

UnderDefense

Application Security Audit for Client

Compliance with OWASP ASVS L1: **Failed**

June 15, 2017

Notice

UnderDefense has made every reasonable attempt to ensure that the information contained within this report is correct, current and properly sets forth the findings as have been determined to date. The parties acknowledge and agree that the other party assumes no responsibility for errors that may be contained in or for misinterpretations that readers may infer from this document.

Inside this report

Executive summary	3
Summary of business risks	4
Findings overview	6
Findings for Client SaaS application	7
Findings for Client server	22
Appendix A: Covered test cases according to OWASP ASVS Level 1	28

Executive summary

This report presents the results of the security assessment for Client enrollment applications conducted as a part of product excellence and certification process. This assessment was performed under the auspices of two certified and licensed penetration testers employed by UnderDefense during June 1–15, 2017.

Results overview

The test uncovered a few vulnerabilities that may cause compromise user data, application settings and user settings modifications, information disclosure, or reputational damage for company. During penetration testing, UnderDefense security experts found 3 high risk, 14 medium risk vulnerabilities, and 4 low severity issues.

The "Detailed Findings" section in each finding aimed at helping system/application owners to recreate the findings by following the steps mentioned in the section.

Scope

Organization	Client
Application	Client SaaS
Audit type	OWASP Top 10 ASVS L1 and Manual Penetration Testing
Asset URL	https://client.com
Audit period	June 1–15, 2017

Contact details

Reviewed by	John Smith
Prepared by	John Smith, Dow Johns

Security tools used for ASVS Level 1

- Burp Suite Pro [Commercial Edition]
- Tenable Nessus [Commercial Edition]
- Acunetix 9 [Commercial Edition]
- Metasploit Pro [Commercial Edition]
- OWASP Mantra
- OWASP Zap
- Nmap
- Sqlmap

Project limitations

Testing was conducted against the staging environment only.

Summary of business risks

Using high risk attacks, it is possible for attacker to compromise all users of Client SaaS application. Combination of several medium and low risk vulnerabilities may cause serious damage to the integrity and confidentiality of applications.

High-level recommendation

The application requires final security review according SDLC best practices before the final release, because some important functionality is not fully implemented, and remediation testing is required. It is recommended to use web application firewall to filter application level attacks against the production environment.

Methodology

UnderDefense Application Security Assessment Methodology is grounded on following guides and standards:

- [Pentest Execution Standard](#)
- [SANS: Conducting a Penetration Test on an Organization](#)
- [SANS: Network Application Security Assessment and Ethical Hacking](#)
- [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.

UnderDefense application penetration test includes all the items in the OWASP Top 10 and more. The penetration tester remotely try 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.

OWASP ASVS Level 1 is typically appropriate for applications where low confidence in the correct use of security controls is required, for providing a quick analysis of enterprise applications, or for assisting in developing a prioritized list of security requirements as a part of a multiphase effort. Level 1 controls can be ensured either automatically by tools or simply manually without access to source code. We consider Level 1 the minimum required for all applications. Threats to the application will most likely be from attackers who are using simple and low effort techniques to identify easy-to-find and easy-to-exploit vulnerabilities. This is in contrast to a determined attacker who will spend focused energy to specifically target the application.

If the data processed by your application has high value, you would rarely want to stop at a Level 1 review.

Performed tests

- All set of applicable OWASP Top 10 Security Tests
- All set of applicable SANS 25 Security Threats
- All set of applicable from OWASP ASVS Level 1 (see [Appendix A](#) with a key checklist)

Criteria Label	Status
Safe against popular attacks	Fails criteria
Protects sensitive data during transmission	Meets criteria
Safeguards passwords	Meets criteria
Protects against password guessing	Fails criteria
Secure Forgot Password Implementation	Fails criteria
Insecure configuration settings on servers accessible directly by users	Meets criteria
Sensitive data not to be stored on client	Meets criteria
Sensitive data not hidden in pages	Meets criteria
No sensitive data included in error messages	Fails criteria
Code obfuscation for secrets	N/A
Re-authentication required for sensitive activities	Meets criteria
No sensitive data in requests to external sites	Meets criteria
Webserver service protected against known vulnerabilities	Meets criteria
No sample or test applications	Meets criteria
No sensitive data in source code	N/A

Findings overview

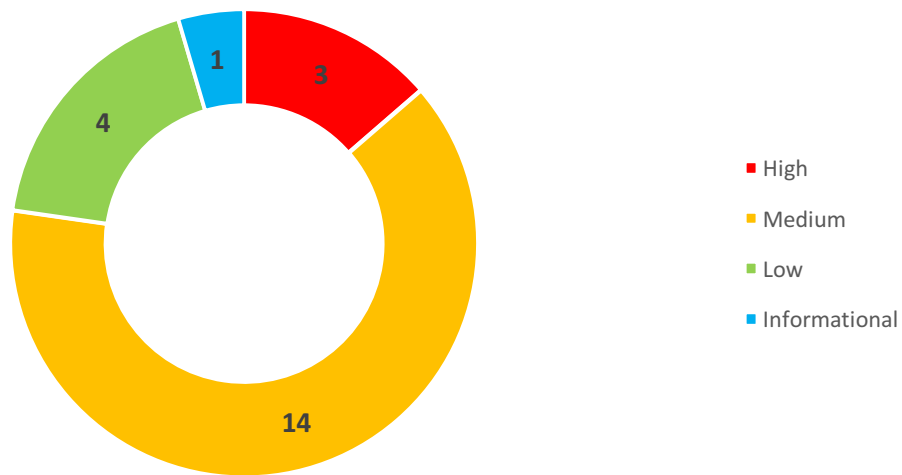
UnderDefense security experts performed manual security testing according to [OWASP Web Application Testing Methodology](#), which demonstrate the following results.

Risk level	High risk	Medium risk	Low risk	Informational
# of vulnerabilities	3	14	4	1

Severity

- **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.

Findings by severity



Findings for Client SaaS application

This sections covers details of all findings for Client SaaS application.

Reflected Cross-Site Scripting

Issue severity: High

Business impact: High

Issue description: Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into benign and trusted web sites. 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 inserts input from a user into the output without validating or encoding it.

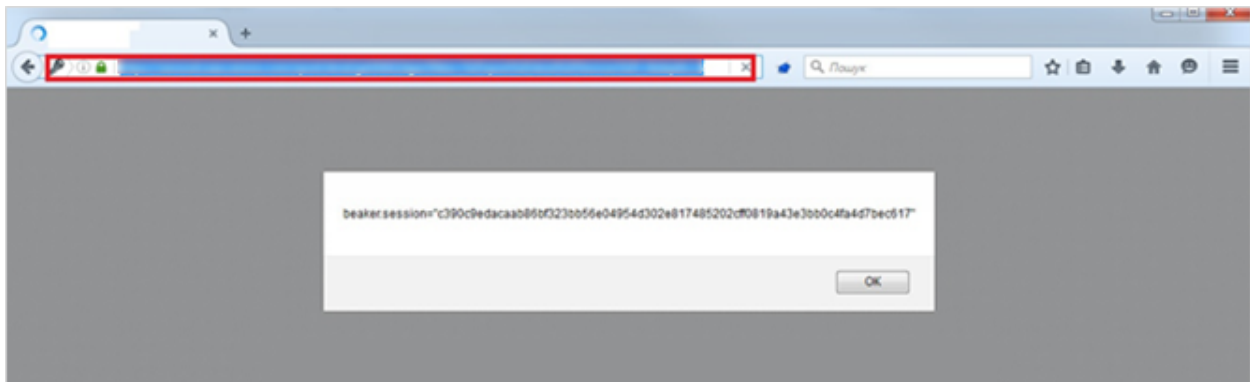
An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens,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

Attacker can craft an URL that will trigger malicious JavaScript payload to steal user session, redirect user to another resource, and so on.

Vulnerable URL:

https://client.com/**/?filter=%2Fzport%2Fdmd%2FDevices%2F%3E&depth=2&objid=192.168%252%22%2F%3E%3Cscript%3Ealert%28document.cookie%29%3C%2Fscript%3E%3Ca%3D%22&submitted=true

Script is successfully triggered:



Recommendations: To filter user input sufficiently, consider [XSS Prevention Cheat Sheet](#).

Stored Cross-Site Scripting

Issue severity: High

Business impact: High

Issue description: Stored attacks are those where the injected script is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, and so on. The victim then retrieves the malicious script from the server when it requests the stored information. Stored XSS is also sometimes referred to as Persistent or Type-I XSS.

Attacker can inject malicious JavaScript code into page (under a Manager role), which will be reflected across all users of the system.

POST /Events/evclasses_router HTTP/1.1

Host: Client.com

Connection: close

Content-Length: 198

Origin: https://Client.com

X-Requested-With: XMLHttpRequest

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)

Chrome/50.0.2661.102 Safari/537.36

Content-Type: application/json

Accept: */*

Referer: https://Client.com/**

Accept-Encoding: gzip, deflate, br

Accept-Language: uk-UA,uk;q=0.8,ru;q=0.6,en-US;q=0.4,en;q=0.2

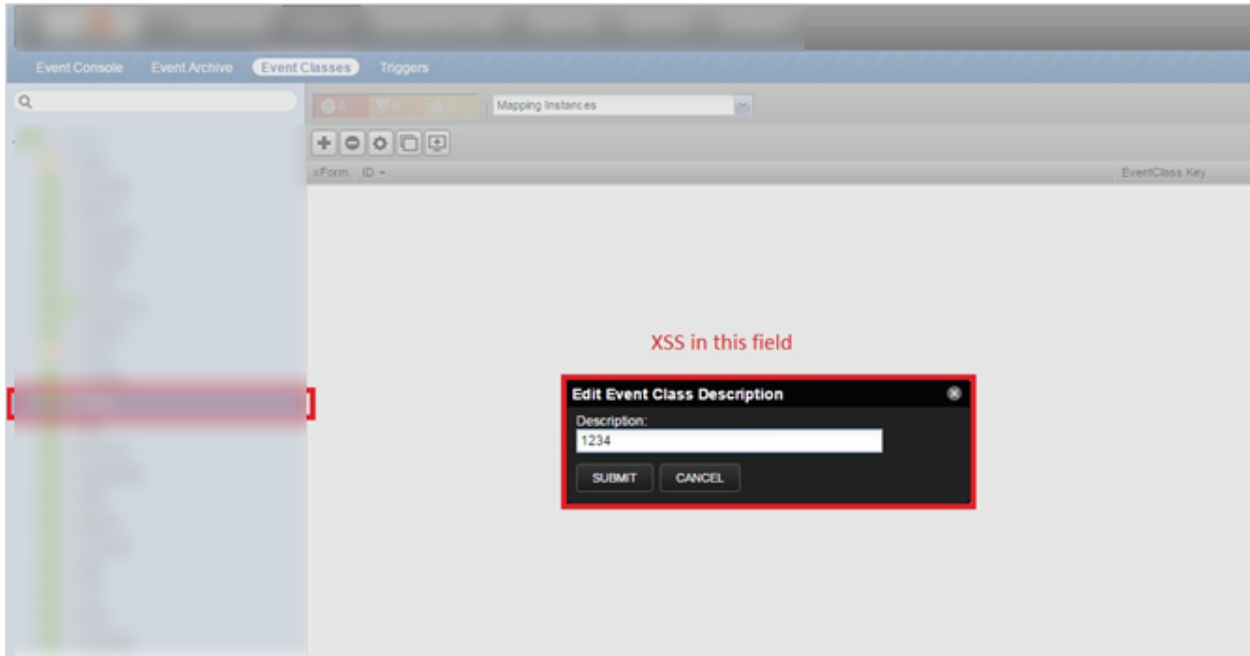
Cookie:

beaker.session="556ab79ec31a6cf70a30a21ff225c2b4805aa19058ad3e3e284e4bb599e07e449076f1aa";

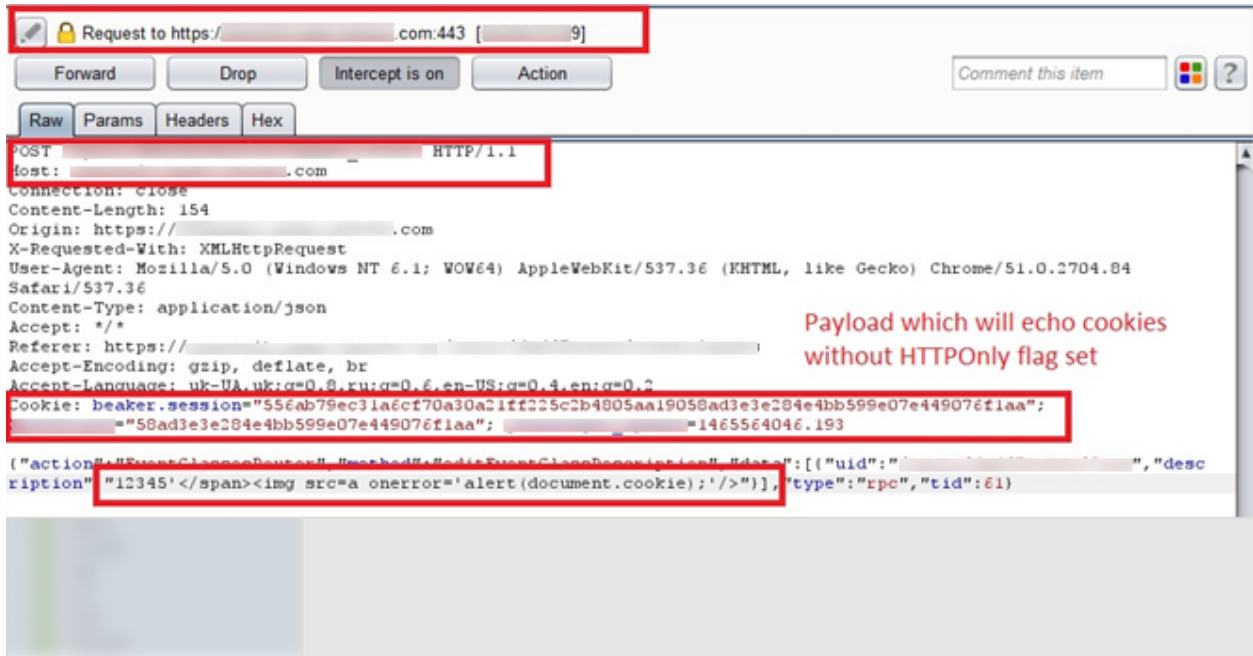
ZAuthToken="58ad3e3e284e4bb599e07e449076f1aa"; ***_update=1465473596.928

```
{"action":"EventClassesRouter","method":"editEventClassDescription","data":{"uid":"/***/License","description":"'12345'</span><img src=a onerror='alert(document.cookie);'/>"},"type":"rpc","tid":101}
```

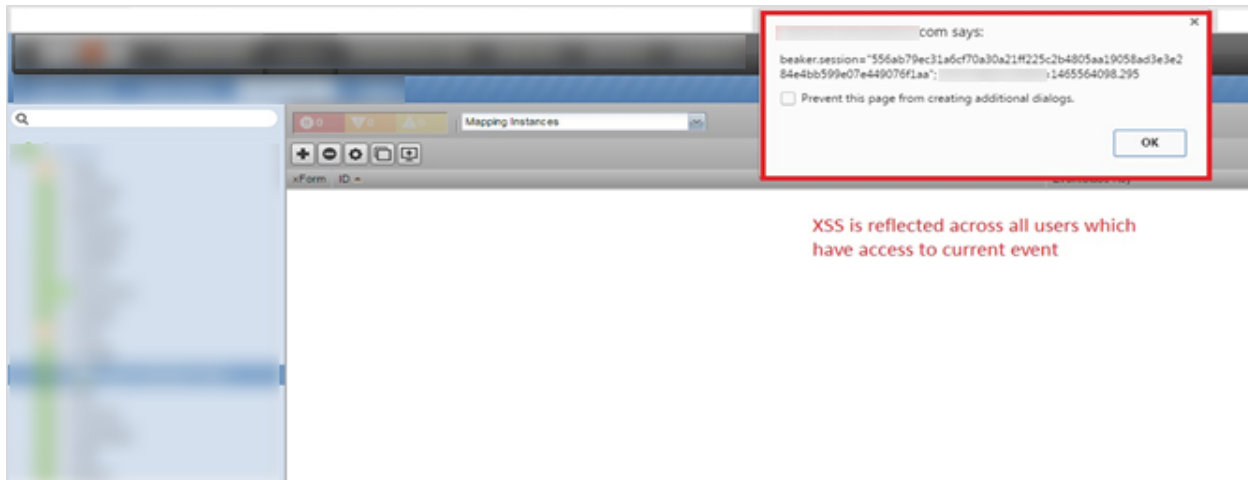

Vulnerable form.



Request with malicious payload.



Payload is triggered across all users.



Recommendations: To filter user input sufficiently, consider [XSS Prevention Cheat Sheet](#) or use framework specific components available.

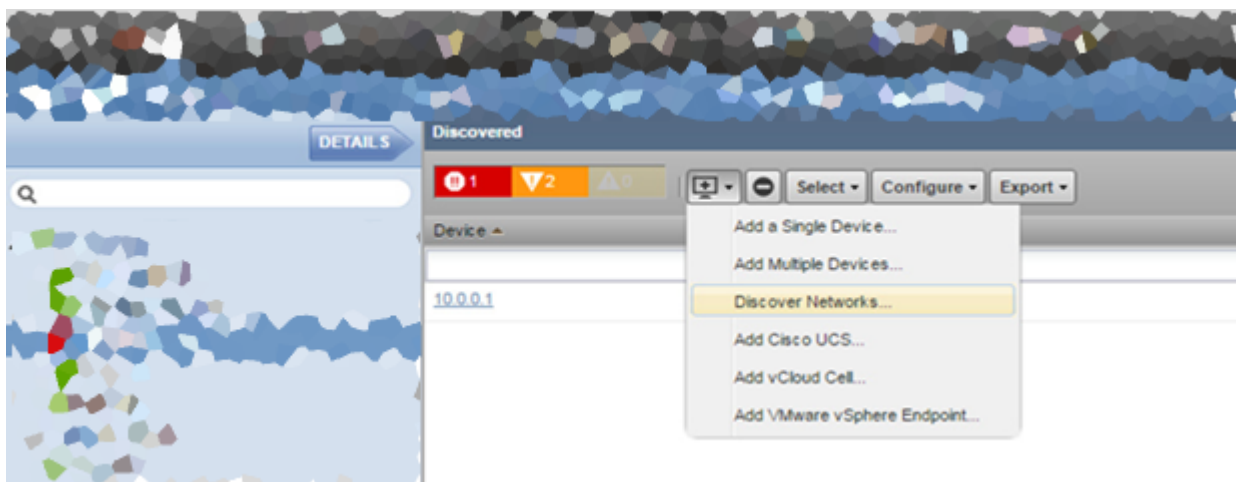
DOM-based Cross-Site Scripting

Issue severity: High

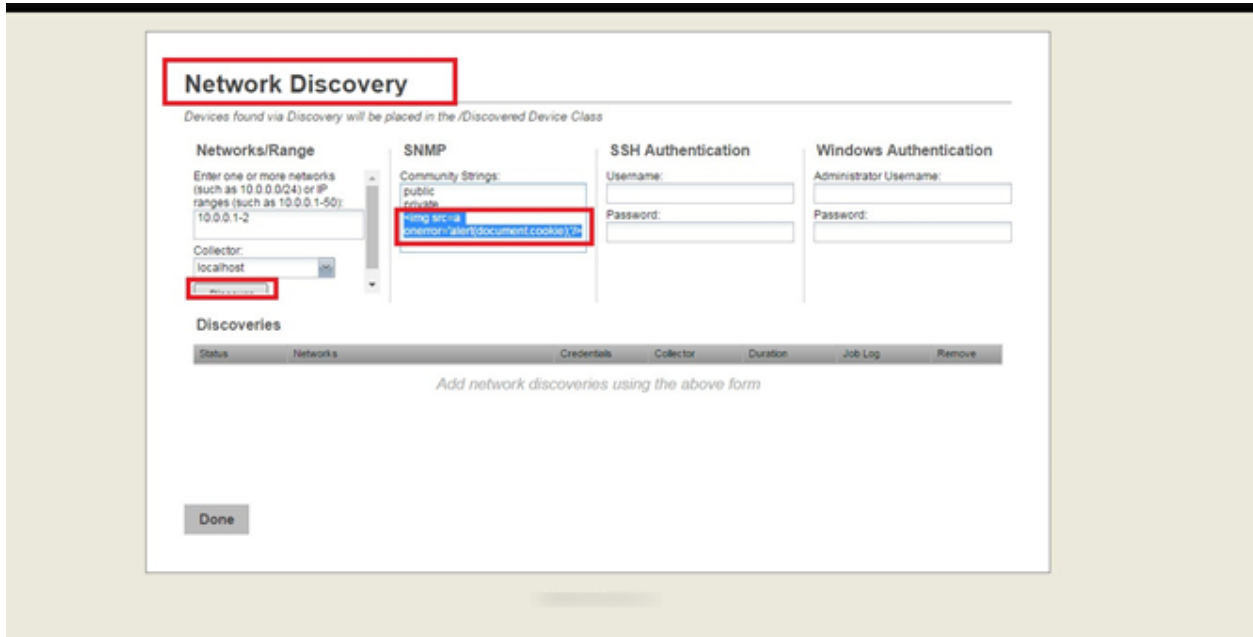
Business impact: High

Issue description: DOM-based XSS (or as it is called in some texts, “type-0 XSS”) is an XSS attack, wherein the attack payload is executed as a result of modifying the DOM “environment” in the victim’s browser used by the original client side script, so that the client side code runs in an “unexpected” manner. That is, the page itself (the HTTP response) does not change, but the client side code, which is contained on the page, is executed differently due to the malicious modifications that have occurred in the DOM environment.

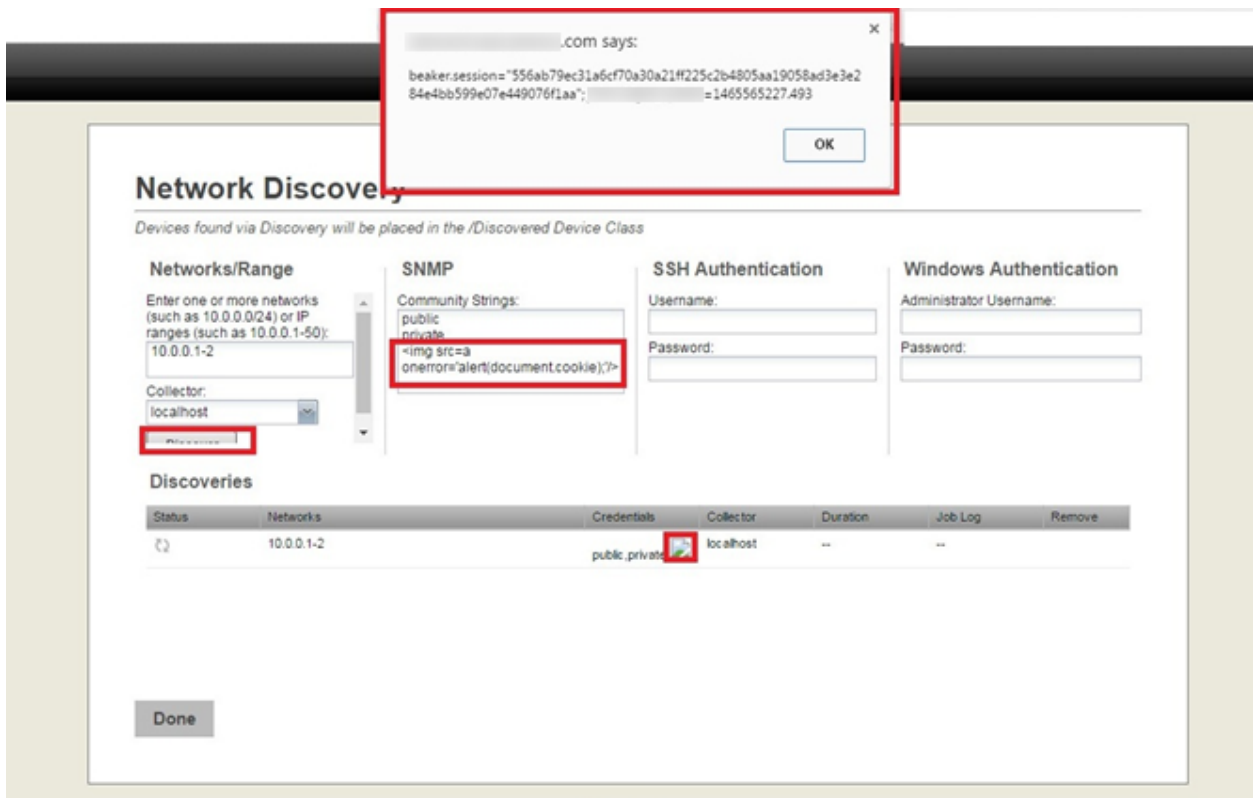
Attacker can inject malicious JavaScript code onto page (under the Manager role) on the **Discover Networks** page.



SNMP field is not filtered properly.



Cookie is echoed successfully.



Recommendations: To filter user input sufficiently, consider [XSS Prevention Cheat Sheet](#).

Insufficient session expiration [CWE-613]

Issue severity: **Medium**

Business impact: **Medium**

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.

Session fixation (WASC-37)

Issue severity: Medium

Business impact: Medium

Issue description: User can use the same session token after logout or password change. Attacker can repeat request with token that should be marked as invalidated.

```
curl -i -s -k -X 'GET' \  
  -H 'User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0' -H 'Referer: \  
https://.Client.com/***?submitted=true' \  
  -b 'j***=1464960334.259; \  
beaker.session="1d1f9a946b96613b622171adeafe6bcfbbe8c4045650fc37ef7243ca9a1801a8be8bfeac"; \  
ZAuthToken="5650fc37ef7243ca9a1801a8be8bfeac"' \  
  'https:// Client.com/***'
```

Recommendations: The logout function should be prominently visible to the user, explicitly invalidate a user's session and disallow reuse of the session token. Server should provide new session id to user browser after logout.

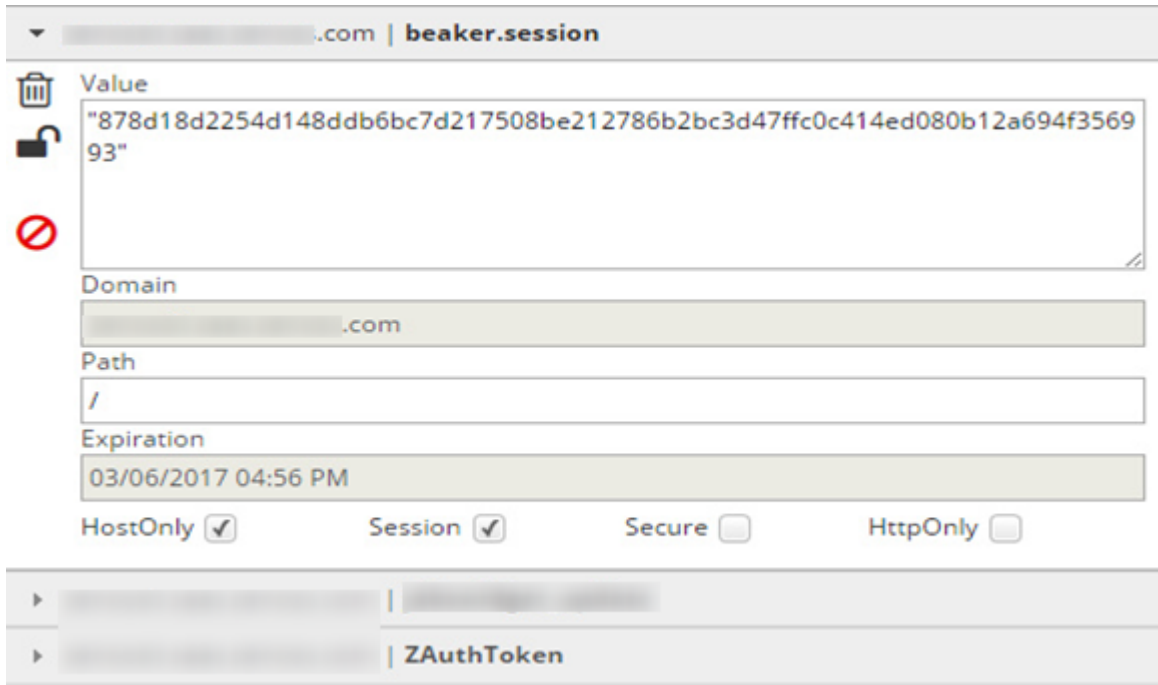
Cookie without Secure flag set

Issue severity: **Medium**

Business impact: **Medium**

Issue description: Session cookie *beaker.session* is set without Secure flag. Secure flag forces browser **not to send** cookie over unsecure channel (use HTTPS instead of HTTP). *Beaker.session* cookie is the most critical and the only one that is required to execute requests to a server. According to our testing, the rest two cookies are optional, and we did not observe any server-side validation for them.

Proof of vulnerability



Recommendations: Ensure that web server sets Secure flag on session cookies.

Verbose error log disclosures information about Client internals

Issue severity: **Low**

Business impact: **Medium**

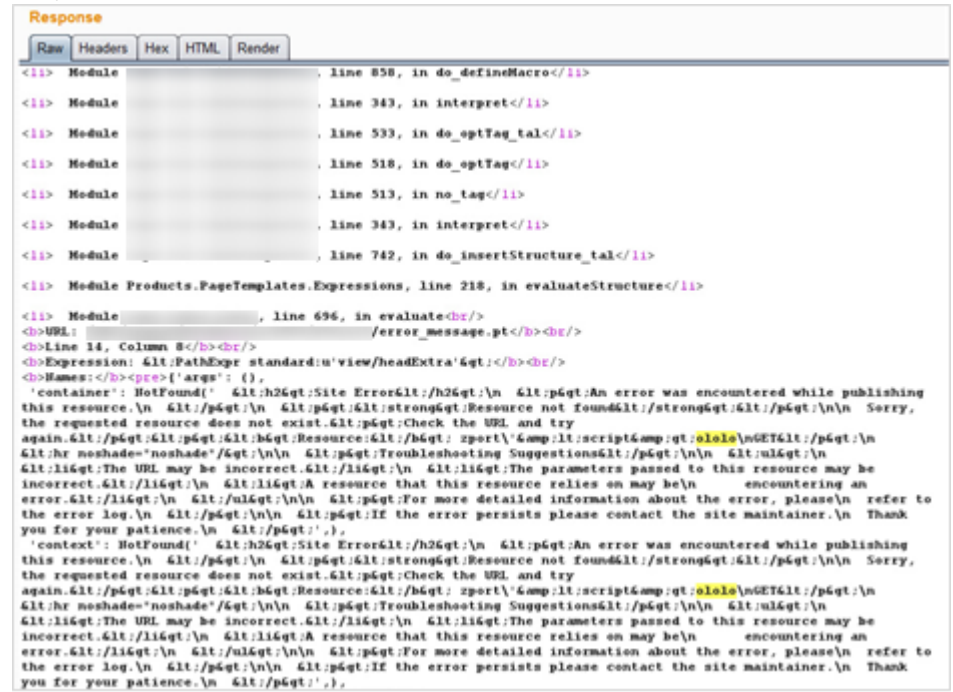
Issue description: Sending special crafted request attacker can get verbose error log, which may reveal useful information, such as software versions, error types, and so on.

Proof of vulnerability

Request:

```
GET /zport HTTP/1.1
Host: Client.com
Connection: close
Origin: https://Client.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/50.0.2661.102 Safari/537.36
Content-Type: application/json
Accept: */*
Referer: https://Client.com/***/Dashboard
Accept-Encoding: gzip, deflate, br
Accept-Language: uk-UA,uk;q=0.8,ru;q=0.6,en-US;q=0.4,en;q=0.2
Cookie:
beaker.session="556ab79ec31a6cf70a30a21ff225c2b4805aa19058ad3e3e284e4bb599e07e449076f1aa";
ZAuthToken="58ad3e3e284e4bb599e07e449076f1aa"; ***_update=1465473596.928
```

Response with verbose error:



```
Response
Raw Headers Hex HTML Render
<!-- Module ... line 858, in do_defineMacro</!-->
<!-- Module ... line 343, in interpret</!-->
<!-- Module ... line 533, in do_optTag_tal</!-->
<!-- Module ... line 518, in do_optTag</!-->
<!-- Module ... line 513, in no_tag</!-->
<!-- Module ... line 343, in interpret</!-->
<!-- Module ... line 742, in do_insertStructure_tal</!-->
<!-- Module Products.PageTemplates.Expressions, line 218, in evaluateStructure</!-->
<!-- Module ... line 696, in evaluate<br/>
<!--URL: .../error_message.pt</br><br/>
<!--Line 14, Column 8</br><br/>
<!--Expression: <!--PathExpr standard:'view/headExtra'&lt;!--</br><br/>
<!--Names: </br><pre>{'args': {}},
'container': 'NotFound' &lt;!--h2&lt;!-- Site Error&lt;!--/h2&lt;!--&lt;!--p&lt;!--An error was encountered while publishing
this resource.&lt;!--&lt;!--p&lt;!--&lt;!--p&lt;!--&lt;!--strong&lt;!--Resource not found&lt;!--/strong&lt;!--&lt;!--p&lt;!--&lt;!--p&lt;!-- Sorry,
the requested resource does not exist.&lt;!--p&lt;!--Check the URL and try
again.&lt;!--p&lt;!--&lt;!--p&lt;!--&lt;!--b&lt;!--Resource:&lt;!--/b&lt;!--&lt;!--p&lt;!-- zport\&lt;!--amp;lt;!--script&lt;!--amp;lt;!--qt:oloisnGET&lt;!--/p&lt;!--&lt;!--
&lt;!--hr noshade="noshade"&lt;!--&lt;!--&lt;!--p&lt;!--Troubleshooting Suggestions&lt;!--/p&lt;!--&lt;!--&lt;!--p&lt;!--ul&lt;!--&lt;!--
&lt;!--li&lt;!--The URL may be incorrect.&lt;!--/li&lt;!--&lt;!--li&lt;!--The parameters passed to this resource may be
incorrect.&lt;!--/li&lt;!--&lt;!--li&lt;!--A resource that this resource relies on may be&lt;!-- encountering an
error.&lt;!--/li&lt;!--&lt;!--p&lt;!--&lt;!--p&lt;!--For more detailed information about the error, please&lt;!-- refer to
the error log.&lt;!--&lt;!--p&lt;!--&lt;!--p&lt;!--If the error persists please contact the site maintainer.&lt;!-- Thank
you for your patience.&lt;!--&lt;!--p&lt;!--,},
'context': 'NotFound' &lt;!--h2&lt;!-- Site Error&lt;!--/h2&lt;!--&lt;!--p&lt;!--An error was encountered while publishing
this resource.&lt;!--&lt;!--p&lt;!--&lt;!--p&lt;!--&lt;!--strong&lt;!--Resource not found&lt;!--/strong&lt;!--&lt;!--p&lt;!--&lt;!--p&lt;!-- Sorry,
the requested resource does not exist.&lt;!--p&lt;!--Check the URL and try
again.&lt;!--p&lt;!--&lt;!--p&lt;!--&lt;!--b&lt;!--Resource:&lt;!--/b&lt;!--&lt;!--p&lt;!-- zport\&lt;!--amp;lt;!--script&lt;!--amp;lt;!--qt:oloisnGET&lt;!--/p&lt;!--&lt;!--
&lt;!--hr noshade="noshade"&lt;!--&lt;!--&lt;!--p&lt;!--Troubleshooting Suggestions&lt;!--/p&lt;!--&lt;!--&lt;!--p&lt;!--ul&lt;!--&lt;!--
&lt;!--li&lt;!--The URL may be incorrect.&lt;!--/li&lt;!--&lt;!--li&lt;!--The parameters passed to this resource may be
incorrect.&lt;!--/li&lt;!--&lt;!--li&lt;!--A resource that this resource relies on may be&lt;!-- encountering an
error.&lt;!--/li&lt;!--&lt;!--p&lt;!--&lt;!--p&lt;!--For more detailed information about the error, please&lt;!-- refer to
the error log.&lt;!--&lt;!--p&lt;!--&lt;!--p&lt;!--If the error persists please contact the site maintainer.&lt;!-- Thank
you for your patience.&lt;!--&lt;!--p&lt;!--,},
```

Recommendations: Ensure that server does not reveal any useful information in any form, even as a debug info in error logs.

Open-redirect vulnerability

Issue severity: **Medium**

Business impact: **Medium**

Issue description: An open redirect is an application that takes a parameter and redirects a user to the parameter value without any validation. This vulnerability is used in phishing attacks to get users to visit malicious sites without realizing it.

Proof of vulnerability

Request:

```
POST http://google.com HTTP/1.1
Host: Client.com
Connection: close
Content-Length: 0
Origin: https://Client.com
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/**.102 Safari/537.36
Content-Type: application/json
Accept: */*
Referer: https://Client.com/***/devices/10.*/***/detail
Accept-Encoding: gzip, deflate, br
Accept-Language: uk-UA,uk;q=0.8,ru;q=0.6,en-US;q=0.4,en;q=0.2
Cookie:
beaker.session="878d18d2254d148ddb6bc7d217508be212786b2bc3d47ffc0c414ed080b12a694f356993";*
*_update=1465199235.61; ***/UserId=oeu1465209245134r0.752719618090913;
***/ments=%7B%222299272282%22%3A%22false%22%2C%222299580245%22%3A%22direct%22%2C%222
305520179%22%3A%22gc%22%7D; ***/uckets=%7B%7D; _ga=GA***;
ZAuthToken="c3d47ffc0c414ed080b12a694f356993"
```

Response with a redirect to another website:

```
HTTP/1.1 301 Moved Permanently
Location: http://google.com/
Date: Wed, 08 Jun 2016 09:24:47 GMT
Content-Length: 0
Content-Type: text/plain; charset=utf-8
Connection: close
```

After such a request, browser will be successfully redirected to an arbitrary website.

Recommendations: Ensure that server does not redirect client to untrusted domains.

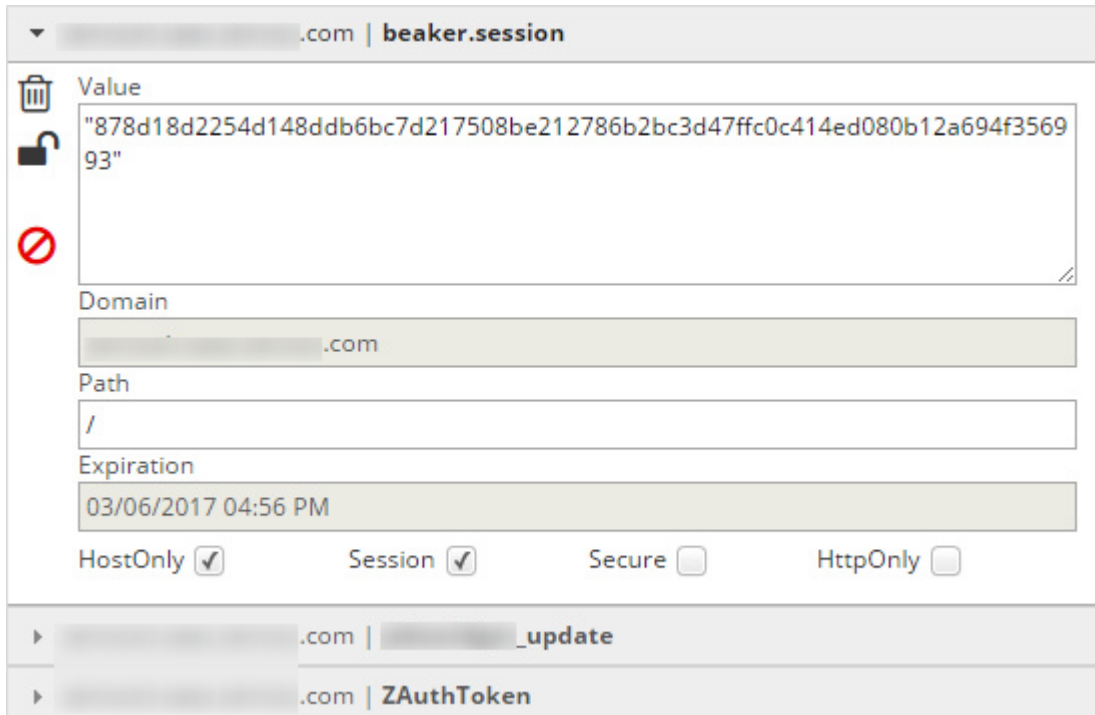
Cookie without HTTPOnly flag set

Issue severity: **Medium**

Business impact: **Medium**

Issue description: Session cookie *beaker.session* is set without HTTPOnly flag. This flag ensures that an attacker cannot steal cookie with Javascript on a client side.

Proof of vulnerability



Recommendations: Ensure that web server sets HTTPOnly flag on session cookies.

Password bruteforce is possible

Issue severity: **Medium**

Business impact: **Medium**

Issue description: https://Client.com/zport/***/login

Because application does not block a user after a few failed login attempts, it is possible to enumerate passwords using the login form. Attacker can harvest user credentials and have unauthorized access to application functionality and confidential data.

Proof of vulnerability: Application does not check the quantity of failed requests and lets user in upon a successful one. Attacker can automate this attack and perform password bruteforcing using this request.

Vulnerable request:

POST /**/login HTTP/1.1

Host: Client.com

Connection: close

Content-Length: 131

Cache-Control: max-age=0

Origin: https://Client.com

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.84

Content-Type: application/x-www-form-urlencoded

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

Referer: https://Client.com/**/login_form?came_from=https%3A//Client.com/**/

Accept-Encoding: gzip, deflate, br

Accept-Language: uk-UA,uk;q=0.8,ru;q=0.6,en-US;q=0.4,en;q=0.2

Cookie: ***_update=1465906625.949

came_from=https%3A%2F%2FClient.com%2F***%2F***%2F&submitted=true&fragment=&__ac_name=pentest02&

Recommendations: Make sure that username is blocked for some time after several failed logins. Block IP address after several same requests with different values. Enable captcha.

Exponentially increase the amount of time a user has to wait between authentication attempts until it reaches a rate that makes brute-forcing impractical (for example, 24 hours).

Explanation: (Common Weaknesses Enumeration ID: 307
<http://cwe.mitre.org/data/definitions/307>)

HTML form without CSRF protection

Issue severity: **Medium**

Business impact: **Medium**

Issue description: Cross-site request forgery, also known as a one-click attack or session riding (abbreviated as CSRF or XSRF), is a type of malicious exploit of a website, whereby unauthorized commands are transmitted from a user that the website trusts.

The impact of this vulnerability: An attacker may force the users of a web application to execute actions of the attacker's choosing. A successful CSRF exploit can compromise end user data and operation in case of normal user. If the targeted end user is the administrator account, this can compromise the entire web application.

Proof of vulnerability: "Change email" request can be triggered without anti-CSRF token. An attacker can trick user to successfully perform this request.

POST /**/pentest07 HTTP/1.1

Host: Client.com

Connection: close

Content-Length: 224

Cache-Control: max-age=0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8

Origin: https://Client.com

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)

Chrome/50.0.2661.102 Safari/537.36

Content-Type: application/x-www-form-urlencoded

Referer: https://Client.com/***/pentest07

Accept-Encoding: gzip, deflate, br

Accept-Language: uk-UA,uk;q=0.8,ru;q=0.6,en-US;q=0.4,en;q=0.2

Cookie:

beaker.session="878d18d2254d148ddb6bc7d217508be212786b2bc3d47ffc0c414ed080b12a694f356993"

```
***=-editUserSettings.pt&email=***%40inc.coma&pager=&defaultPageSize=40&net***=&timezone=America%2FChicago&password=&sendpassword=&oldpassword=&***_editUserSettings%3Amethod=+Save+Settings+
```

Recommendations: Check if this form requires CSRF protection and implement CSRF countermeasures if necessary.

An anti-CSRF token is a session-specific or even transaction-specific random string appended as a parameter to important transactions. Upon handling the client's request, the server ensures that the CSRF token is the value expected for that session/transaction. If the token is not correct, then the application denies the transaction. This helps protect against CSRF because each request will have at least one unique parameter that an attacker cannot know ahead of time.

Note that you may be able to mitigate the risk of CSRF by using an alternative user-specific token, such as the userid, rather than a specific anti-CSRF token.

When a web server is designed to receive a request from a client without any mechanism for verifying that it was intentionally sent, then it might be possible for an attacker to trick a client into making an unintentional request to the web server which will be treated as an authentic request. This can be done via a URL, image load, XMLHttpRequest, and others, and can result in data disclosure or unintended code execution (Common Weaknesses Enumeration ID: 352 - <http://cwe.mitre.org/data/definitions/352>).

Username enumeration

Issue severity: **Medium**

Business impact: **Medium**

Issue description: “Forgot password” functionality response identifies if a username is already registered. Attacker can launch bruteforce or dictionary attack to harvest usernames of clients.

The application should not leak any information—regarding the validity of the username, any suspension of the account, and so on—in the event of failed responses to the challenge.

Recommendations: Provide less verbose responses in the “Forgot password” functionality. Make sure that security question value is checked properly. Block IP address after several same requests with different values. Enable captcha.

No clickjacking protection

Issue severity: **Low**

Business impact: **Low**

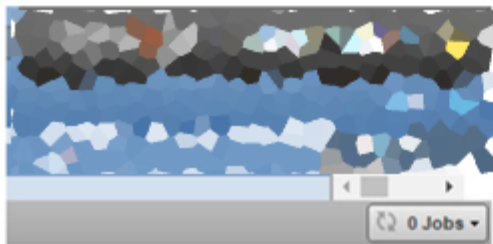
Issue description: Clickjacking, also known as a "UI redress attack", is when an attacker uses multiple transparent or opaque layers to trick a user into clicking a button or a link on another page when they were intending to click the top-level page. Thus, the attacker is "hijacking" clicks meant for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

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:

Framed page example:

Win in One Click!!!



Code snippet:

```
<body>
  <h1 align="center">Win in One Click!!!</h1>
  <div id="wrap">
    <iframe id="clickjacking" src="https://[redacted]/contest07" scrolling="no" frameborder="none" >
  </div>
</body>
```

Recommendations: There are two main ways to prevent clickjacking:

- Sending the proper X-Frame-Options HTTP response headers that instruct the browser to not allow framing from other domains.
- Employing defensive code in the UI to ensure that the current frame is the most top-level window.

References:

- <https://www.owasp.org/index.php/Clickjacking>
- https://www.owasp.org/index.php/Clickjacking_Defense_Cheat_Sheet

Lack of Content-Security-Policy

Issue severity: **Low**

Business impact: **Low**

Issue description: The new Content-Security-Policy HTTP response header helps you reduce XSS risks on modern browsers by declaring what dynamic resources are allowed to load via a HTTP Header.

Recommendations: Add Content-Security-Policy support to target application.

References:

- https://www.owasp.org/index.php/List_of_useful_HTTP_headers
- <http://content-security-policy.com/>

Lack of X-XSS-Protection

Issue severity: **Low**

Business impact: **Low**

Issue description: To improve the security of your site against some types of cross-site scripting (XSS) attacks, it is recommended that you add the following header to your site:

```
X-XSS-Protection: 1; mode=block
```

Recommendations: Add X-XSS-Protection header to the target application.

Reference: https://www.owasp.org/index.php/List_of_useful_HTTP_headers

Findings for Client server

No brute-force protection

Issue severity: **Medium**

Business impact: **Medium**

Issue description: <https://Client.com/#/login>

Application allows an attacker to brute-force passwords against Control Center application. An account or attacker IP address is not blocked for some period of time. More advanced solution to stop brute-force attacks is to use captcha. It should be generated in case of brute-force after 5 unsuccessful login attempts.

Proof of vulnerability

The screenshot displays two windows side-by-side. The left window is titled 'Intruder attack 1' and shows a table of HTTP requests. The right window is a browser showing the 'Control Center' login page with the username 'pentest01' and a password field. An error message at the bottom of the browser reads 'Error Username/Password is invalid'.

Request	Payload	Status	Error	Timeout	Length	Comment
724		401			296	
725		401			296	
726		401			296	
727		401			296	
728		401			296	
729		401			296	
730		401			296	
731		401			296	
732		401			296	
733		401			296	
734		401			296	
735		401			296	
736		401			296	

```
HTTP/1.1 401 Unauthorized
Content-Type: application/json
Vary: Accept-Encoding
Date: Fri, 03 Jun 2016 13:14:13 GMT
Content-Length: 135
Connection: close

{"Detail":"Login failed","Links":[{"Name":"Create","Method":"POST","Uri":"/login"}, {"Name":"Login","Method":"POST","Uri":"/login"}]}
```

Recommendations: Enable captcha for blocking brute-force. This will ensure that the request will fail during automated attacks.

Using components with known vulnerabilities

Issue severity: **Medium**

Business impact: **Medium**

Issue description: <https://Client.com/#/login>

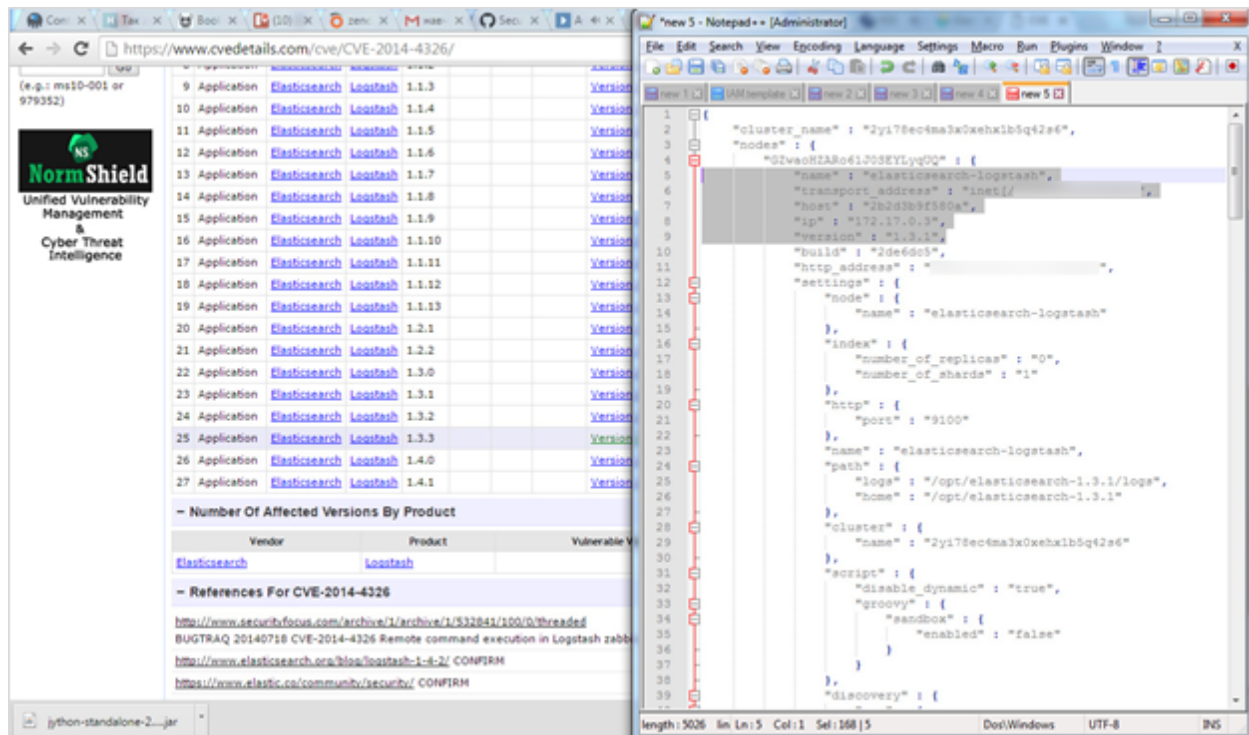
CWE-937: OWASP Top Ten 2013 Category A9:

1. Vulnerability Details: [CVE-2014-4326](#)
Logstash 1.4.2 and prior versions are vulnerable to a directory traversal attack that allows an attacker to overwrite files on the server running Logstash.
2. Vulnerability Details: [CVE-2015-4152](#)
Elasticsearch Logstash 1.0.14 through 1.4.x before 1.4.2 allows remote attackers to execute arbitrary commands via a crafted event in (1) zabbix.rb or (2) nagios_nsca.rb in outputs.

<https://packetstormsecurity.com/files/132233/Logstash-1.4.2-Directory-Traversal.html>

CVSS score	7.5
Confidentiality impact	Partial (There is a considerable informational disclosure.)
Integrity impact	Partial (Modification of some system files or information is possible, but the attacker does not have control over what can be modified, or the scope of what the attacker can affect is limited.)
Availability impact	Partial (There is reduced performance or interruptions in resource availability.)
Access complexity	Low (Specialized access conditions or extenuating circumstances do not exist. Very little knowledge or skill is required to exploit.)
Authentication	Not required (Authentication is not required to exploit the vulnerability.)
Gained access	None
Vulnerability type(s)	Execute code

Proof of vulnerability



The screenshot shows a web browser displaying the CVE-2014-4326 details page. The page lists affected versions of Elasticsearch Logstash from 1.1.3 to 1.4.1. A table below the list shows the number of affected versions by product, with Logstash having 27 affected versions. Below the table, there are references for the CVE, including a security focus article and a remote command execution exploit.

The Notepad++ window shows the configuration file for the Elasticsearch cluster. The configuration includes the cluster name, node name, transport address, host, IP, version, build, http address, settings, node name, index settings, http settings, path, logs, home, cluster name, script settings, and discovery settings.



The screenshot shows a network traffic capture in Wireshark. The packet is a GET request to the Elasticsearch API. The request includes headers such as User-Agent, Referer, and token. The body of the request is empty.

```
curl -i -s -k -X 'GET' \  
-H 'User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:18.0) Gecko/20100101 Firefox/18.0' -H \  
'Referer: https://Client.com/static/logview/' \  
-b 'token=ZSQw9+6f6lZlVxi5XWlOnSyP6qyOuTN62lKVbvK3qJw; username=pentest' \  
'https://Client.com/api/**/elastic/**'
```

Recommendations: Users that currently use the file output plugin or may use it in the future should upgrade to 1.5.0 or 1.4.3. This will address the vulnerability and preserve file output functionality.

Users that do not want to upgrade can address the vulnerability by disabling the file output plugin.

Information leakage

Issue severity: [Info](#)

Business impact: [Info](#)

Issue description: <https://Client.com/#/login>

An information leak is the intentional or unintentional disclosure of information that either (1) is regarded as sensitive within the product's own functionality, such as a private message, or (2) provides information about the product or its environment that could be useful in an attack but is normally not available to the attacker, such as the installation path of a product that is remotely accessible.

Many information leaks are resultant (for example, path disclosure in PHP script error), but they can also be primary (for example, timing discrepancies in crypto). There are many different types of problems that involve information leaks. Their severity can range widely depending on the type of information that is leaked.

Proof of vulnerability

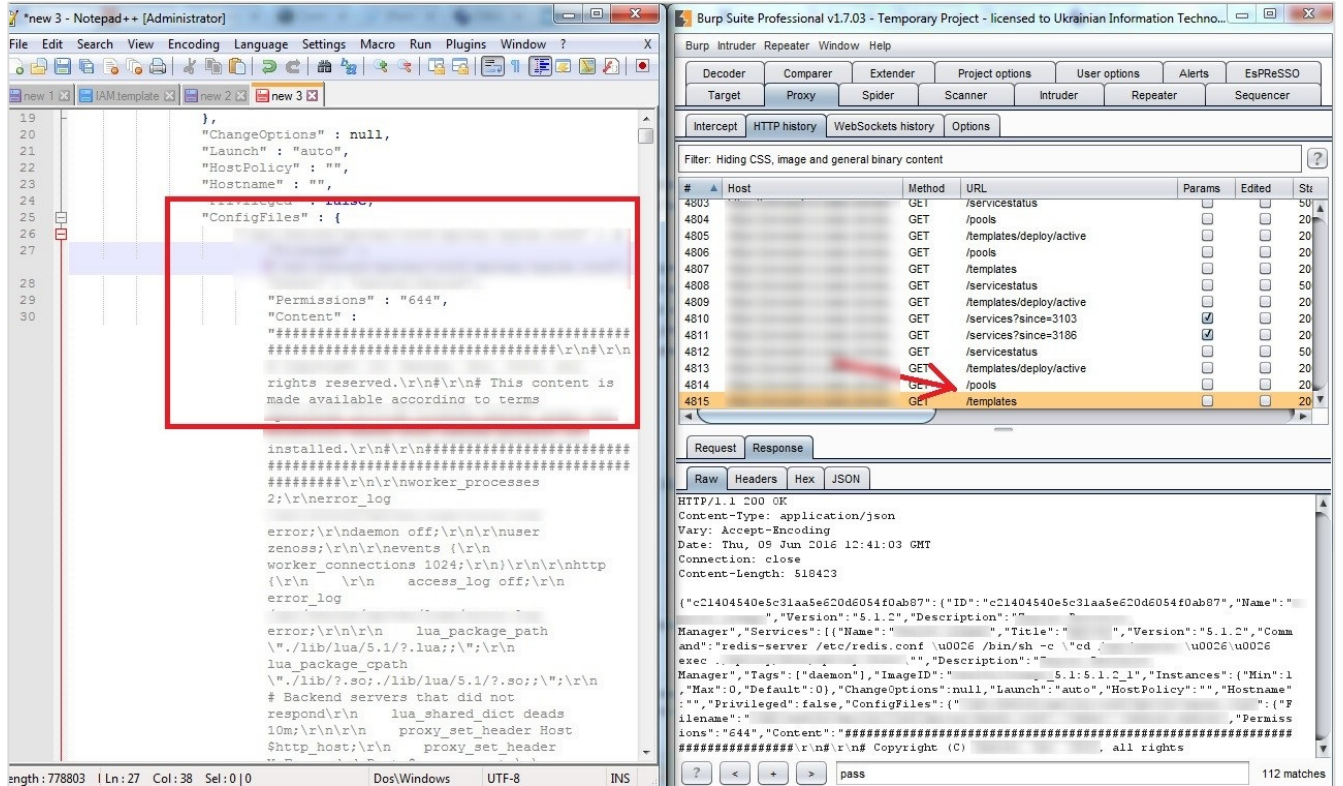
Responses with sensitive info in template.

The image shows two windows side-by-side. The left window is Notepad++ displaying a JavaScript file with a JSON object containing a 'Context' field. The right window is Burp Suite Professional v1.7.03, showing a list of HTTP requests and their responses. The selected request (4815) is a GET request to '/templates'. The response is a JSON object containing sensitive information, including a 'Manager' field with a 'Services' array and a 'Content' field with a long string of characters.

#	Host	Method	URL	Params	Edited	Str
4803		GET	/servicestatus		<input type="checkbox"/>	50
4804		GET	/pools		<input type="checkbox"/>	20
4805		GET	/templates/deploy/active		<input type="checkbox"/>	20
4806		GET	/pools		<input type="checkbox"/>	20
4807		GET	/templates		<input type="checkbox"/>	20
4808		GET	/servicestatus		<input type="checkbox"/>	50
4809		GET	/templates/deploy/active		<input type="checkbox"/>	20
4810		GET	/services?since=3103	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
4811		GET	/services?since=3186	<input checked="" type="checkbox"/>	<input type="checkbox"/>	20
4812		GET	/servicestatus		<input type="checkbox"/>	50
4813		GET	/templates/deploy/active		<input type="checkbox"/>	20
4814		GET	/pools		<input type="checkbox"/>	20
4815		GET	/templates		<input type="checkbox"/>	20

```
HTTP/1.1 200 OK
Content-Type: application/json
Vary: Accept-Encoding
Date: Thu, 09 Jun 2016 12:41:03 GMT
Connection: close
Content-Length: 518423

{"ID":"c21404540e5c31aa5e20d6054f0ab87","Name":"Z
","Version":"5.1.2","Description":"
Manager","Services":[{"Name":"","Title":"","Version":"5.1.2","Comm
and":"redis-server /etc/redis.conf \u0026 /bin/sh -c \"cd
exec
Manager","Tags":["daemon"],"ImageID":"
5.1:5.1.2_1","Instances":{"Min":1
,"Max":0,"Default":0,"ChangeOptions":null,"Launch":"auto","HostPolicy":"","Hostname
":"","Privileged":false,"ConfigFiles":{"
filename":"","Owner":"
Permissions":"644","Content":"#####
#####r\nr\n# Copyright (C)
all rights
```



Recommendations: Ensure that templates returned to the client do not contain sensitive information, which may be useful for an attacker.

HSTS missing from HTTPS server

Issue severity: Medium

Business impact: Medium

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.

Appendix A: Covered test cases according to OWASP ASVS Level 1

#	Category	Detail	Level 1
1.1	V1. Architecture, design, and threat modelling	Verify that all the needed application components are identified and are known.	Covered
2.1	V2: Authentication Verification Requirements	Verify that all pages and resources by default require authentication except those specifically intended to be public (principle of complete mediation).	Covered
2.2	V2: Authentication Verification Requirements	Verify that all password fields do not echo the user's password when it is entered.	Covered
2.4	V2: Authentication Verification Requirements	Verify that all authentication controls are enforced on the server side.	Covered
2.6	V2: Authentication Verification Requirements	Verify that all authentication controls fail securely to ensure attackers cannot log in.	Covered
2.7	V2: Authentication Verification Requirements	Verify that password entry fields allow or encourage the use of passphrases, and do not prevent long passphrases/highly complex passwords from being entered.	Covered
2.8	V2: Authentication Verification Requirements	Verify all account identity authentication functions (such as update profile, forgot password, disabled/lost token, help desk or IVR) that might regain access to the account are at least as resistant to attack as the primary authentication mechanism.	Covered
2.9	V2: Authentication Verification Requirements	Verify that the change password functionality includes the old password, the new password, and a password confirmation.	Covered
2.16	V2: Authentication Verification Requirements	Verify that credentials are transported using a suitable encrypted link and that all pages/functions that require a user to enter credentials are done so using an encrypted link.	Covered
2.17	V2: Authentication Verification Requirements	Verify that the forgotten password function and other recovery paths do not reveal the current password and that the new password is not sent in clear text to the user.	Covered
2.18	V2: Authentication Verification Requirements	Verify that information enumeration is not possible via login, password reset, or forgot account functionality.	Covered

#	Category	Detail	Level 1
2.19	V2: Authentication Verification Requirements	Verify that there are no default passwords in use for the application framework or any components used by the application (such as “admin/password”).	Covered
2.20	V2: Authentication Verification Requirements	Verify that request throttling is in place to prevent automated attacks against common authentication attacks such as brute-force attacks or denial of service attacks.	Covered
2.22	V2: Authentication Verification Requirements	Verify that forgotten password and other recovery paths use a soft token, mobile push, or an offline recovery mechanism.	Covered
2.24	V2: Authentication Verification Requirements	Verify that if knowledge-based questions (also known as "secret questions") are required, the questions should be strong enough to protect the application.	Covered
2.27	V2: Authentication Verification Requirements	Verify that measures are in place to block the use of commonly chosen passwords and weak passphrases.	Covered
2.30	V2: Authentication Verification Requirements	Verify that if an application allows users to authenticate, they use a proven secure authentication mechanism.	Covered
2.32	V2: Authentication Verification Requirements	Verify that administrative interfaces are not accessible to untrusted parties.	Covered
3.1	V3: Session Management Verification Requirements	Verify that there is no custom session manager or that a custom session manager is resistant against all common session management attacks.	Covered
3.2	V3: Session Management Verification Requirements	Verify that sessions are invalidated when the user logs out.	Covered
3.3	V3: Session Management Verification Requirements	Verify that sessions time out after a specified period of inactivity.	Covered
3.5	V3: Session Management Verification Requirements	Verify that all pages that require authentication have easy and visible access to logout functionality.	Covered
3.6	V3: Session Management Verification Requirements	Verify that the session id is never disclosed in URLs, error messages, or logs. This includes verifying that the application does not support URL rewriting of session cookies.	Covered
3.7	V3: Session Management Verification Requirements	Verify that all successful authentication and re-authentication generates a new session and session id.	Covered

#	Category	Detail	Level 1
3.12	V3: Session Management Verification Requirements	Verify that session ids stored in cookies have their path set to an appropriately restrictive value for the application, and authentication session tokens additionally set the "HttpOnly" and "secure" attributes.	Covered
3.16	V3: Session Management Verification Requirements	Verify that the application limits the number of active concurrent sessions.	Covered
3.17	V3: Session Management Verification Requirements	Verify that an active session list is displayed in the account profile or similar of each user. The user should be able to terminate any active session.	Covered
3.18	V3: Session Management Verification Requirements	Verify that the user is prompted with the option to terminate all other active sessions after a successful change password process.	Covered
4.1	V4: Access Control Verification Requirements	Verify that the principle of least privilege exists: users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege.	Covered
4.4	V4: Access Control Verification Requirements	Verify that access to sensitive records is protected, such that only authorized objects or data is accessible to each user (for example, protect against users tampering with a parameter to see or alter another user's account).	Covered
4.5	V4: Access Control Verification Requirements	Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS_Store, .git, or .svn folders.	Covered
4.8	V4: Access Control Verification Requirements	Verify that access controls fail securely.	Covered
4.9	V4: Access Control Verification Requirements	Verify that the same access control rules implied by the presentation layer are enforced on the server side.	Covered
4.13	V4: Access Control Verification Requirements	Verify that the application or framework uses strong random anti-CSRF tokens or has another transaction protection mechanism.	Covered
4.16	V4: Access Control Verification Requirements	Verify that the application correctly enforces context-sensitive authorization so as to not allow unauthorized manipulation by means of parameter tampering.	Covered

#	Category	Detail	Level 1
5.1	V5: Malicious input handling verification requirements	Verify that the runtime environment is not susceptible to buffer overflows, or that security controls prevent buffer overflows.	Covered
5.3	V5: Malicious input handling verification requirements	Verify that server-side input validation failures result in request rejection and are logged.	Covered
5.5	V5: Malicious input handling verification requirements	Verify that input validation routines are enforced on the server side.	Covered
5.10	V5: Malicious input handling verification requirements	Verify that all SQL queries, HQL, OSQL, NOSQL, and stored procedures, calling of stored procedures are protected by the use of prepared statements or query parameterization; thus, not susceptible to SQL injection.	Covered
5.11	V5: Malicious input handling verification requirements	Verify that the application is not susceptible to LDAP Injection or that security controls prevent LDAP Injection.	Covered
5.12	V5: Malicious input handling verification requirements	Verify that the application is not susceptible to OS Command Injection or that security controls prevent OS Command Injection.	Covered
5.13	V5: Malicious input handling verification requirements	Verify that the application is not susceptible to Remote File Inclusion (RFI) or Local File Inclusion (LFI) when content is used that is a path to a file.	Covered
5.14	V5: Malicious input handling verification requirements	Verify that the application is not susceptible to common XML attacks, such as XPath query tampering, XML External Entity attacks, and XML injection attacks.	Covered
5.15	V5: Malicious input handling verification requirements	Ensure that all string variables placed into HTML or other web client code is either properly contextually encoded manually, or utilize templates that automatically encode contextually to ensure the application is not susceptible to reflected, stored and DOM Cross-Site Scripting (XSS) attacks.	Covered
5.22	V5: Malicious input handling verification requirements	Make sure untrusted HTML from WYSIWYG editors or similar are properly sanitized with an HTML sanitizer and handle it appropriately according to the input validation task and encoding task.	Covered

#	Category	Detail	Level 1
7.2	V7: Cryptography at rest verification requirements	Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable oracle padding.	Covered
7.7	V7: Cryptography at rest verification requirements	Verify that cryptographic algorithms used by the application have been validated against FIPS 140-2 or an equivalent standard.	Covered
8.1	V8: Error handling and logging verification requirements	Verify that the application does not output error messages or stack traces containing sensitive data that could assist an attacker, including session id, software/framework versions and personal information.	Covered
9.1	V9: Data protection verification requirements	Verify that all forms containing sensitive information have disabled client-side caching, including autocomplete features.	Covered
9.3	V9: Data protection verification requirements	Verify that all sensitive data is sent to the server in the HTTP message body or headers (i.e., URL parameters are never used to send sensitive data).	Covered
9.4	V9: Data protection verification requirements	Verify that the application sets appropriate anti-caching headers as per the risk of the application, such as the following: Expires: Tue, 03 Jul 2001 06:00:00 GMT Last-Modified: {now} GMT Cache-Control: no-store, no-cache, must-revalidate, max-age=0 Cache-Control: post-check=0, pre-check=0 Pragma: no-cache	Covered
9.9	V9: Data protection verification requirements	Verify that data stored in client side storage (such as HTML5 local storage, session storage, IndexedDB, regular cookies or Flash cookies) does not contain sensitive or PII.	Covered
10.1	V10: Communications security verification requirements	Verify that a path can be built from a trusted CA to each Transport Layer Security (TLS) server certificate, and that each server certificate is valid.	Covered
10.3	V10: Communications security verification requirements	Verify that TLS is used for all connections (including both external and backend connections) that are authenticated or that involve sensitive data or functions, and does not fall back to insecure or unencrypted protocols. Ensure the strongest alternative is the preferred algorithm.	Covered

#	Category	Detail	Level 1
10.11	V10: Communications security verification requirements	Verify that HTTP Strict Transport Security headers are included on all requests and for all subdomains, such as Strict-Transport-Security: max-age=15724800; includeSubdomains	Covered
10.13	V10: Communications security verification requirements	Ensure forward secrecy ciphers are in use to mitigate passive attackers recording traffic.	Covered
10.14	V10: Communications security verification requirements	Verify that proper certification revocation, such as Online Certificate Status Protocol (OCSP) Stapling, is enabled and configured.	Covered
10.15	V10: Communications security verification requirements	Verify that only strong algorithms, ciphers, and protocols are used, through all the certificate hierarchy, including root and intermediary certificates of your selected certifying authority.	Covered
10.16	V10: Communications security verification requirements	Verify that the TLS settings are in line with current leading practice, particularly as common configurations, ciphers, and algorithms become insecure.	Covered
11.1	V11: HTTP security configuration verification requirements	Verify that the application accepts only a defined set of required HTTP request methods, such as GET and POST are accepted, and unused methods (for example, TRACE, PUT, and DELETE) are explicitly blocked.	Covered
11.2	V11: HTTP security configuration verification requirements	Verify that every HTTP response contains a content type header specifying a safe character set (for example, UTF-8, ISO 8859-1).	Covered
11.5	V11: HTTP security configuration verification requirements	Verify that the HTTP headers or any part of the HTTP response do not expose detailed version information of system components.	Covered
11.6	V11: HTTP security configuration verification requirements	Verify that all API responses contain X-Content-Type-Options: nosniff and Content-Disposition: attachment; filename="api.json" (or other appropriate filename for the content type).	Covered
11.7	V11: HTTP security configuration verification requirements	Verify that the Content Security Policy V2 (CSP) is in use in a way that either disables inline JavaScript or provides an integrity check on inline JavaScript with CSP noncing or hashing.	Covered

#	Category	Detail	Level 1
11.8	V11: HTTP security configuration verification requirements	Verify that the X-XSS-Protection: 1; mode=block header is in place.	Covered
16.1	V16: Files and resources verification requirements	Verify that URL redirects and forwards only allow whitelisted destinations, or show a warning when redirecting to potentially untrusted content.	Covered
16.2	V16: Files and resources verification requirements	Verify that untrusted file data submitted to the application is not used directly with file I/O commands, particularly to protect against path traversal, local file include, file mime type, and OS command injection vulnerabilities.	Covered
16.3	V16: Files and resources verification requirements	Verify that files obtained from untrusted sources are validated to be of expected type and scanned by antivirus scanners to prevent upload of known malicious content.	Covered
16.4	V16: Files and resources verification requirements	Verify that untrusted data is not used within inclusion, class loader, or reflection capabilities to prevent remote/local file inclusion vulnerabilities.	Covered
16.5	V16: Files and resources verification requirements	Verify that untrusted data is not used within cross-domain resource sharing (CORS) to protect against arbitrary remote content.	Covered
16.8	V16: Files and resources verification requirements	Verify the application code does not execute uploaded data obtained from untrusted sources.	Covered
16.9	V16: Files and resources verification requirements	Do not use Flash, Active-X, Silverlight, NACL, client-side Java or other client side technologies not supported natively via W3C browser standards.	Covered
17.1	V17: Mobile verification requirements	Verify that ID values stored on the device and retrievable by other applications, such as the UDID or IMEI number are not used as authentication tokens.	Covered
17.2	V17: Mobile verification requirements	Verify that the mobile app does not store sensitive data onto potentially unencrypted shared resources on the device (for example, SD card or shared folders).	Covered
17.3	V17: Mobile verification requirements	Verify that sensitive data is not stored unprotected on the device, even in system protected areas such as key chains.	Covered

#	Category	Detail	Level 1
17.7	V17: Mobile verification requirements	Verify that the application sensitive code is laid out unpredictably in memory (for example, ASLR).	Covered
17.9	V17: Mobile verification requirements	Verify that the app does not export sensitive activities, intents, content providers, and so on for other mobile apps on the same device to exploit.	Covered
17.11	V17: Mobile verification requirements	Verify that the app's exposed activities, intents, content providers, and others validate all inputs.	Covered
18.1	V18: Web services verification requirements	Verify that the same encoding style is used between the client and the server.	Covered
18.2	V18: Web services verification requirements	Verify that access to administration and management functions within the Web Service Application is limited to web service administrators.	Covered
18.3	V18: Web services verification requirements	Verify that XML or JSON schema is in place and verified before accepting input.	Covered
18.4	V18: Web services verification requirements	Verify that all input is limited to an appropriate size limit.	Covered
18.5	V18: Web services verification requirements	Verify that SOAP based web services are compliant with Web Services-Interoperability (WS-I) Basic Profile at minimum.	Covered
18.6	V18: Web services verification requirements	Verify the use of session-based authentication and authorization. Avoid the use of static "API keys" and similar.	Covered
18.7	V18: Web services verification requirements	Verify that the REST service is protected from Cross-Site Request Forgery.	Covered
19.1	V19. Configuration	All components should be up to date with proper security configuration(s) and version(s). This should include removal of unneeded configurations and folders such as sample applications, platform documentation, and default or example users.	Covered

