UnderDefense
CyberSecurity solutions protecting your business

REST API Penetration Testing Report for [CLIENT]

Executive summary

This report presents the results of the “Grey Box” penetration testing for [CLIENT] REST API. 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] Grey Box REST API 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 Web API 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 API perimeter	Good	B

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 ORGANIZATION]
Audit type	Gray Box REST API Penetration Testing
Asset URL	● m.example1.com ● m.example2.com ● m.example3.com ● example4.com ● example5.co.uk ● example6.com ● example7.com
Audit period	Feb. 25 - Mar. 15, 2019

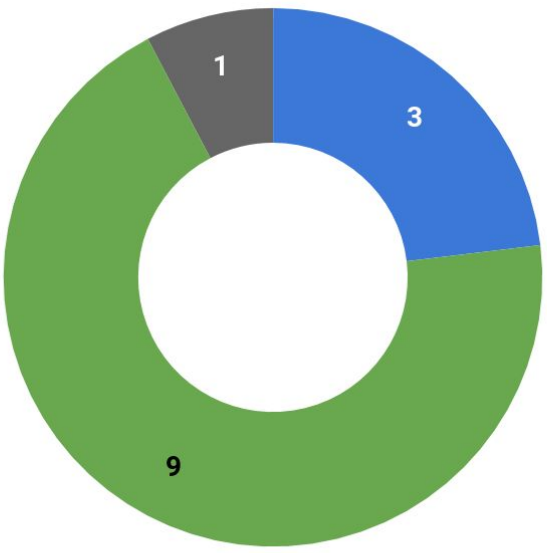
Consultants performed discovery process to gather information about the target and searched for information disclosure vulnerabilities. With this data in hand, we conducted the bulk of the testing manually, which consisted of input validation tests, impersonation (authentication and authorization) tests, 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 a few vulnerabilities that may cause sensitive data leakage, broken confidentiality and integrity and availability of the resource.

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

Vulnerabilities by severity



Security experts performed manual security testing according to OWASP Web Application Testing Methodology, which demonstrate the following results.

Severity	Critical	High	Medium	Low	Informational
# of issues	0	0	3	9	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

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.
- 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
- Conduct Static code analysis for codebase
- 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
- 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
- Do not send any unnecessary data in requests and cookies
- Improve server and application configuration to meet security best practises

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	Fails criteria
A4:2017-XL 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

Security tools used

- Burp Suite Pro [Commercial Edition]
- Nmap
- TestSSL
- SQLmap
- Different Burp Suite plugins (Joseph, etc.)

Project limitations

The Grey Box assessment was conducted against production environment with all limitations, it provides.

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
- OWASP ASVS

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.

Findings Details

User enumeration

SEVERITY: **Medium**

LOCATION:

- /api/v1/ge/forgottenpasswords

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 'Sign Up', and 'Forgot Password' functionality which allows attackers to enumerate existing users.

PROOF OF VULNERABILITY:

User enumeration over *forgottenpasswords* API call on:

- m.example1.com
- m.example2.com
- m.example3.com
- m.example4.com

Request:

```
POST /api/v1/ge/forgottenpasswords HTTP/1.1
marketCode: en
Content-Type: application/json
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 9_1 like Mac OS X) AppleWebKit/601.1.46
(KHTML, like Gecko) Version/9.0 Mobile/13B143 Safari/601.1
cache-control: no-cache
Accept: */*
Host: m.example2.com
Accept-Encoding: gzip, deflate
Content-Length: 140
Connection: close

{
  "Identifier": "hacker1",
  "ResetType": "SMS",
  "NationalId": "0as9aa0d596697",
  "PhoneExtension": "+380",
  "PhoneNumber": "2213"
}
```

If user exists we receive RESPONSE:

```
{"code":"E_FORGOTTENPASSWORDS_INVALIDCREDENTIALS"}
```

User enumeration on www.example7.com in */en/api/Authentication/Login* API call (out of scope item):

If password is invalid response looks like:

```
HTTP/1.1 200 OK
Cache-Control: no-cache,no-store, no-cache, must-revalidate, post-check=0, pre-check=0
...
{"Success":false,"Message":"InvalidCredentials","UserNeedsToAcceptTerms":false,"Paramete
rs":null,"ActivityId":"5a836419-57b0-471f-9dd7-67c915b1963c","LastLoginDate":null,"Statu
sCode":401}
```

If username is invalid response looks like:

```
HTTP/1.1 200 OK
Cache-Control: no-cache,no-store, no-cache, must-revalidate, post-check=0, pre-check=0
...
{"Success":false,"Message":"UserStatusUnknown","UserNeedsToAcceptTerms":false,"Parameter
s":null,"ActivityId":"6beee024-7363-4d53-8cb3-194f5837deb4","LastLoginDate":null,"Status
Code":404}
```

RECOMMENDATIONS:

Provide less verbose responses in the functionality. The server should return the same generic messages 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/

Session token in local storage

SEVERITY: **Medium**

LOCATION:

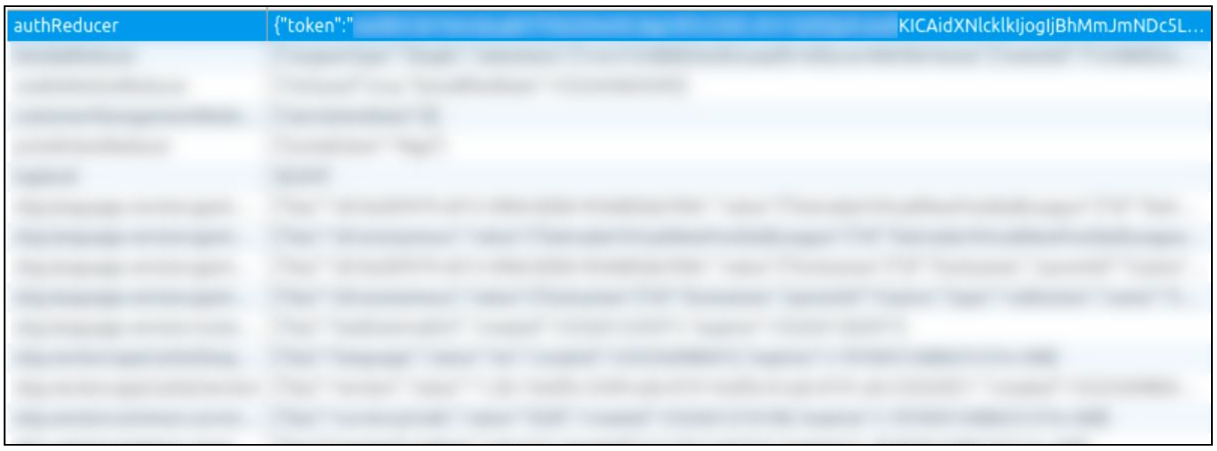
- https://m.example1.com/en
- https://m.example2.com/en/
- https://m.example3.com/en
- https://www.m.example4.com
- https://www.m.example5.co.uk

ISSUE DESCRIPTION:

A JWT needs to be stored in a safe place inside the user's browser. If you store it inside localStorage, it's accessible by any script inside your page (which is as bad as it sounds as an XSS attack can let an external attacker get access to the token).

PROOF OF VULNERABILITY:

Session token in local storage:



authReducer	{"token":"[redacted]KICAidXNlcklkIjogIjBhMmJmNDc5L...

[redacted]

RECOMMENDATIONS:

Don't store session token in local storage (or session storage). If any of the 3rd part scripts you include in your page gets compromised, it can access all your users' tokens.

Data Validation

SEVERITY: **Medium**

LOCATION:

- /api/v1/customerregistrationtokens

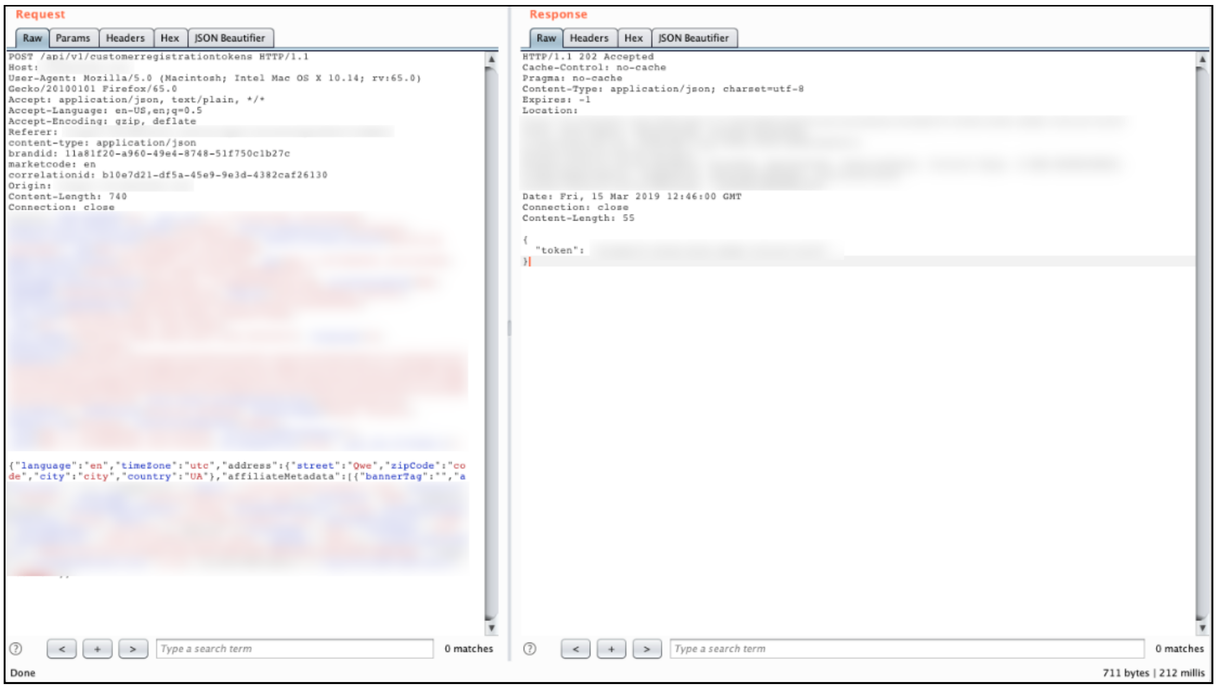
ISSUE DESCRIPTION:

The most common web application security weakness is the failure to properly validate input from the client or environment. This weakness leads to almost all of the major vulnerabilities in applications, such as Interpreter Injection, locale/Unicode attacks, file system attacks and buffer overflows. Data from the client should never be trusted for the client has every possibility to tamper with the data.

In many cases, Encoding has the potential to defuse attacks that rely on lack of input validation. For example, if you use HTML entity encoding on user input before it is sent to a browser, it will prevent most XSS attacks. However, simply preventing attacks is not enough – you must perform Intrusion Detection in your applications. Otherwise, you are allowing attackers to repeatedly attack your application until they find a vulnerability that you haven't protected against. Detecting attempts to find these weaknesses is a critical protection mechanism.

PROOF OF VULNERABILITY:

We are able to insert any data to fields as city, Zip code, street while registering/editing accounts.



```
POST /api/v1/customerregistrationtokens HTTP/1.1
Host: [redacted]
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:65.0)
Gecko/20100101 Firefox/65.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: [redacted]
content-type: application/json
brandid: 11a81f20-a960-49e4-8748-51f750c1b27c
marketcode: en
correlationid: b10e7d21-df5a-45e9-9e3d-4382caf26130
Origin: [redacted]
Content-Length: 740
Connection: close

[redacted]
{"language":"en","timeZone":"utc","address":{"street":"Qwe","zipCode":"co
de","city":"city","country":"UA"},"affiliateMetadata":[{"bannerTag":"","a
[redacted]
```

```
HTTP/1.1 202 Accepted
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/json; charset=utf-8
Expires: -1
Location:
[redacted]
Date: Fri, 15 Mar 2019 12:46:00 GMT
Connection: close
Content-Length: 55

{
  "token": [redacted]
}
```

Request body	Response

{"language":"en",
"timeZone":"utc",
"address":{
"street":"Qwe",
"zipCode":"code",
"city":"city",
"country":"UA"},"
affiliateMetadata":[
{"bannerTag":"","affiliate":""}],
"credential":
{"email":"attacker@example.com",
"password":"Weak6c"
,"username":"attacker@example.com"},
"currency":"EUR",
"subscriptions":
{"acceptEmailOffers":false,
"acceptSmsOffers":false,"
acceptTelesalesOffers":false,
"email":"attacker@example.com",
"phoneExtension":"+380"
,"phoneNumber":"33212222"}
,"person":{
"firstName":"John",
"lastName":"Doe"
,"dateOfBirth"
:"1995-03-04T00:00:00.000Z",
"gender":"Male"},
"legal":{"termsAndConditions":true},
"productMetadata":{
"registeredFromProduct":"common"}}
	HTTP/1.1 202 Accepted
	...
	{
	  "token":
	"6e3de379-6f2d-4246-a7dd-7c5c3a812c59"
	}

RECOMMENDATIONS:

Every user input field should be validated both on front end and back end.

https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Input_Validation_Cheat_Sheet.md

Possible BREACH vulnerability

SEVERITY: **Low**

LOCATION:

- m.example5.co.uk

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
 ...
 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
 ...
```

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:

- m.example5.co.uk
- m.example4.com
- m.example1.com
- m.example2.com
- m.example3.com

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
...
 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:

- m.example5.co.uk
- m.example4.com
- m.example1.com
- m.example2.com
- m.example3.com

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

...
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)
 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/

Clickjacking

SEVERITY: **Low**

LOCATION:

- m.example5.co.uk
- m.example4.com
- m.example1.com
- m.example2.com
- m.example3.com

ISSUE DESCRIPTION:

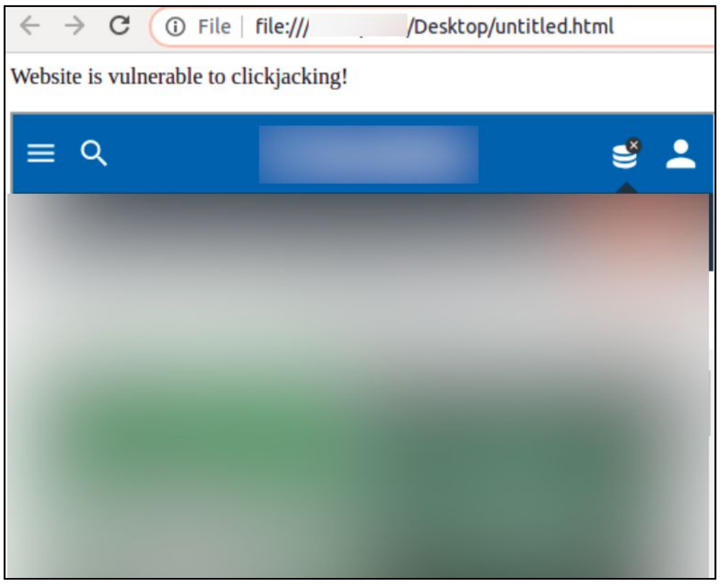
Clickjacking, also known as a "UI redress attack", is when an attacker uses multiple transparent or opaque layers to trick a user into clicking on a button or link on another page when they were intending to click on the top level page. Thus, the attacker is "hijacking" clicks meant for their page and routing them to another page, most likely owned by another application, domain, or both.

PROOF OF VULNERABILITY:

Html code which creates iframe of the website.

```
<html>
   <head>
     <title>Clickjack test page</title>
   </head>
   <body>
     <p>Website is vulnerable to clickjacking!</p>
     <iframe src="https://example1.com/" width="600" height="300"></iframe>
   </body>
</html>
```

As a result it’s possible to create iframe of the website.



File | file:///[redacted]/Desktop/untitled.html

Website is vulnerable to clickjacking!

RECOMMENDATIONS:

There are two main ways to prevent clickjacking:

- Sending the proper Content Security Policy (CSP) frame-ancestors directive response headers that instruct the browser to not allow framing from other domains. (This replaces the older X-Frame-Options HTTP headers.)
- Employing defensive code in the UI to ensure that the current frame is the most top level window.

REFERENCE:

https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet

Outdated components with known vulnerabilities

SEVERITY: **Low**

LOCATION:

- www.m.example4.com
- www.m.example5.co.uk
- https://subdomain.example6.com/wp-json/ (out of scope)

ISSUE DESCRIPTION:

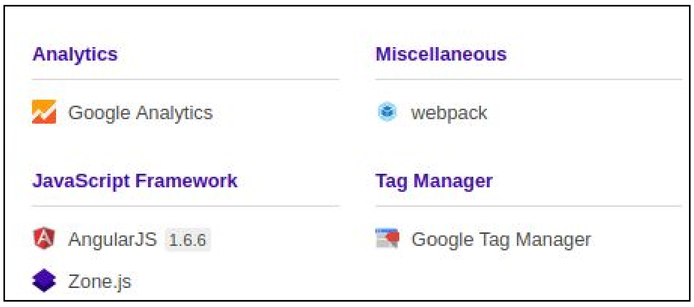
When new vulnerabilities are discovered in software, it's important to apply patches and update to a version of the software for which the vulnerability is fixed. Attackers can use known vulnerabilities, so security patches should be deployed as soon as they are available.

We have been able to find outdated version of Angular and Nginx services which are exposed for several known vulnerabilities.

PROOF OF VULNERABILITY:

Vulnerability in angularJS:

https://snyk.io/test/npm/angular/1.6.6

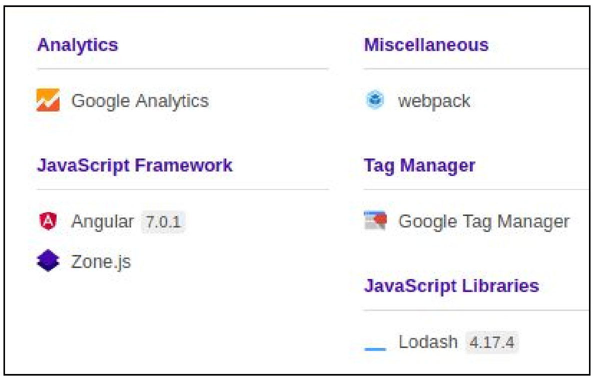


Analytics
Google Analytics
JavaScript Framework
AngularJS 1.6.6
Zone.js
Miscellaneous
webpack
Tag Manager
Google Tag Manager

Vulnerability in Lodash and Angular build:

https://snyk.io/test/npm/lodash/4.17.4

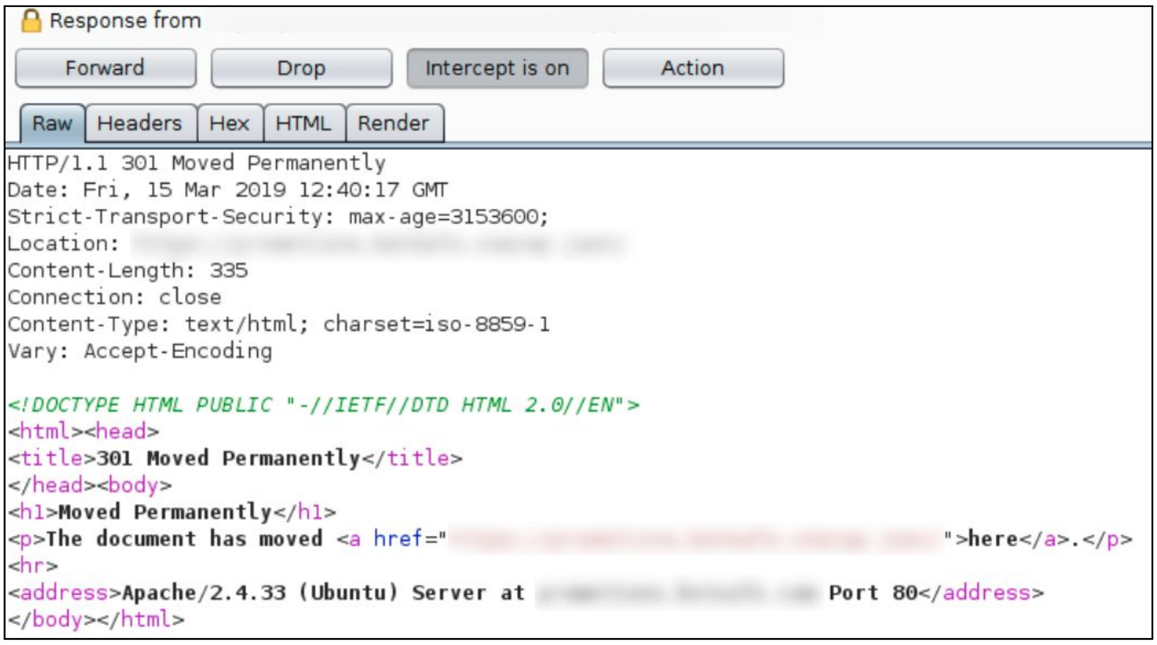
https://snyk.io/test/npm/@bizappframework/angular-build



Analytics
Google Analytics
JavaScript Framework
Angular 7.0.1
Zone.js
Miscellaneous
webpack
Tag Manager
Google Tag Manager
JavaScript Libraries
Lodash 4.17.4

Vulnerable Apache server:

https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/version_id-252825/Apache-Http-Server-2.4.33.html



```
HTTP/1.1 301 Moved Permanently
Date: Fri, 15 Mar 2019 12:40:17 GMT
Strict-Transport-Security: max-age=3153600;
Location: [redacted]
Content-Length: 335
Connection: close
Content-Type: text/html; charset=iso-8859-1
Vary: Accept-Encoding

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="[redacted]">here</a>.</p>
<hr>
<address>Apache/2.4.33 (Ubuntu) Server at [redacted] Port 80</address>
</body></html>
```

RECOMMENDATIONS:

Update outdated software and always keep it up-to-date in order to avoid the threat of known vulnerabilities.

Insufficient Brute-force protection

SEVERITY: **Low**

LOCATION:

- m.example1.com

ISSUE DESCRIPTION:

A system administrator uses a weak and guessable password which is easy to bruteforce and gain control over administration functionality. Obvious and easy to remember passwords can also be brute forced easily.

There is no Brute-force protection mechanism on login page. Attackers can easily provide brute-force attack to gain access to administrative console.

RECOMMENDATIONS:

We strongly recommend to use long passwords that can't be guessed easily.

It is recommended to use a challenge-response test to prevent automated submissions of the login page. Tools such as the free reCAPTCHA can be used to require the user to enter a word or solve a simple math problem to ensure the user is, in fact, a person.

Other way progressive delays technique may be used. With progressive delays, user accounts are locked out for a set period of time after a few failed login attempts. The lock-out time increases with each subsequent failed attempt. This prevents automated tools from performing a brute force attack and effectively makes it impractical to perform such an attack.

Weak Lock out mechanism

SEVERITY: **Low**

LOCATION:

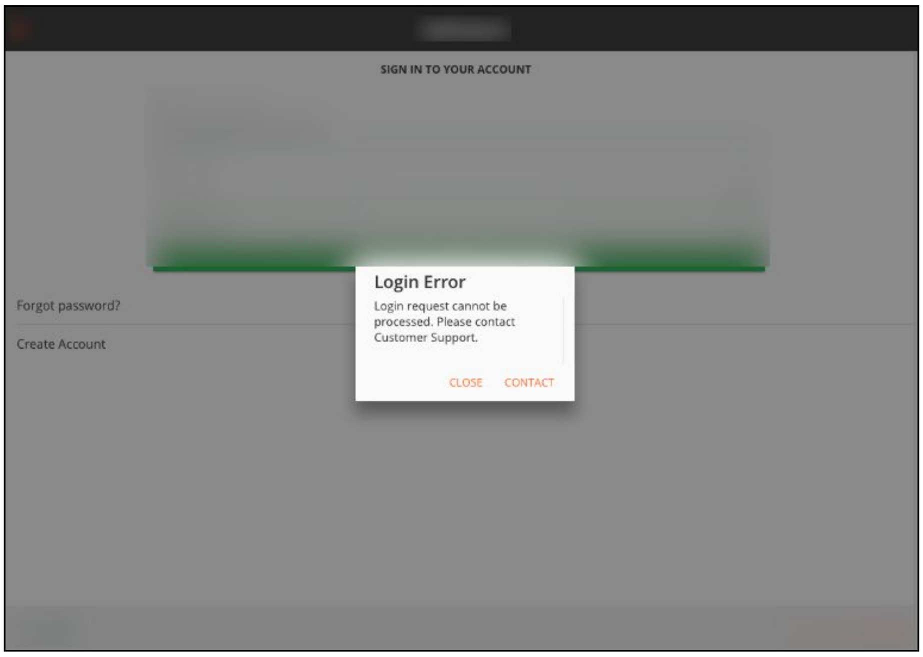
- /api/v1/single-sign-on-sessions/

ISSUE DESCRIPTION:

Account lockout mechanisms are used to mitigate brute force password guessing attacks. Accounts are typically locked after 3 to 5 unsuccessful login attempts and can only be unlocked after a predetermined period of time, via a Customer Support service or intervention by an administrator. Account lockout mechanisms require a balance between protecting accounts from unauthorized access and protecting users from being denied authorized access.

PROOF OF VULNERABILITY:

User account locked after 5 login attempts.



Login Error

Login request cannot be processed. Please contact Customer Support.

RECOMMENDATIONS:

Apply account unlock mechanisms depending on the risk level. In order from lowest to highest assurance:

- Time-based lockout and unlock.
- Self-service unlock (sends unlock email to registered email address).
- Manual administrator unlock.
- Manual administrator unlock with positive user identification.

Weak password policy

SEVERITY: **Low**

LOCATION:

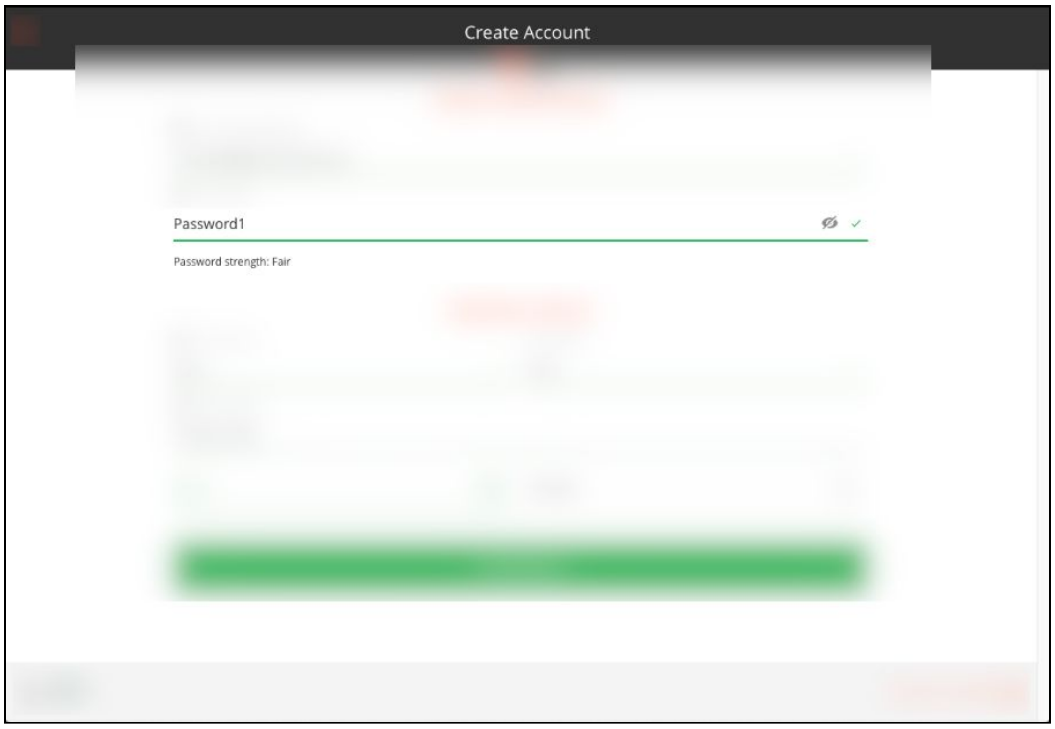
- /en/open-account?product=common
- /api/v1/customerregistrationtokens

ISSUE DESCRIPTION:

The most common web application security weakness is the failure to properly validate input from the client or environment. This weakness leads to almost all of the major vulnerabilities in applications, such as Interpreter Injection, locale/Unicode attacks, file system attacks and buffer overflows. Data from the client should never be trusted for the client has every possibility to tamper with the data.

In many cases, Encoding has the potential to defuse attacks that rely on lack of input validation. For example, if you use HTML entity encoding on user input before it is sent to a browser, it will prevent most XSS attacks. However, simply preventing attacks is not enough - you must perform Intrusion Detection in your applications. Otherwise, you are allowing attackers to repeatedly attack your application until they find a vulnerability that you haven't protected against. Detecting attempts to find these weaknesses is a critical protection mechanism.

PROOF OF VULNERABILITY:



RECOMMENDATIONS:

According to OWASP Application Security Verification Standard, passwords should match next criterias:

- User set passwords must be at least 12 characters in length.
- Unicode characters must be permitted in passwords.
- Password shouldn't have composition rules limiting the type of characters permitted.

Also, according to NIST 800-63, passwords, called "Memorized Secrets", shouldn't be:

- Passwords obtained from previous breach corpuses.
- Dictionary words.
- Repetitive or sequential characters (e.g. 'aaaaaa', '1234abcd').
- Context-specific words, such as the name of the service, the username, and derivatives thereof.

Reflected XSS in url address

SEVERITY: **Low**

LOCATION:

- subdomain.example7.com/download/de/?'"">"<body><div><script>alert(1)</script></
- subdomain.example6.com/download/de/?'"">"<body><div><script>alert(1)</script></

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 in the output it generates without validating or encoding it.

Reflected attacks are those where the injected script is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request. Reflected attacks are delivered to victims via another route, such as in an email message, or on some other website. When a user is tricked into clicking on a malicious link, submitting a specially crafted form, or even just browsing to a malicious site, the injected code travels to the vulnerable web site, which reflects the attack back to the user’s browser. The browser then executes the code because it came from a "trusted" server.

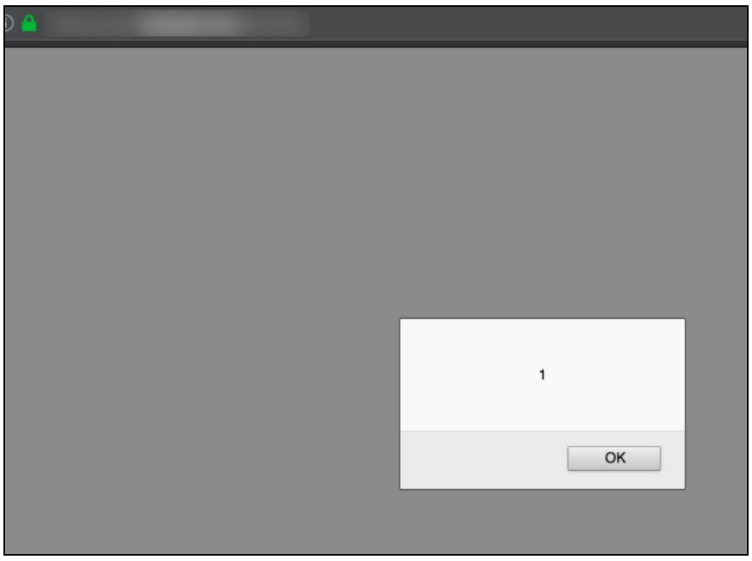
PROOF OF VULNERABILITY:

Request

```
GET
/download/de/?'"">"<body><div><script>alert(localStorage.getItem('authReducer')</script>
</div><img%20src='x'%20onerror="alert(1)</body> HTTP/1.1
Host: subdomain.example6.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:65.0) Gecko/20100101
Firefox/65.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
```

Response

```
...
<!-- Open Graph -->
            <meta property="og:title" content=""/>    <meta property="og:type"
content="website"/>
    <meta property="og:url"
content="https://subdomain.example6.com/download/de/?'"">"<body><div><script>alert(local
Storage.getItem('authReducer')</script></div><img%20src='x'%20onerror="alert(1)</body>"/
>
...
```



RECOMMENDATIONS:

Use verification and sanitization on both client and server side, For more detailed information, please see the link below:
https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

Informative Error Messages

SEVERITY: **Informational**

LOCATION:

- /api/v1/customerregistrationtokens
- https://subdomain.example6.com/wp-json

ISSUE DESCRIPTION:

During a penetration testing, we come up against some error codes generated from applications or web servers. It's possible to cause these errors to be displayed by using a particular requests, either specially crafted with tools or created manually. These errors containing information about server and it's version.

Such information is very useful for hackers during their activities, because they reveal a lot of information about databases, bugs, and other components directly linked with web applications.

Do not expose sensitive information in error messages and server headers. Any system internal information should be hidden from the user.

PROOF OF VULNERABILITY:

It's possible to cause these errors to be displayed by using a particular requests, either specially crafted with tools or created manually.

Request

```
POST /api/v1/customerregistrationtokens HTTP/1.1
Host: m.example3.com
...
{"language":"en","timeZone":"utc","address":{"street":"sad","zipCode":"qwe","city":"qwe"
,"country":"UA"},"affiliateMetadata":[{"bannerTag":"","affiliate":""}],"credential":{"em
ail":"attacker@example.com","password":"Qwerty123456","username":"attacker@example.com"}
,"currency":"EUR","subscriptions":{"acceptEmailOffers":false,"acceptSmsOffers":false,"ac
ceptTelesalesOffers":false,"email":"attacker@example.com","phoneExtension":"+380","phone
Number":"0911111111"},"person":{"firstName":"Test","lastName":"Test","dateOfBirth":"1990
-03-13T00:00:00.000Z","gender":"Male"},"legal":{"termsAndConditions":true},"productMetad
ata":{"registeredFromProduct":"common"}}
```

Response

```
HTTP/1.1 415 Unsupported Media Type
Cache-Control: no-cache
...

{
  "message": "The request contains an entity body but no Content-Type header. The
inferred media type 'application/octet-stream' is not supported for this resource.",
  "exceptionMessage": "No MediaTypeFormatter is available to read an object of type
'MgaRegistrationRequest' from content with media type 'application/octet-stream'.",
  "exceptionType": "System.Net.Http.UnsupportedMediaTypeException",
  "stackTrace": "   at System.Net.Http.HttpContentExtensions.ReadAsAsync[T](HttpContent
content, Type type, IEnumerable`1 formatters, IFormatterLogger formatterLogger,
CancellationToken cancellationToken)\r\n   at
System.Web.Http.ModelBinding.FormatterParameterBinding.ReadContentAsync(HttpRequestMessa
ge request, Type type, IEnumerable`1 formatters, IFormatterLogger formatterLogger,
CancellationToken cancellationToken)"
}
```

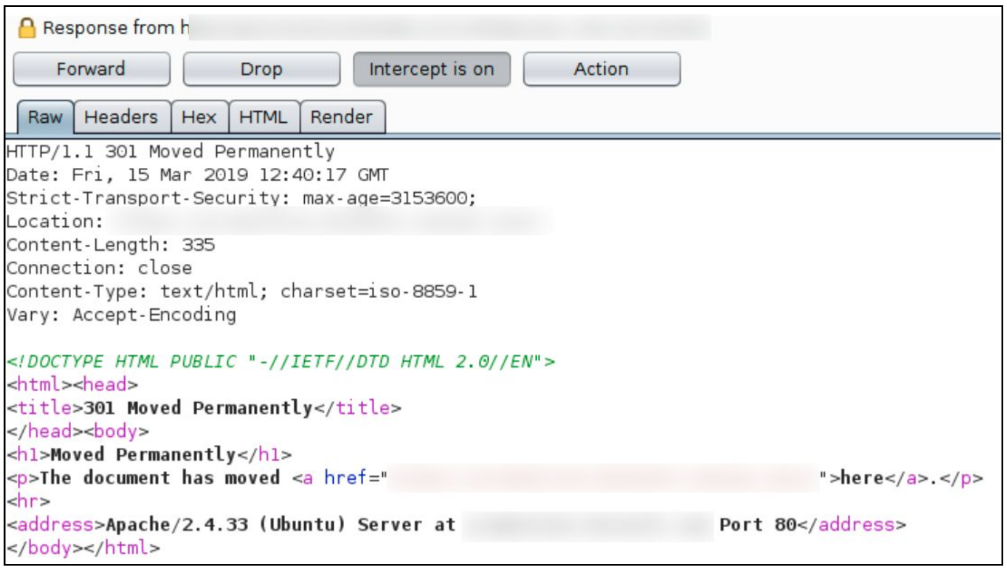
Request parameters

```
["1",{"1":"0"}]
```

Response

```
{"code":"E_VALIDATION","errors":[{"code":"Cannot deserialize the current JSON array
(e.g. [1,2,3]) into type
'xxx.Customers.Registration.WebApi.Models.Requests.MgaRegistrationRequest' because the
type requires a JSON object (e.g. {\"name\":\"value\"}) to deserialize correctly.\r\nTo
fix this error either change the JSON to a JSON object (e.g. {\"name\":\"value\"}) or
change the deserialized type to an array or a type that implements a collection
interface (e.g. ICollection, IList) like List<T> that can be deserialized from a JSON
array. JsonArrayAttribute can also be added to the type to force it to deserialize from
a JSON array.\r\nPath '', line 1, position 1.","field":"request"}]}
```

When reaching https://subdomain.example6.com/wp-json we noticed server version in response.



```
HTTP/1.1 301 Moved Permanently
Date: Fri, 15 Mar 2019 12:40:17 GMT
Strict-Transport-Security: max-age=3153600;
Location: [redacted]
Content-Length: 335
Connection: close
Content-Type: text/html; charset=iso-8859-1
Vary: Accept-Encoding

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="[redacted]">here</a>.</p>
<hr>
<address>Apache/2.4.33 (Ubuntu) Server at [redacted] Port 80</address>
</body></html>
```

A lot of links on https://subdomain.example6.com/wp-json/ are returning JSON data.

```
HTTP/1.1 200 OK
Date: Fri, 15 Mar 2019 12:41:23 GMT
Strict-Transport-Security: max-age=3153600;
X-Robots-Tag: noindex
Link: <https://subdomain.example6.com/wp-json/>; rel="https://api.w.org/"
X-Content-Type-Options: nosniff
Access-Control-Expose-Headers: X-WP-Total, X-WP-TotalPages
Access-Control-Allow-Headers: Authorization, Content-Type
Allow: GET
Connection: close
Content-Type: application/json; charset=UTF-8
Vary: Accept-Encoding
Content-Length: 52834


{"name":"subdomain","description":"Description","url":"https:\/\/subdomain.example6.com"
,"home":"https:\/\/subdomain.example6.com","gmt_offset":"0","timezone_string":"","namesp
aces":["oembed\/1.0","wp\/v2"],"authentication":[],"routes":{"\/":{"namespace":"","metho
ds":["GET"],"endpoints":[{"methods":["GET"],"args":{"context":{"required":false,"default
":"view"}}}],"_links":{"self":"https:\/\/subdomain.example6.com\/wp-json\/"}},"\/oembed\
/1.0":{"namespace":"oembed\/1.0","methods":["GET"],"endpoints":[{"methods":["GET"],"args
":{"namespace":{"required":false,"default":"oembed\/1.0"},"context":{"required":false,"d
efault":"view"}}}],"_links":{"self":"https:\/\/subdomain.example6.com\/wp-json\/oembed\/
1.0"}},"\/oembed\/1.0\/embed":{"namespace":"oembed\/1.0","methods":["GET"],"endpoints":[
{"me
...
```

RECOMMENDATIONS:

We recommend to configure web server to respond with generic error message, so that web server will not reveal any sensitive information. For more details visit link below: https://www.tecmint.com/hide-nginx-server-version-in-linux/