



appsec

Report

Penetration test of web application [REDACTED]

MILAN BARTOŠ <MILAN@APPSEC.CZ>

Document is intellectual property of company APPSEC s.r.o.. Disposition right belongs to company [REDACTED]. It's forbidden to reproduce, publish or use the document outside of the scope of penetration testing project. Neither as a whole nor in parts.

Contents

1	Introduction	3
1.1	Disclaimer	3
1.2	Tools used for testing	3
1.3	Testing process	3
2	Management summary	4
2.1	Identified vulnerabilities	4
2.2	Conclusion	4
3	Classification of vulnerabilities	5
3.1	According to severity	5
3.2	According to OWASP	5
4	Penetration test results	6
4.1	Tested areas by OWASP	6
4.2	Identified vulnerabilities	10
4.2.1	Stored Cross Site Scripting (XSS) ■	11
4.2.2	Reflected Cross Site Scripting (XSS) ■	13
4.2.3	Possible bruteforcing of passwords ■	15
4.2.4	Possible replay attack during login ■	16
4.2.5	XML injection ■	17
4.2.6	Weak password policy ■	18
4.2.7	Self Cross Site Scripting (XSS) ■	19
4.2.8	Content spoofing in multiple website sections ■	21
4.2.9	Vulnerable front-end components ■	22
4.2.10	Two-factor authorization instead of two-factor authentication ■	23

1 Introduction

1.1 Disclaimer

Penetration test is usually described as exact and complete simulation of attack against particular service or application. Despite penetration test and real attack have many similarities, e.g. knowledge of pentester and attacker, used tools and others, some important differences exist and they need to be taken into account. It's mainly limitation of penetration test by money and/or time.

In case of real attack, attacker can plan the attack itself even for months. He can afford to collect information necessary for attack for a long time. We can't afford such luxury when performing penetration test as such penetration test would be financially unfeasible and wouldn't deliver the results in timely manner.

Because of this, some cooperation from test subject might be necessary during penetration test.

1.2 Tools used for testing

- Burp Suite Pro
- Nmap
- Firefox - {Cookies Manager, FireBug, FoxyProxy}
- Kali Linux
- APPSEC Toolkit
- Metasploit Framework

1.3 Testing process

Tests were performed according to experience and knowledge of penetration testers. Testing was performed with accordance to OWASP Testing Guide and PTES.

2 Management summary

Goal of this project was to identify vulnerabilities in [REDACTED] application that was available at address [REDACTED]. Those OWASP chapters that are mentioned in Tested areas by OWASP chapter were tested.

2.1 Identified vulnerabilities

During penetration testing we have identified nine vulnerabilities, two rated with high severity, four rated with medium severity and three rated with low severity. Most of identified vulnerabilities is in some way related to missing proper input sanitization. This is the area that [REDACTED] team needs to pay much more attention in the future.

2.2 Conclusion

Serious vulnerabilities that could enable attacker to attack one or more users of the system were found. Among most serious vulnerabilities are Cross Site Scripting and various other injection attacks. This either needs to be addressed by proper training or by using some web framework that includes input sanitization as a default functionality.

We recommend paying more attention to user input and user influenceable input validation and filtering. It's also recommended to perform thorough source code analysis to identify those inputs that might have been missed during the penetration test.

After fixing all necessary vulnerabilities we recommend retesting the application to make sure that fixed vulnerabilities were indeed fixed. Regular penetration testing is also recommended as there might have be more vulnerabilities introduced since last testing was performed.

3 Classification of vulnerabilities

Classification of found vulnerabilities is written in this chapter. Each vulnerability is assigned severity according to seriousness of consequences that would lead from exploitation of such vulnerability.

3.1 According to severity

- Critical ■
Serious vulnerability that has direct consequences for the security of tested system.
- High ■
Vulnerability that can have direct consequences for the security of tested system in case of skilled or motivated attacker.
- Medium ■
Vulnerability that isn't serious alone, but in combination with other vulnerability/vulnerabilities can have direct consequences for the security of tested system.
- Low ■
Finding that has low impact on security of tested system, is usually just best practices.
- Informative ■
Finding is of informative severity and isn't security vulnerability.

3.2 According to OWASP

For testing of web applications, we use our methodology that is based on widely recognized project OWASP and its OWASP Testing Guide version 4.

4 Penetration test results

In this chapter there are technical information about performed penetration test. You can see OWASP chapters and information if that chapter was tested or not.

4.1 Tested areas by OWASP

Following chapter is showing tables containing sections of OWASP, that were tested during penetration test.

Information Gathering

Information Gathering	Tested
OTG-INFO-001 - Conduct Search Engine Discovery and Reconnaissance for Information Leakage	✓
OTG-INFO-002 - Fingerprint Web Server	✓
OTG-INFO-003 - Review Webserver Metafiles for Information Leakage	✓
OTG-INFO-004 - Enumerate Applications on Webserver	✓
OTG-INFO-005 - Review Webpage Comments and Metadata for Information Leakage	✓
OTG-INFO-006 - Identify application entry points	✓
OTG-INFO-007 - Map execution paths through application	✓
OTG-INFO-008 - Fingerprint Web Application Framework	✓
OTG-INFO-009 - Fingerprint Web Application	✓
OTG-INFO-010 - Map Application Architecture	✓

Configuration and Deploy Management Testing

Configuration and Deploy Management Testing	Tested
OTG-CONFIG-001 - Test Network/Infrastructure Configuration	✓
OTG-CONFIG-002 - Test Application Platform Configuration	✓
OTG-CONFIG-003 - Test File Extensions Handling for Sensitive Information	✓
OTG-CONFIG-004 - Review Old, Backup and Unreferenced Files for Sensitive Information	✓
OTG-CONFIG-005 - Enumerate Infrastructure and Application Admin Interfaces	✓
OTG-CONFIG-006 - Test HTTP Methods	✓
OTG-CONFIG-007 - Test HTTP Strict Transport Security	✓
OTG-CONFIG-008 - Test RIA cross domain policy	✓

Identity Management Testing

Identity Management Testing	Tested
OTG-IDENT-001 - Test Role Definitions	✓
OTG-IDENT-002 - Test User Registration Process	✓
OTG-IDENT-003 - Test Account Provisioning Process	✓
OTG-IDENT-004 - Testing for Account Enumeration and Guessable User Account	✓
OTG-IDENT-005 - Testing for Weak or unenforced username policy	✓

Authentication Testing

Authentication Testing	Tested
OTG-AUTHN-001 - Testing for Credentials Transported over an Encrypted Channel	✓
OTG-AUTHN-002 - Testing for default credentials	✓
OTG-AUTHN-003 - Testing for Weak lock out mechanism	✓
OTG-AUTHN-004 - Testing for bypassing authentication schema	✓
OTG-AUTHN-005 - Test remember password functionality	✓
OTG-AUTHN-006 - Testing for Browser cache weakness	✓
OTG-AUTHN-007 - Testing for Weak password policy	✓
OTG-AUTHN-008 - Testing for Weak security question/answer	✓
OTG-AUTHN-009 - Testing for weak password change or reset functionalities	✓
OTG-AUTHN-010 - Testing for Weaker authentication in alternative channel	✓

Authorization Testing

Authorization Testing	Tested
OTG-AUTHZ-001 - Testing Directory traversal/file include	✓
OTG-AUTHZ-002 - Testing for bypassing authorization schema	✓
OTG-AUTHZ-003 - Testing for Privilege Escalation	✓

Session Management Testing

Session Management Testing	Tested
OTG-SESS-001 - Testing for Bypassing Session Management Schema	✓
OTG-SESS-002 - Testing for Cookies attributes	✓
OTG-SESS-003 - Testing for Session Fixation	✓
OTG-SESS-004 - Testing for Exposed Session Variables	✓
OTG-SESS-005 - Testing for Cross Site Request Forgery	✓
OTG-SESS-006 - Testing for logout functionality	✓
OTG-SESS-007 - Test Session Timeout	✓
OTG-SESS-008 - Testing for Session puzzling	✓

Data Validation Testing

Data Validation Testing	Tested
OTG-INPVAL-001 - Testing for Reflected Cross Site Scripting	✓
OTG-INPVAL-002 - Testing for Stored Cross Site Scripting	✓
OTG-INPVAL-003 - Testing for HTTP Verb Tampering	✓
OTG-INPVAL-004 - Testing for HTTP Parameter pollution	✓
OTG-INPVAL-005 - Testing for SQL Injection	✓
OTG-INPVAL-006 - Testing for LDAP Injection	✓
OTG-INPVAL-007 - Testing for ORM Injection	✓
OTG-INPVAL-008 - Testing for XML Injection	✓
OTG-INPVAL-009 - Testing for SSI Injection	✓
OTG-INPVAL-010 - Testing for XPath Injection	✓
OTG-INPVAL-011 - IMAP/SMTP Injection	✓
OTG-INPVAL-012 - Testing for Code Injection	✓
OTG-INPVAL-013 - Testing for Command Injection	✓
OTG-INPVAL-014 - Testing for Buffer overflow	✓
OTG-INPVAL-015 - Testing for incubated vulnerabilities	✓
OTG-INPVAL-016 - Testing for HTTP Splitting/Smuggling	✓

Error Handling Testing

Error Handling Testing	Tested
OTG-ERR-001 - Analysis of Error Codes	✓
OTG-ERR-002 - Analysis of Stack Traces	✓

Cryptography Testing

Cryptography Testing	Tested
OTG-CRYPST-001 - Testing for Weak SSL/TSL Ciphers, Insufficient Transport Layer Protection	✓
OTG-CRYPST-002 - Testing for Padding Oracle	✓
OTG-CRYPST-003 - Testing for Sensitive information sent via unencrypted channels	✓

Business Logic Testing

Business Logic Testing	Tested
OTG-BUSLOGIC-001 - Test Business Logic Data Validation	✓
OTG-BUSLOGIC-002 - Test Ability to Forge Requests	✓
OTG-BUSLOGIC-003 - Test Integrity Checks	✓
OTG-BUSLOGIC-004 - Test for Process Timing	✓
OTG-BUSLOGIC-005 - Test Number of Times a Function Can be Used Limits	✓
OTG-BUSLOGIC-006 - Testing for the Circumvention of Work Flows	✓
OTG-BUSLOGIC-007 - Test Defenses Against Application Mis-use	✓
OTG-BUSLOGIC-008 - Test Upload of Unexpected File Types	✓
OTG-BUSLOGIC-009 - Test Upload of Malicious Files	✓

Client Side Testing

Client Side Testing	Tested
OTG-CLIENT-001 - Testing for DOM based Cross Site Scripting	✓
OTG-CLIENT-002 - Testing for JavaScript Execution	✓
OTG-CLIENT-003 - Testing for HTML Injection	✓
OTG-CLIENT-004 - Testing for Client Side URL Redirect	✓
OTG-CLIENT-005 - Testing for CSS Injection	✓
OTG-CLIENT-006 - Testing for Client Side Resource Manipulation	✓
OTG-CLIENT-007 - Test Cross Origin Resource Sharing	✓
OTG-CLIENT-008 - Testing for Cross Site Flashing	✓
OTG-CLIENT-009 - Testing for Clickjacking	✓
OTG-CLIENT-010 - Testing WebSockets	✓
OTG-CLIENT-011 - Test Web Messaging	✓
OTG-CLIENT-012 - Test Local Storage	✓

4.2 Identified vulnerabilities

In this chapter, there are all vulnerabilities and findings sorted from the most serious to least serious. It's all findings that were identified during penetration testing and they contain their risk and recommendations on how to eliminate them. It's not list of all possible vulnerabilities that could be in the application, because of resource limits on penetration testing.

4.2.1 Stored Cross Site Scripting (XSS) ■

Severity:

Finding

Cross site scripting vulnerability is a type of injection vulnerability when malicious code is injected into otherwise nonmalicious application. Attacker can use XSS to send malicious web application scripts to an unsuspecting user.

Stored XSS is most dangerous cross site scripting attack as attackers malicious code is stored within application itself and is delivered through usual use of web application.

Following locations were identified as being vulnerable to Stored XSS:

- [redacted]
 - The Stored XSS vulnerability [redacted]
 - [redacted]
 - [redacted]
- [redacted]
 - Stored XSS vulnerability [redacted]
 - [redacted]
 - [redacted]
 - [redacted]
- [redacted]
 - Stored XSS vulnerability [redacted]
 - [redacted]
 - [redacted]
- [redacted]
 - The Stored XSS vulnerability [redacted]
 - [redacted]
- [redacted]
 - The Stored XSS vulnerability [redacted]
 - [redacted]
 - [redacted]
- [redacted]
 - The Stored XSS vulnerability [redacted]

- [REDACTED]
- [REDACTED]
- [REDACTED]
- [REDACTED]
 - The Stored XSS vulnerability [REDACTED]
 - [REDACTED]
 - [REDACTED]

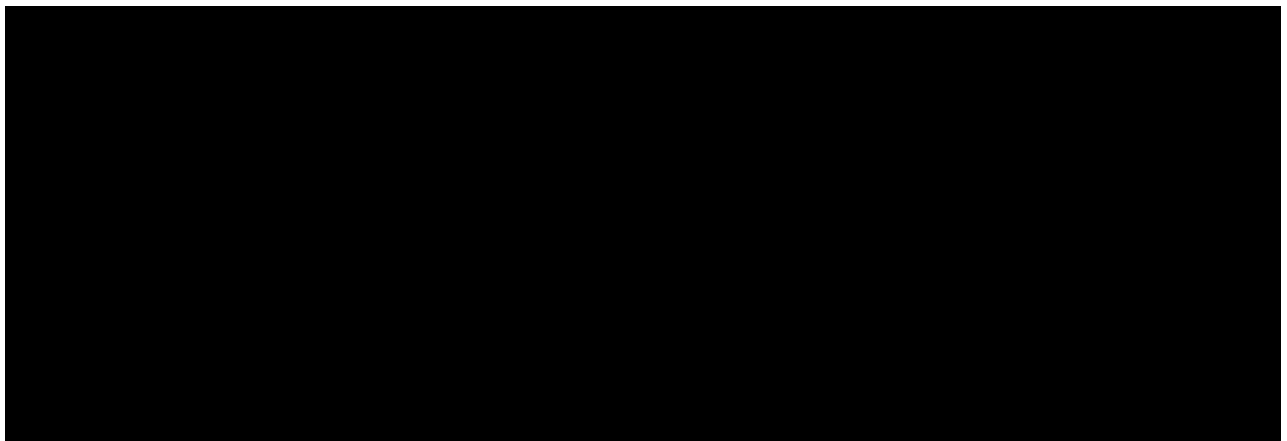


Figure 1: Performing cross site scripting attack

Risk

This vulnerability can be used to conduct a number of browser-based attacks including:

- Hijacking another user's browser
- Capturing sensitive information viewed by application users
- Pseudo defacement of the application
- Port scanning of internal hosts ("internal" in relation to the users of the web application)
- Directed delivery of browser-based exploits
- Other malicious activities

Recommendation

We recommend checking all functions that work in any way with user input (forms, cookies, parameters, ...), or user-influenceable input (http headers, out of band inputs, ...) for weak input validation. In case input validation is weak, filters need to be implemented for such inputs. Filters should use whitelist approach with strictly limited data that are allowed. All data also need to have specified encoding.

4.2.2 Reflected Cross Site Scripting (XSS) ■

Severity:



Finding

Cross site scripting vulnerability is a type of injection vulnerability when malicious code is injected into otherwise nonmalicious application. Attacker can use XSS to send malicious web application scripts to an unsuspecting user.

In case of Reflected XSS, attacker needs to find some way how to deliver malicious payload to the user. Usually it's done within URL or from some other website by POST request.

Following locations were identified as being vulnerable to Reflected XSS:

- 
 - The reflected XSS vulnerability 
 - 

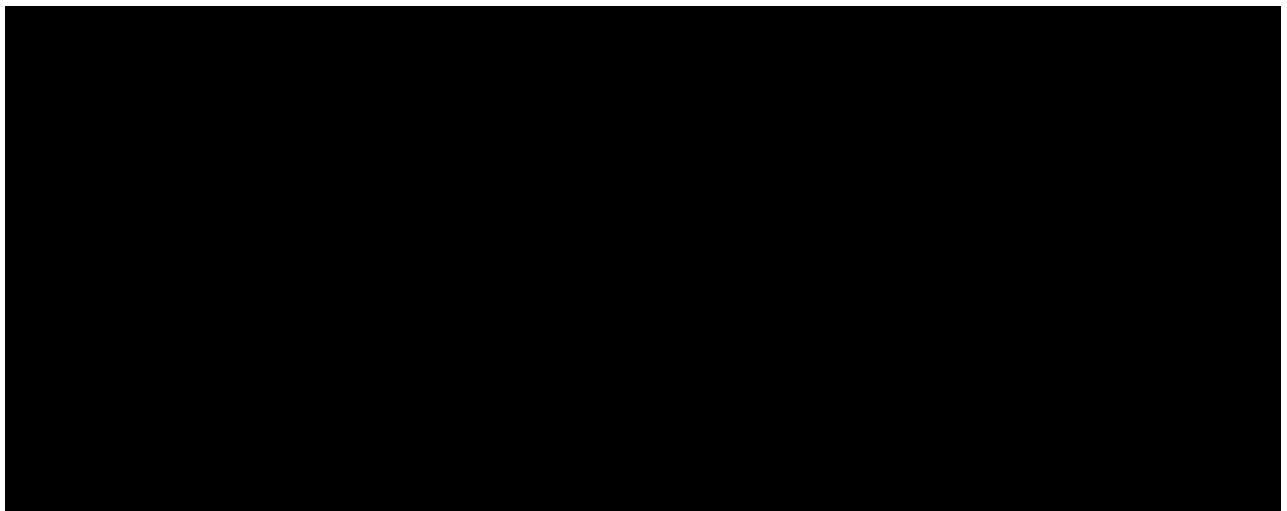
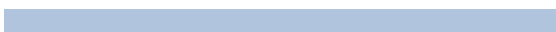




Figure 2: Performing cross site scripting attack

- 
 - The reflected XSS vulnerability 
 - 

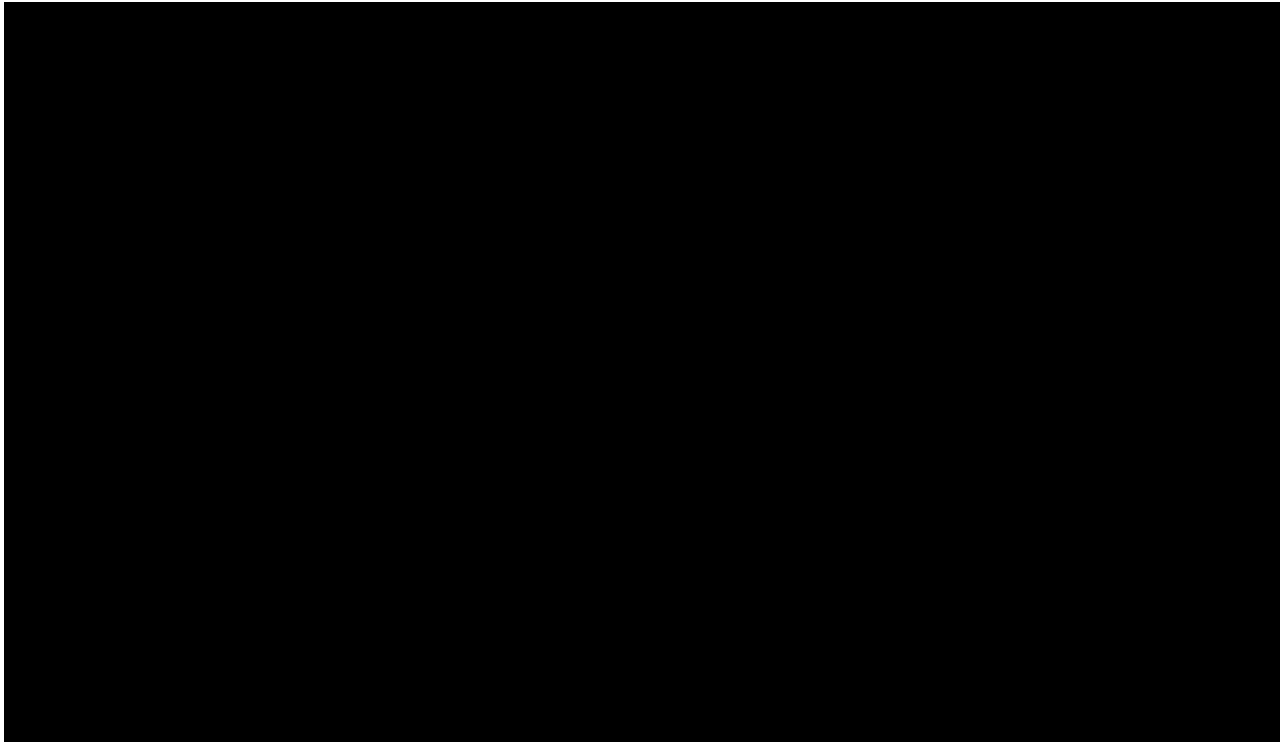


Figure 3: Performing cross site scripting attack

- - The reflected XSS vulnerability
 -

Risk

This vulnerability can be used to conduct a number of browser-based attacks including:

- Hijacking another user's browser
- Capturing sensitive information viewed by application users
- Pseudo defacement of the application
- Port scanning of internal hosts ("internal" in relation to the users of the web application)
- Directed delivery of browser-based exploits
- Other malicious activities

Recommendation

We recommend checking all functions that work in any way with user input (forms, cookies, parameters, ...), or user-influenceable input (http headers, out of band inputs, ...) for weak input validation. In case input validation is weak, filters need to be implemented for such inputs. Filters should use whitelist approach with strictly limited data that are allowed. All data also need to have specified encoding.

4.2.3 Possible bruteforcing of passwords ■

Severity:**Finding**

During penetration test we found that there is no mechanism that would prevent bruteforcing of passwords for user accounts. There was detected no delay in each invalid log in attempt. Also no CAPTCHA mechanism after few invalid login attempts was detected. No account lockout was detected.

Risk

When no mechanism that would prevent bruteforcing of passwords exists, attacker can try passwords for user accounts indefinitely and eventually find the correct password for the account. This can become serious issue because of Weak password policy finding.

Recommendation

We recommend implementing some mechanism that would prevent bruteforcing of user account passwords. It's up to the application authors and management to decide what way of protection is best for application of such kind.

4.2.4 Possible replay attack during login ■

Severity:**Finding**

During penetration testing we found, that token generated by application for log in can be used more times than just once. This can be tested by intercepting the login attempt with atoken parameter populated and sending the request more times than just once. Only first request should succeed and all others should fail. This isn't happening.

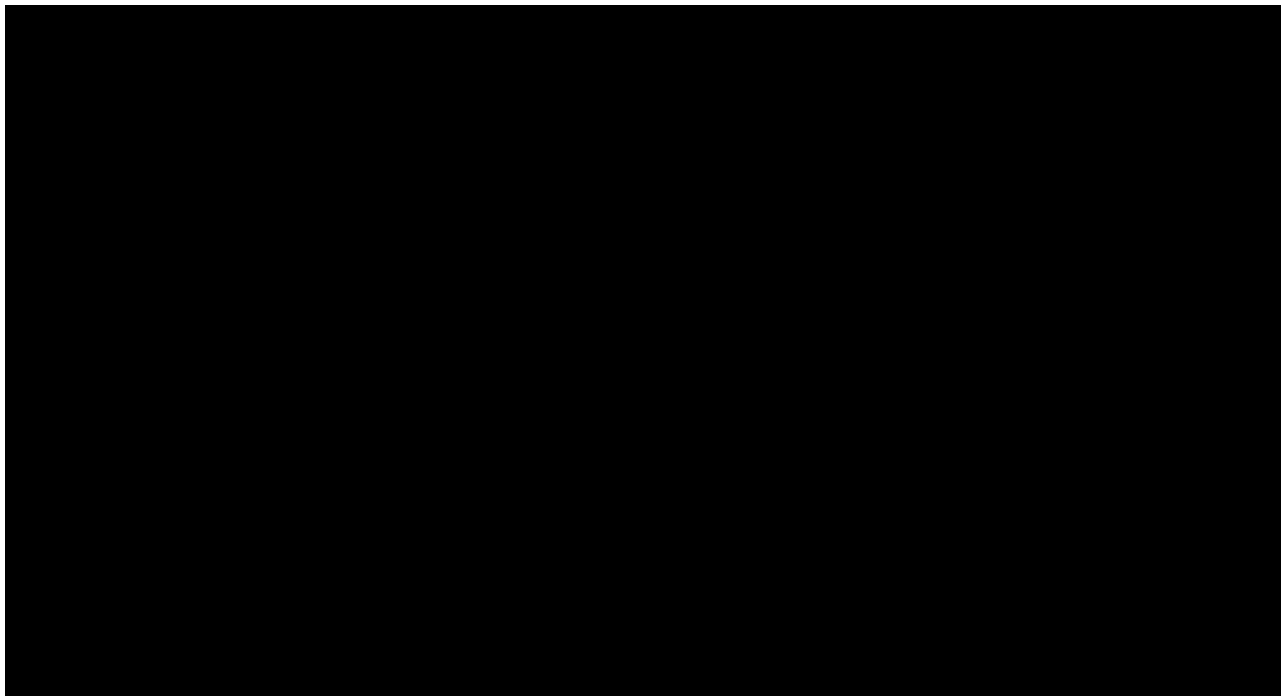


Figure 4: Replay attack performed in Burp Suite Pro

Risk

When attacker is able to intercept communication or in any other way gets the token into his possession, he might be able to perform replay attack and effectively log in to the application as the user the token belongs to.

Recommendation

To reduce risks associated with replay attacks, we recommend that token can only be used only once. After first use, token should be invalidated and it shouldn't be possible to use it again.

4.2.5 XML injection ■

Severity:**Finding**

During penetration testing we found that [REDACTED] is vulnerable to injection of XML data from user-influenceable data [REDACTED].

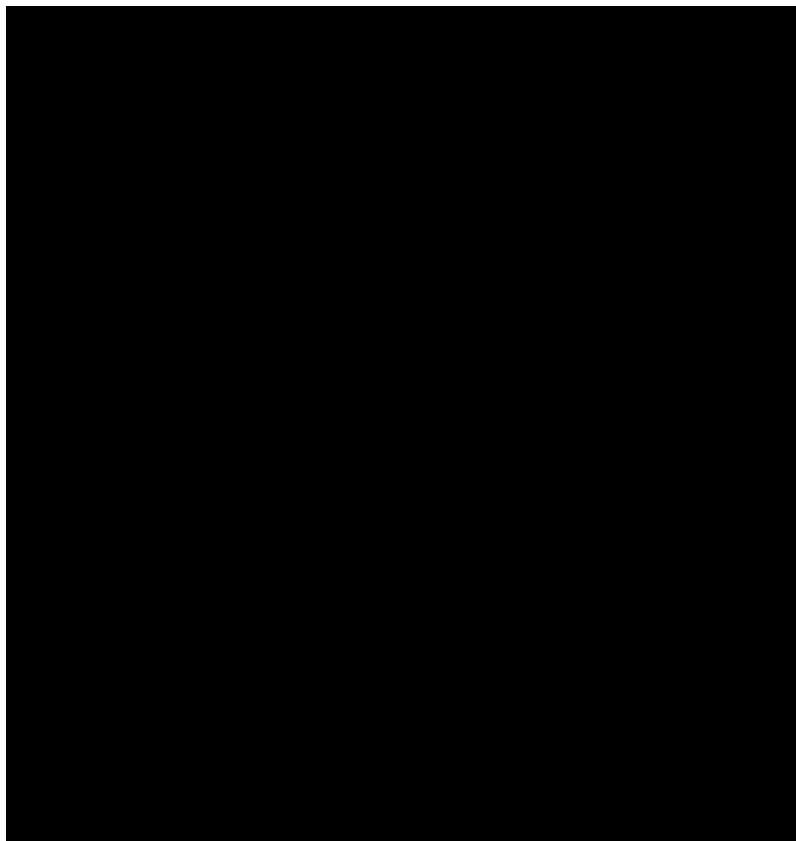


Figure 5: Injection of XML data

Risk

Based on particular location and functionality that is influenced by XML injection, attacker might be able to influence file content or result of sensitive operation.

Recommendation

We recommend checking all functions that work in any way with user input (forms, cookies, parameters, ...), or user-influenceable input (http headers, out of band inputs, ...) for weak input validation. In case input validation is weak, filters need to be implemented for such inputs. Filters should use whitelist approach with strictly limited data that are allowed. All data also need to have specified encoding.

4.2.6 Weak password policy ■

Severity:**Finding**

During the penetration test we found that passwords generated by application consist only of lowercase letters and numbers. They are also just 6 characters long. Such passwords can't be considered secure under any circumstances and are easy to bruteforce.

Risk

If default passwords are not changed for every user, weak generated passwords end up being used. This allows attacker to brute force passwords to user accounts with very little effort. Also in case database is leaked, brute forcing hashes (if passwords are hashed...) would be very easy with this strength of passwords.

Recommendation

We recommend improving the strenght of default generated passwords, e.g. by following the following rules:

- at least 8 characters, preferably 12
- at least one character from each of the following group
 - lowercase letter
 - uppercase letter
 - number
 - special character (, . - / # @ * % ())

4.2.7 Self Cross Site Scripting (XSS) ■

Severity:





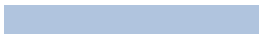









Finding

Cross site scripting vulnerability is a type of injection vulnerability when malicious code is injected into otherwise nonmalicious application. Attacker can use XSS to send malicious web application scripts to an unsuspecting user.

In case of Self XSS, attacker needs to find some way how to make user write the XSS payload into his application interface. This is usually done via phishing e-mails or phishing calls.

Following locations were identified as being vulnerable to Self XSS:

- 
 - 
 - 
 - 
- 
 - 
 - 
 - 
 - 
- 
 - 
 - 
 - 

Risk

This vulnerability can be used to conduct a number of browser-based attacks including:

- Hijacking another user's browser
- Capturing sensitive information viewed by application users
- Pseudo defacement of the application
- Port scanning of internal hosts ("internal" in relation to the users of the web application)
- Directed delivery of browser-based exploits
- Other malicious activities

Recommendation

We recommend checking all functions that work in any way with user input (forms, cookies, parameters, ...), or user-influenceable input (http headers, out of band inputs, ...) for weak input validation. In case input validation is weak, filters need to be implemented for such inputs. Filters should use whitelist approach with strictly limited data that are allowed. All data also need to have specified encoding.

4.2.8 Content spoofing in multiple website sections ■

Severity:



Finding

During penetration test it was found, that the web application displays on the page information passed via [REDACTED] HTTP GET parameters. A remote attacker can trick the victim to open the URL and perform spoofing attack.

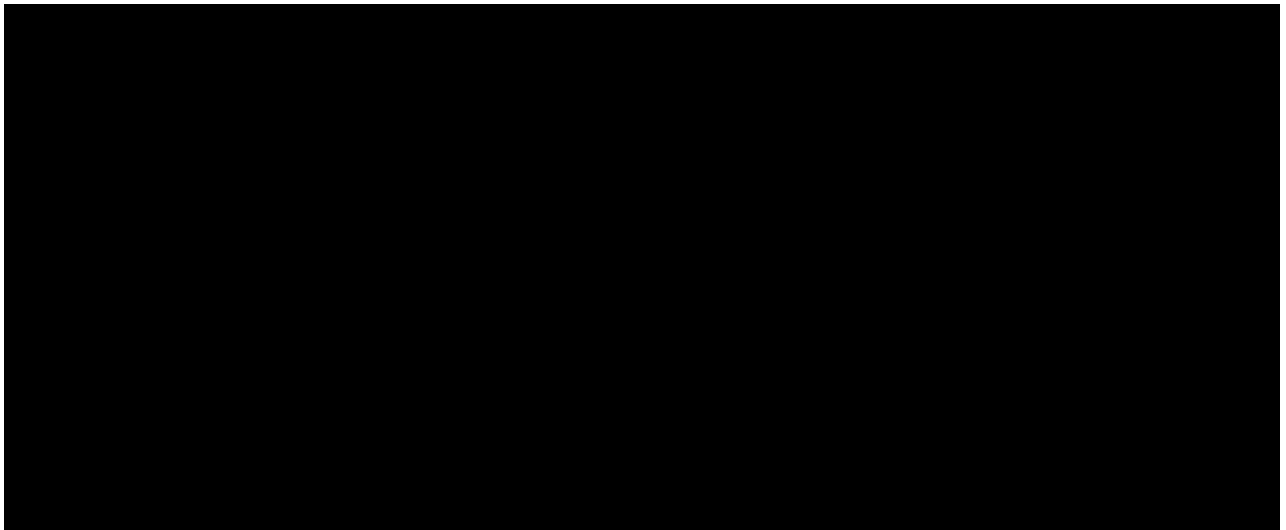


Figure 6: Content spoofing

Risk

Attacker might be able to trick user to open spoofed URL and potentially perform related malicious actions against the user.

Recommendation

Do not display information that comes from untrusted sources. If it needs to be done, perform proper sanitization of such input.

4.2.9 Vulnerable front-end components ■

Severity:**Finding**

During penetration test it was found, that some very old javascript libraries are used in webmail application. It's namely:

- [redacted]
- [redacted]

Risk

Security risk of using old versions of components in application is that old versions may contain some vulnerabilities that get to be public. The longer old version of component is used the longer the risk of some attacker exploiting such vulnerability.

Recommendation

We recommend updating used javascript libraries to newest supported versions and update them regularly to reduce time window in which they can be attacked through some known vulnerability.

4.2.10 Two-factor authorization instead of two-factor authentication ■

Severity:

Finding

It was found that two-factor authentication is actually called two-factor authorization in the application. This lead to assumption that no information security educated person was involved in development, otherwise this wouldn't stay unnoticed.



Figure 7: Two factor authorization

Risk

If no information security educated person was involved in development, there is high probability that application will contain security vulnerabilities that were missed by information security-unaware developers.

Recommendation

This finding is information and isn't security vulnerability.