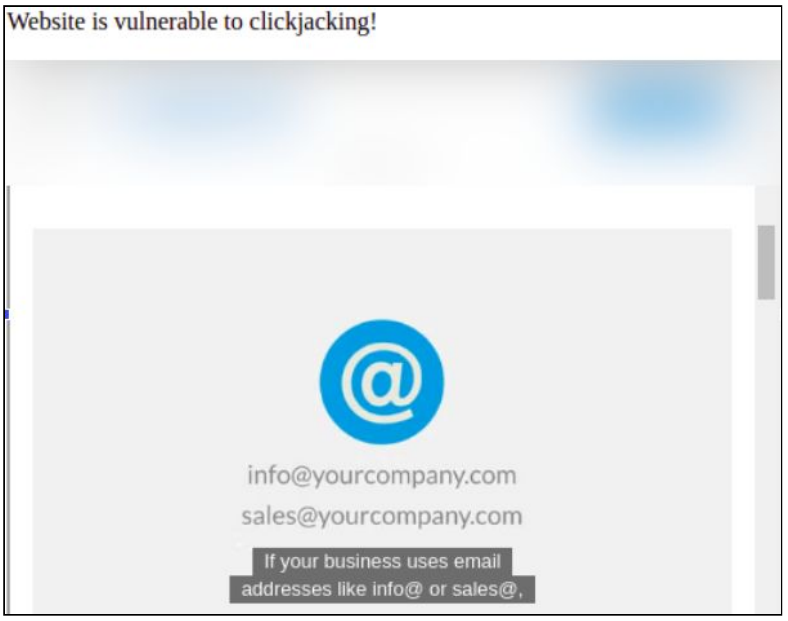
The X-Frame-Options HTTP header field indicates a policy that specifies whether the browser should render the transmitted resource within a frame or an iframe. Servers can declare this policy in the header of their HTTP responses to prevent clickjacking attacks, which ensures that their content is not embedded into other pages or frames.

Clickjacking is when an attacker uses multiple transparent or opaque layers to trick a user into clicking on a button on a framed page when they were intending to click on the top level page.

Using a similar technique, keystrokes can also be hijacked. With a carefully crafted combination of stylesheets, iframes, and text boxes, a user can be led to believe they are typing in the password to their email or bank account, but are instead typing into an invisible frame controlled by the attacker.

PROOF OF VULNERABILITY:





RECOMMENDATIONS:

Widely used tactic to mitigate such issues is sending the proper X-Frame-Options in HTTP response headers that instruct the browser to not allow framing from other domains.

- X-Frame-Options: DENY It completely denies to be loaded in frame/iframe.
- X-Frame-Options: SAMEORIGIN It allows only if the site which wants to load has a same origin.
- X-Frame-Options: ALLOW-FROM URL It grants a specific URL to load itself in a iframe. However please pay attention to that, not all browsers support this.

In other way it is a good practice to employ defensive code in the UI to ensure that the current frame is the most top level window.

Password Brute Force Allowed

SEVERITY: **Low**

LOCATION:

- https://api.example.co/api/v1/login
- https://app.example.co/sign-in

ISSUE DESCRIPTION:

A brute force attack can manifest itself in many different ways, but primarily consists in an attacker configuring predetermined values, making requests to a server using those values, and then analyzing the response. For the sake of efficiency, an attacker may use a dictionary attack (with or without mutations) or a traditional brute-force attack (with given classes of characters e.g.: alphanumerical, special, case (in)sensitive). Considering a given method, number of tries, efficiency of the system which conducts the attack, and estimated efficiency of the system which is attacked the attacker is able to calculate approximately how long it will take to submit

all chosen predetermined values.

PROOF OF VULNERABILITY:

You can try to access account and try to find valid password.

Request

```
POST /api/v1/login HTTP/1.1
Host: api.example.co
Connection: close
Content-Length: 47
Origin: https://app.example.co
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/70.0.3538.77 Safari/537.36
Content-Type: application/json
Accept: */*
Referer: https://app.example.co/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: xxx

{"username":"john.wood","password":"1"}
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 20 Feb 2019 12:11:29 GMT
Content-Type: application/json
Content-Length: 155
Connection: close
Access-Control-Allow-Origin: https://app.example.co
Vary: Origin
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers: X-Cache-Date,X-Atmosphere-tracking-id

{"success":false,"user":null,"mailboxes":[],"accountStatus":null,"onboarding":null,"tags
":[],"users":[],"failureReason":"Password does not match username"}
```

But if you make more than 16 failed attempts to login the account will be blocked for some time and will not allowed to sing-in even with valid credentials. Vulnerability is partly mitigated but attacker can try to make only several requests with delay.

RECOMMENDATIONS:

There are a number of techniques for preventing brute force attacks:

- account lockout policy;
- progressive delays;
- use a challenge-response test to prevent automated submissions of the login page (CAPTCHA);
- IP address lock-out.

Details on how to prevent this attack you can find here:

- https://www.owasp.org/index.php/Blocking_Brute_Force_Attacks

User and E-mail Enumeration

SEVERITY: **Low**

LOCATION:

- https://api.example.co/api/v1/login
- https://www.example.co/api/integration/signup/username-in-use.php

ISSUE DESCRIPTION:

User enumeration is when a malicious actor can use brute-force to either guess or confirm valid users in a system. User enumeration is often a web application vulnerability, though it can also be found in any system that requires user authentication. Two of the most common areas where user enumeration occurs are in a site's login page and its 'Forgot Password' functionality. We have been able to find user enumeration vulnerability on 'Login' and 'Forgot Password' functionality which allows attacker to enumerate existing users.

PROOF OF VULNERABILITY:

User enumeration at login page of https://api.example.co/api/v1/login :

Request

```
POST /api/v1/login HTTP/1.1
Host: api.example.co
Connection: close
Content-Length: 80
Origin: https://app.example.co
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/70.0.3538.77 Safari/537.36
Content-Type: application/json
Accept: */*
Referer: https://app.example.co/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: xxx

{"username":"123@gmail.com","password":"1"}
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 20 Feb 2019 10:32:56 GMT
Content-Type: application/json
Content-Length: 146
Connection: close
Access-Control-Allow-Origin: https://app.example.co
Vary: Origin
Access-Control-Allow-Credentials: true
Access-Control-Expose-Headers: X-Cache-Date,X-Atmosphere-tracking-id

{"success":false,"user":null,"mailboxes":[],"accountStatus":null,"onboarding":null,"tags
":[],"users":[],"failureReason":"Username does not exist"}
```

User enumeration in registration page.

Request

```
POST /api/integration/signup/username-in-use.php HTTP/1.1
Host: www.example.co
Connection: close
Content-Length: 18
Accept: */*
Origin: https://www.example.co
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/70.0.3538.77 Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Referer: https://www.example.co/signup
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: CookieConsent=-1;

userName=123123123
```

Response

```
HTTP/1.1 200 OK
Date: Tue, 26 Feb 2019 11:17:09 GMT
Server: Client Service_Software
Set-Cookie: geoCountryCode=UA; expires=Thu, 28-Mar-2019 11:17:09 GMT; path=/;
domain=.example.co
Content-Length: 864
Connection: close
Content-Type: application/json

{"body":true,"httpCode":200,"curlError":"","curlErrno":0,"curlInfo":{"url":"https:\/\/ap
i.example.co\/integration\/signup\/username-in-use","content_type":"application\/json","
http_code":200,"header_size":131,"request_size":231,"filetime":-1,"ssl_verify_result":0,
"redirect_count":0,"total_time":0.039346,"namelookup_time":0.020667,"connect_time":0.021
356,"pretransfer_time":0.03187,"size_upload":50,"size_download":4,"speed_download":101,"
speed_upload":1270,"download_content_length":4,"upload_content_length":50,"starttransfer
_time":0.039331,"redirect_time":0,"certinfo":[],"redirect_url":""},"headers":{"10023":["
Signature: xxx","Content-Type:
application\/json"],"10002":"https:\/\/api.example.co\/integration\/signup\/username-in-
use","19913":1,"47":1,"10015":"{\"username\":\"123123123\",\"timestamp\":1551179829000}"
},"cookies":[]}
```

RECOMMENDATIONS:

Provide less verbose responses in the functionality. The website should display the same generic message regardless if the username/email address exists or not. A message such as 'Further instructions have been sent to your email address' or similar. For more detailed information please consider using the link below:
https://blog.rapid7.com/2017/06/15/about-user-enumeration/

HSTS missing from HTTPS server

SEVERITY: **Low**

LOCATION:

- https://api.example.co
- https://www.example.co
- https://app.example.co

ISSUE DESCRIPTION:

The remote HTTPS server is not enforcing HTTP Strict Transport Security (HSTS). The lack of HSTS allows downgrade attacks, SSL-stripping man-in-the-middle attacks, and weakens cookie-hijacking protections.

RECOMMENDATIONS:

Configure the remote web server to use HSTS.
More information you can find here:

- https://www.owasp.org/index.php/HTTP_Strict_Transport_Security_Cheat_Sheet

Weak 128 Bit ciphers

SEVERITY: **Low**

LOCATION:

- https://www.example.co

ISSUE DESCRIPTION:

A weak cipher is defined as an encryption/decryption algorithm that uses a key of insufficient length. Using an insufficient length for a key in an encryption/decryption algorithm opens up the possibility (or probability) that the encryption scheme could be broken (i.e. cracked). The larger the key size the stronger the cipher. Weak ciphers are generally known as encryption/ decryption algorithms that use key sizes that are less than 128 bits (i.e., 16 bytes … 8 bits in a byte) in length.

PROOF OF VULNERABILITY:

```
...
Weak 128 Bit ciphers (SEED, IDEA, RC[2,4])    offered (NOT ok)
...
```

RECOMMENDATIONS:

Disable weak ciphers.

Insecure Software Version

SEVERITY: **Low**

LOCATION:

- https://app.example.co/ckeditor/ckeditor.js
- https://app.example.co

ISSUE DESCRIPTION:

When new vulnerabilities are discovered in software, it is important to apply patches and update to a version of the software for which the vulnerability is fixed. Attackers can use known vulnerabilities in their purposes, so security patches should be deployed as soon as they are available.

PROOF OF VULNERABILITY:

Vulnerable ckeditor.js lib:

```
/*
Copyright (c) 2003-2018, CKSource -
Frederico Knabben. All rights reserved.
For licensing, see LICENSE.md or
http://ckeditor.com/license
*/
(function()
{window.CKEDITOR&&window.CKEDITOR.dom||
(window.CKEDITOR||
(window.CKEDITOR=function(){var a=/(^|.*
[\\\/])ckeditor\.js(?:\?.*|;.*)?$/i,g=
{timestamp:"IALL",version:"4.5.11",revision:
"3876e730d",rnd:Math.floor(900*Math.random()
)+100,_:{pending:
[],basePathSrcPattern:a},status:"unloaded",b
asePath:function(){var
b=window.CKEDITOR_BASEPATH||"";if(!b)for(var
e=document.getElementsByTagName("script"),g=
```

Vulnerable lodash.js in app.example.co:

This vulnerability give possibility for attacker to change objects which can lead to errors, denial of service or gain remote code execution. You can read more in this report:

https://hackerone.com/rservice3.comeports/310443

RECOMMENDATIONS:

Update outdated software and always keep it up-to-date.

Possible SWEET32 vulnerability

SEVERITY: **Low**

LOCATION:

- www.example.co

ISSUE DESCRIPTION:

Legacy block ciphers having a block size of 64 bits are vulnerable to a practical collision attack when used in CBC mode. All versions of the SSL/TLS protocols that support cipher suites which use 3DES as the symmetric encryption cipher are affected.

PROOF OF VULNERABILITY:

TestSSL results.

```
 Testing vulnerabilities

...
 POODLE, SSL (CVE-2014-3566)              not vulnerable (OK)
 TLS_FALLBACK_SCSV (RFC 7507)             Downgrade attack prevention supported (OK)
 SWEET32 (CVE-2016-2183, CVE-2016-6329)   VULNERABLE, uses 64 bit block ciphers
 FREAK (CVE-2015-0204)                    not vulnerable (OK)
 DROWN (CVE-2016-0800, CVE-2016-0703)     not vulnerable on this host and port (OK)
                                                make sure you don't use this certificate
elsewhere with SSLv2 enabled services
...
```

RECOMMENDATIONS:

Consider using the following guide to secure web server.

REFERENCE:

https://www.openssl.org/blog/blog/2016/08/24/sweet32/

Possible BREACH vulnerability

SEVERITY: **Low**

LOCATION:

- app.example.co
- www.example.co

ISSUE DESCRIPTION:

This web application is potentially vulnerable to the BREACH attack.

An attacker with the ability to:

- Inject partial chosen plaintext into a victim's requests
- Measure the size of encrypted traffic
- can leverage information leaked by compression to recover targeted parts of the plaintext.

BREACH (Browser Reconnaissance & Exfiltration via Adaptive Compression of Hypertext) is a category of vulnerabilities and not a specific instance affecting a specific piece of software.

To be vulnerable, a web application must:

- Be served from a server that uses HTTP-level compression
- Reflect user-input in HTTP response bodies
- Reflect a secret (such as a CSRF token) in HTTP response bodies

PROOF OF VULNERABILITY:

TestSSL results.

```
 Testing vulnerabilities

 Heartbleed (CVE-2014-0160)                not vulnerable (OK), no heartbeat extension
 CCS (CVE-2014-0224)                       not vulnerable (OK)
  Ticketbleed (CVE-2016-9244), experiment.    not vulnerable (OK), no session ticket
extension
 ROBOT                                     not vulnerable (OK)
 Secure Renegotiation (CVE-2009-3555)      not vulnerable (OK)
 Secure Client-Initiated Renegotiation     VULNERABLE (NOT ok), DoS threat
 CRIME, TLS (CVE-2012-4929)                not vulnerable (OK)
 BREACH (CVE-2013-3587)                              potentially NOT ok, uses gzip HTTP
compression. - only supplied "/" tested
                                                   Can be ignored for static pages or if no
secrets in the page
 POODLE, SSL (CVE-2014-3566)               not vulnerable (OK)
 TLS_FALLBACK_SCSV (RFC 7507)              Downgrade attack prevention supported (OK)
 ...
```

RECOMMENDATIONS:

The mitigations are ordered by effectiveness (not by their practicality - as this may differ from one application to another).

- Disabling HTTP compression
- Separating secrets from user input
- Randomizing secrets per request
- Masking secrets (effectively randomizing by XORing with a random secret per request)
- Protecting vulnerable pages with CSRF
- Length hiding (by adding random number of bytes to the responses)
- Rate-limiting the requests

Possible BEAST vulnerability

SEVERITY: **Low**

LOCATION:

- www.example.co

ISSUE DESCRIPTION:

The SSL protocol, as used in certain configurations in Microsoft Windows and Microsoft Internet Explorer, Mozilla Firefox, Google Chrome, Opera, and other products, encrypts data by using CBC mode with chained initialization vectors, which allows man-in-the-middle attackers to obtain plaintext HTTP headers via a blockwise chosen-boundary attack (BCBA) on an HTTPS session, in conjunction with JavaScript code that uses (1) the HTML5 WebSocket API, (2) the Java URLConnection API, or (3) the Silverlight WebClient API, aka a "BEAST" attack.

PROOF OF VULNERABILITY:

TestSSL results.

```
Testing vulnerabilities

...                                                  https://censys.io/ipv4?q=123 could help
you to find out
 LOGJAM (CVE-2015-4000), experimental      not vulnerable (OK): no DH EXPORT ciphers, no
DH key detected
 BEAST (CVE-2011-3389)                    TLS1: ECDHE-RSA-AES128-SHA
                                                ECDHE-RSA-AES256-SHA
                                                AES128-SHA AES256-SHA
                                                 VULNERABLE -- but also supports higher
protocols  TLSv1.1 TLSv1.2 (likely mitigated)
...
```

RECOMMENDATIONS:

Disable TLS 1.0 and have users connect using TLS 1.1 or TLS 1.2 protocols which are immune to the BEAST attack. TLS 1.0 is now considered insecure and disabling the protocol improves the overall security.

REFERENCE:

https://www.acunetix.com/blog/articles/tls-ssl-cipher-hardening

Possible LUCKY13 vulnerability

SEVERITY: **Low**

LOCATION:

- www.example.co
- app.example.co
- api.example.co

ISSUE DESCRIPTION:

The TLS protocol 1.1 and 1.2 and the DTLS protocol 1.0 and 1.2, as used in OpenSSL, OpenJDK, PolarSSL, and other products, do not properly consider timing side-channel attacks on a MAC check requirement during the processing of malformed CBC padding, which allows remote attackers to conduct distinguishing attacks and plaintext-recovery attacks via statistical analysis of timing data for crafted packets, aka the "Lucky Thirteen" issue.

PROOF OF VULNERABILITY:

TestSSL results.

```
Testing vulnerabilities
... LOGJAM (CVE-2015-4000), experimental      not vulnerable (OK): no DH EXPORT ciphers,
no DH key detected

 LUCKY13 (CVE-2013-0169), experimental       potentially VULNERABLE, uses cipher block
chaining (CBC) ciphers with TLS. Check patches
 RC4 (CVE-2013-2566, CVE-2015-2808)        no RC4 ciphers detected (OK)
...
```

RECOMMENDATIONS:

Avoid using TLS in CBC-mode and to switch to using AEAD algorithms.

REFERENCE:

https://blog.cloudflare.com/new-ssl-vulnerabilities-cloudflare-users-prot/

Leaked data on several exposed databases

SEVERITY: **Informational**

ISSUE DESCRIPTION:

The definition of data leakage is the unauthorized transmission of data from within an organization to an external destination or recipient. Data leakage threats usually occur via the web and email, but can also occur via mobile data storage devices such as optical media, USB keys, and laptops. There are a lot of possible threats which can lead to sensitive data exposure like breach of Third-Party service providers or phishing attack.
We have found leaked information including corporate emails exposed on several data breaches.

PROOF OF VULNERABILITY:

List of exposed mails and where it was found:

Mails	Leaked Database
alain@example.com	Anti Public Combo List
alain@example.com	Exploit.in
bwhalen@example.com	Anti Public Combo List
curt@example.com	Exploit.in
hello@example.com	Anti Public Combo List
hello@example.com	Exploit.in
kyle@example.com	LinkedIn
kyle@example.com	Exploit.in
miked@example.com	Exploit.in
miked@example.com	LinkedIn
noah@example.com	Exploit.in
noah@example.com	Exploit.in
sean@example.com	Exploit.in
sean@example.com	Anti Public Combo List
sean@example.com	LinkedIn
tberry@example.com	Anti Public Combo List
tberry@example.com	Exploit.in
timg@example.com	Anti Public Combo List

RECOMMENDATIONS:

Employees should be aware of the risks of reusing corporate email and passwords for non-work related purposes. In case there is need for creating account on third party services using corporate emails, employees should create new password for each service. We recommend to use password managers.

Cloud Infrastructure Findings Details

CloudTrail not configured

SEVERITY: **High**

ISSUE DESCRIPTION:

AWS CloudTrail records all AWS API calls to your account in a log file. The recorded information includes the IP address of the API caller, the time of the API call, the identity (username) of the API caller, the request parameters, and the response elements returned. Enabling CloudTrail on your AWS account provides a history of all API calls for your account, including calls from the AWS Management Console, command line tools, AWS SDK, and other AWS Services like CloudFormation. This audit log allows for in depth security analysis and insight into resource changes.

RECOMMENDATIONS:

Ensure that your CloudTrail trails are recording both regional and global events in order to increase the visibility of the API activity in your AWS account for security and management purposes

For more detailed information please consider using the link below:

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/monitor-with-cloudtrail.html

EBS volume not encrypted

SEVERITY: **High**

ISSUE DESCRIPTION:

When dealing with production data that is crucial to your business, it is highly recommended to implement encryption in order to protect it from attackers or unauthorized personnel. With Elastic Block Store encryption enabled, the data stored on the volume, the disk I/O and the snapshots created from the volume are all encrypted. The EBS encryption keys use AES-256 algorithm and are entirely managed and protected by the AWS key management infrastructure, through AWS Key Management Service (AWS KMS).

With encryption enabled, your EBS volumes can hold very sensitive and critical data. The EBS encryption and decryption is handled transparently and does not require any additional action from you, your server instance, or your application.

PROOF OF VULNERABILITY:

vol-xxxxxxxxxxxxxxxx

Attributes

- Attachments:
  - 0:
    - AttachTime: 2019-02-21 23:41:52+00:00
    - DeleteOnTermination: true
    - Device: /dev/sda1
    - InstanceId: xxxxxxxxxxxxxxx
    - State: attached
    - VolumeId: vol-xxxxxxxxxxxxxxxx
- AvailabilityZone: us-east-1b
- CreateTime: 2019-02-21 23:41:52.337000+00:00
- Encrypted: false
- Iops: 100
- LastSnapshotDate:
- Size: 8
- SnapshotId: snap-xxxxxxxxxxxxxxxx
- State: in-use
- VolumeType: gp2
- id: vol-xxxxxxxxxxxxxxxx
- name: vol-xxxxxxxxxxxxxxxx
- resource:
  - Attachments:
    - 0:
      - AttachTime: 2019-02-21 23:41:52+00:00
      - DeleteOnTermination: true
      - Device: /dev/sda1
      - InstanceId: i-xxxxxxxxxxxxxxxx
      - State: attached
      - VolumeId: vol-xxxxxxxxxxxxxxxx
  - AvailabilityZone: us-east-1b
  - CreateTime: 2019-02-21 23:41:52.337000+00:00
  - Encrypted: false
  - Iops: 100
  - LastSnapshotDate:
  - Size: 8
  - SnapshotId: snap-xxxxxxxxxxxxxxxx
  - State: in-use
  - VolumeType: gp2
  - id: vol-xxxxxxxxxxxxxxxx
  - name: vol-xxxxxxxxxxxxxxxx

RECOMMENDATIONS:

That is recommended to keep the data stored on the volume, the disk I/O and the snapshots created from the volume all encrypted.

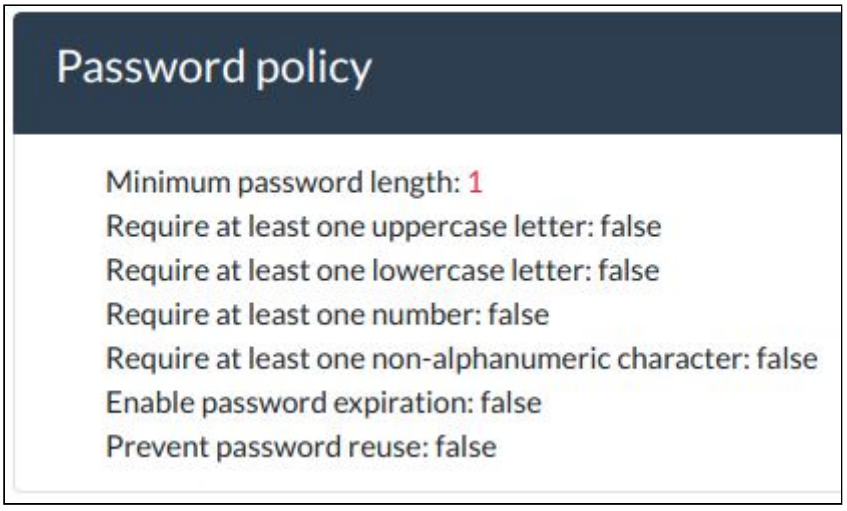
To determine if your EBS volumes are encrypted, please consider using the link below:
https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSEncryption.html

Weak Account Password Policy for IAM Users

SEVERITY: **Medium**

ISSUE DESCRIPTION:

PROOF OF VULNERABILITY:



RECOMMENDATIONS:

For more detailed information please consider using the links below:

https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_passwords_account-policy.html

https://www.cloudconformity.com/conformity-rules/IAM/password-policy.html

Cross-account AssumeRole policy lacks external ID and MFA

SEVERITY: **Medium**

PROOF OF VULNERABILITY:

RECOMMENDATIONS:

Ensure that Amazon IAM roles used to establish a trusted relationship between your AWS account and a third-party entity (also known as cross-account access roles) are using Multi-Factor Authentication (MFA) or external IDs to secure the access to your resources and to prevent "confused deputy" attacks. The MFA/external ID adds an extra layer of security on top of roles temporary security credentials and facilitates external third-party accounts to access your AWS resources in a secure way.

When authorizing cross-account role assumption, an external ID or MFA should be required. For more detailed information please consider using the link below:

https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create_for-user.html

Users without MFA

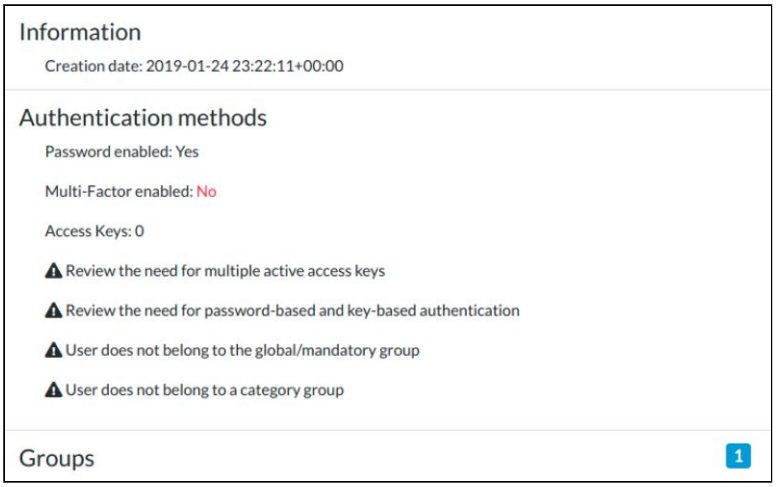
SEVERITY: **Medium**

ISSUE DESCRIPTION:

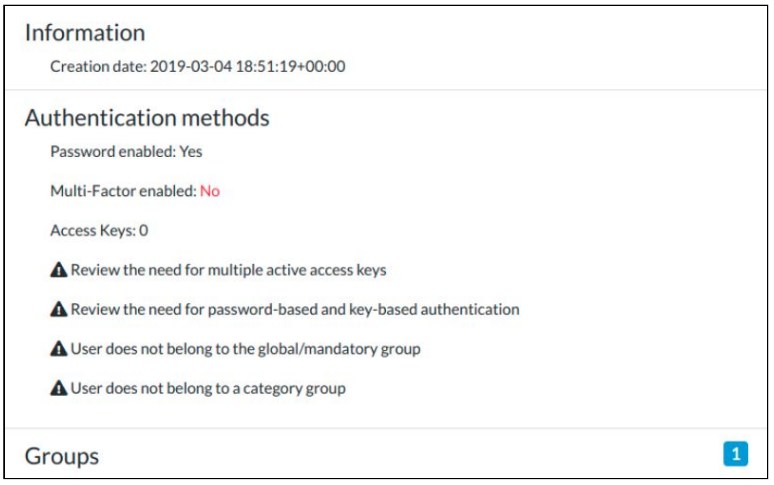
Having MFA-protected IAM users is the best way to protect your AWS resources and services against attackers. An MFA device signature adds an extra layer of protection on top of your existing IAM user credentials (username and password), making your AWS account virtually impossible to penetrate without the MFA generated passcode.

PROOF OF VULNERABILITY:

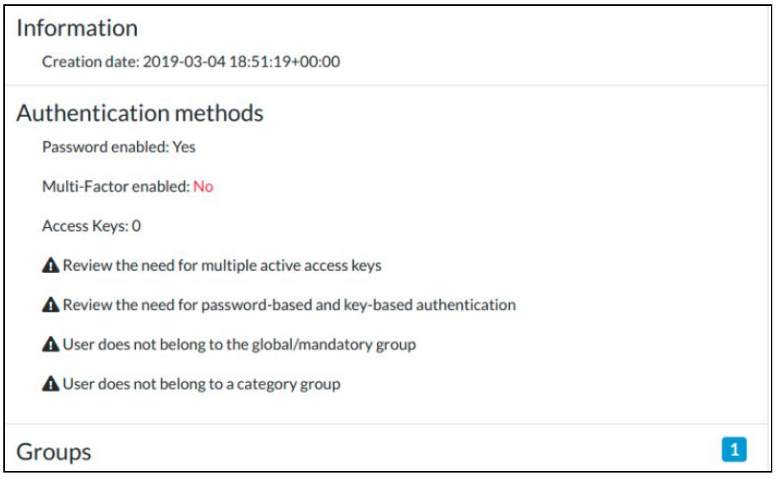
First user:



Second user:



Third user:



RECOMMENDATIONS:

For more detailed information please consider using the link below:

https://www.cloudconformity.com/conformity-rules/IAM/iam-user-multi-factor-authentication-enabled.html

APPENDIX A - Scope

DNS Name/ IP Address
example.co app.example.co client_service-production.redacted.us-east-1.rds.amazonaws.com client_service-production.redacted.ng.0006.sds1.cache.amazonaws.com search-client_service-production-redacted.us-east-1.es.amazonaws.com