**Pentesting Report**

Table of Contents

[1 Executive summary 2](#_Toc167869430)

[2 Customer contacts 2](#_Toc167869431)

[3 IPS contacts 2](#_Toc167869432)

[4 Method 3](#_Toc167869433)

[5 Scope 3](#_Toc167869434)

[6 Assessment 4](#_Toc167869435)

[6.1 Summary of findings 4](#_Toc167869436)

[6.2 Identified Devices 4](#_Toc167869437)

[6.3 Overview 4](#_Toc167869438)

[7 Technical findings details 6](#_Toc167869439)

[7.1 Improper Neutralization of Special Elements Used in a Template Engine 6](#_Toc167869440)

[7.2 Permissive Cross-domain Policy with Untrusted Domains 8](#_Toc167869441)

[7.3 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') 10](#_Toc167869442)

[7.4 Improper Restriction of XML External Entity Reference 11](#_Toc167869443)

[7.5 Debug Messages Revealing Unnecessary Information 14](#_Toc167869444)

[7.6 Improper Restriction of XML External Entity Reference 14](#_Toc167869445)

[8 Recommendations 14](#_Toc167869446)

[8.0 Overview 14](#_Toc167869447)

[8.1 Remedies 15](#_Toc167869448)

[8.1.1 Improper Neutralization of Special Elements Used in a Template Engine 15](#_Toc167869449)

[8.1.2 Permissive Cross-domain Policy with Untrusted Domains 15](#_Toc167869450)

[8.1.3 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') 17](#_Toc167869451)

[8.1.4 Improper Restriction of XML External Entity Reference 17](#_Toc167869452)

[8.1.5 Debug Messages Revealing Unnecessary Information 17](#_Toc167869453)

[8.1.6 Missing rate limiting on the API backend 17](#_Toc167869454)

# Executive summary

PacketFront Software Solutions AB (“PacketFront Software” herein) contracted IP-Solutions to perform a Penetration Test of the [BBE](https://pfsw.com/bbe/)\* product and its backend APIs to identify security weaknesses, determine the impact to PacketFront Software, document all findings in a clear and repeatable manner, and provide remediation recommendations.

# Customer contacts

|  |  |  |
| --- | --- | --- |
| *Role* | *Name* | *Contact details* |
| CTO | Jonas Ohlsson | jonas.ohlsson@pfsw.com |
| IT-Architect | Daniel Lundqvist | daniel.lundqvist@pfsw.com |

# IPS contacts

|  |  |  |
| --- | --- | --- |
| *Role* | *Name* | *Contact details* |
| IT-Security Con-sultant | Aleksandar Milosavljevic | aleksandar.milosavljevic@ip-solut-ions.se |
| BDM-Security | Mattias Paajanen | mattias.paajanen@ip-solutions.se |

# Method

IP-Solutions performed a testing under “grey box” approach during days between 2024-05-23 to 2024-05-28 with a limited knowledge of the internal network and the BECS product itself. In addition, the customer provided a local account used by the tester to login into the web management portal and a reference document of the internal API as well.

Testing was conducted from a “non-evasive” point of view with the primary scope to identify as many vulnerabilities and misconfigurations as possible, without creating any possible disruptions.

Testing was performed remotely from consultant’s laptop where IP-Solutions AB public ip have been whitelisted for the entire duration of the test. Each weakness identified was documented and manually investigated to determine the exploitation possibilities, and escalation potential. In the case of successful foothold IP-Solutions weren’t allowed for further testing as lateral movement and horizontal/vertical privilege escalation instead, the tester was asked to promptly inform the customer by documenting all the findings.

# Scope

The customer provided a list of owned public IPv4 addresses.

|  |  |
| --- | --- |
| *IP* | *Info* |
| 151.236.205.230 | Used by the BBE demo instance. |
| 151.236.205.231 | Not in use during the testing exercise. |
| 151.236.205.232 | Not in use during the testing exercise. |

# Assessment

## Summary of findings

|  |  |  |
| --- | --- | --- |
| *Severity* | *ID* | *Details* |
| Critical | 1 | Improper Neutralization of Special Elements Used in a Template Engine. |
| High | 2 | Permissive Cross-domain Policy with Untrusted Domains. |
| Medium | 3 | Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting'). |
| Low | 4 | Improper Restriction of XML External Entity Reference |
| Low | 5 | Debug Messages Revealing Unnecessary Information. |
| Informational | 6 | Missing rate limiting on the API backend. |

## Identified Devices

|  |  |  |
| --- | --- | --- |
| *Hostname* | *IP* | *Details* |
| PFSW-demo | 151.236.205.230 | BBE demo instance. |

## Overview

The tester found several issues on PacketFront’s BBE application, where one critical finding has been discovered that led to an actual breach of the environment via *\*RCE (Remote Code Execution)*. In accordance with the customer, it is relevant to mention that invasive attacks like DOS (Denial of Service) and heavy brute-forcing haven’t been tested on the device.

The first finding was related to improper sanitization of the elements used by the *Freemarker* *\*template engine* allowing the user to customize the “System Parameters” from the web management portal and eventually execute shell commands on behalf of the backend server.

The second finding was related to a permissive *\*CORS (Cross-origin resource sharing)* policy that permits exfiltration of sensitive data to untrusted domains resulting in session spoofing attacks that can steal other user’s session. These types of attacks often are carried in tandem with *\*XSS (Cross-site scripting)* attacks.

The next issue pertains to missing input sanitization during web page generation for ticket creation functionality. Generally, *\*XSS* attacks are not considered dangerous because JavaScript code is executed locally in the client browser, however the danger can exponentially increase if chained with *\*CORS* attacks, as described above.

The fourth issue is related to improper neutralization of *\*XEE (XML external entity)* data via an obsolete library used in a PDF generator for reporting customization functionality.

Another issue relates to excessive verbosity that occurs during a client request, revealing unnecessary information such as the specific version of the JSP (Java) server used by the backend.

Lastly, the tester didn’t notice any rate limiting technology being applied during the testing exercise.

In addition, the customer should ensure that all remediation steps and mitigating controls are carefully planned and tested to prevent any service disruptions or loss of data.

*\*Template Engine: A Template Engine simplifies the use of static templates with minimal code. During runtime on the client side, variables in the template are replaced with actual values. These engines help developers create templates for web pages, written in a markup language with placeholders for dynamic content. When rendered, these placeholders are substituted with real data, producing a dynamic document.*

*\*RCE: Remote Code Execution (also known as RCE) refers to a class of cyberattacks in which attackers remotely execute commands to place malware or other malicious code on your computer or network. A remote code execution vulnerability can compromise a user’s sensitive data without the hackers needing to gain physical access to your network.*

*\*CORS: Cross-origin resource sharing (CORS) is a browser mechanism which enables controlled access to resources located outside of a given domain.*

*\*XSS: Cross-site scripting (also known as XSS) is a web security vulnerability that allows an attacker to compromise the interactions that users have with a vulnerable application.*

*\*XXE: XML external entity injection (also known as XXE) is a web security vulnerability that allows an attacker to interfere with an application's processing of XML data. It often allows an attacker to view files on the application server filesystem, and to interact with any back-end or external systems that the application itself can access.*

# Technical findings details

## Improper Neutralization of Special Elements Used in a Template Engine

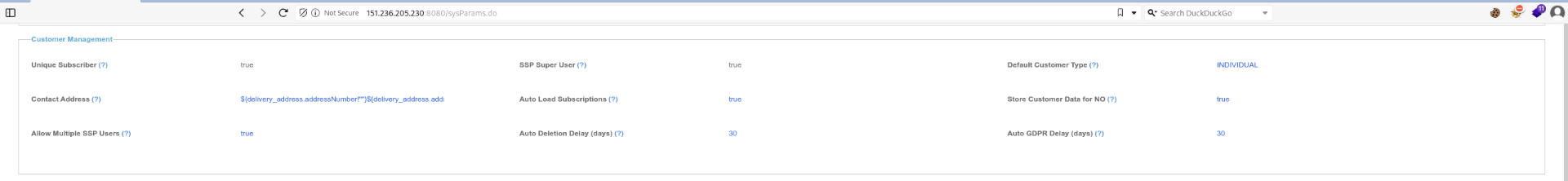
Severity: **CRITITAL**

The tester identified from the official documentation that the backend uses the *Freemarker (Java)* template engine to populate dynamically parts of the website.

A screenshot of a computer

Description automatically generated

Being part of the *BBEAdmin* group we have access to the “System 🡪 System Parameters” module where a user can customize how fields are shown and populated by the template engine to match the user needs. Specifically, the tester used the “Contact address” field described in the following image as carrier to the attacks.



Template engines are known to be vulnerable to SSTI (Server-Side template injection) attacks and happens when an attacker can use native template syntax to inject a malicious payload into a template, which is then executed server-side. Trying to inject an easy arithmetical multiplication as described below, it’s a quick test that can reveal the presence of the vulnerability.



Upon visiting the customer list, we can see that the “Customer Address” field is populated with the value of the arithmetic multiplication which denotes the server-side code execution.

A screenshot of a computer

Description automatically generated

This attack can be then taken further to gain RCE (Remote code execution) on the backend server, specifical reference commands have been taken from [this article](https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Server%20Side%20Template%20Injection#freemarker).

To demonstrate the danger of these type of attacks the tester proceed then by executing a command of the machine via the package [freemarker.template.utility](https://freemarker.apache.org/docs/api/freemarker/template/utility/package-summary.html) that allows the execution of external commands.

A screenshot of a computer

Description automatically generated

Getting back to the Customer’s list a verbose output of the executed *\*id* command is now shown on the Contact Address field.

A screenshot of a computer

Description automatically generated

As requested by the customer, the tester didn’t take any further steps in the attack chain. It is also relevant to mention that the targeted field used in this POC isn’t the only one vulnerable and virtually many more fields could have been used to carry the same type of attack as several reporting functionalities shown in the picture here below.

A screenshot of a computer

Description automatically generated

*\*id: in Linux bash is used to find out user and group names and numeric ID’s (UID or group ID) of the current user or any other user in the server.*

## Permissive Cross-domain Policy with Untrusted Domains

Severity: **HIGH**

The Cross-origin resource sharing (CORS) is poorly implemented on the backend webserver which allows potential adversaries to exfiltrate sensitive data like SessionTokens to untrusted domains.

The tester wrote a simple JS code that uses the [XmlHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) function to perform http requests and specifically exfiltrate the cookie of the victim that is executing the code on the website.

A screenshot of a computer

Description automatically generated

A temporary HTTP webserver is set up on the tester’s machine to host the malicious JS script and exfiltrate the victim’s cookie.



From here the tester identified the presence of a XSS (Cross-site scripting) injection via ticket creation functionality, specifically in the ticket description field, where a JS code that fetches the malicious JS script from the tester’s self-hosted http server.

A screenshot of a computer

Description automatically generated

To simulate a more realistic scenario, another local user was added to the system.

A screenshot of a computer

Description automatically generated

Before proceeding with the final steps of the attack chain we can check the new user’s session token via a cookie-editor plugin, largely available for most of the browsers out there.

A screenshot of a computer

Description automatically generated

When the victim (David in this example) opens the malicious ticket, his browser will execute locally the JS code and fetch the self-hosted script that will finally exfiltrate the session id by performing a cross-site request to an untrusted domain.

A screen shot of a computer

Description automatically generated

The specifically “JSESSIIONID=XXXX” is the exact session id used by the victim’s session and it can be used to gain access to other user’s sessions without knowing their login credentials.

## Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Severity: **MEDIUM**

It has already been mentioned in the previous paragraphs that there is a missing sanitization of user input data in the ticketing functionality, specifically in the *“Ticket Description”* field, allowing the injection of JS code that leads to XSS (Cross-site scripting) attacks.

An adversary holding access to the ticketing functionality can create a new ticket and inject a JS code in the description field, reference commands have been take from [this article](https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/XSS%20Injection#common-payloads).

A screenshot of a computer

Description automatically generated

Upon opening of the ticket by any user, the JS code is then executed locally on the victim browser interpreter and an alert message pop up.

A screenshot of a computer

Description automatically generated

It is important to mention that this specific attack targets only users who have access to the ticketing function. The code is stored in the ticket itself, and while it might not be particularly dangerous on its own, it becomes exponentially more potent when combined with CSRF (Cross-Site Request Forgery) and CORS (Cross-Origin Resource Sharing) bypass attacks.

## Improper Restriction of XML External Entity Reference

Severity: **LOW**

The application used by the backend to dynamically generate customized reports in PDF uses an old library that is known to be vulnerable to XEE (XML External Entity) attacks.

The tester proceeds to generate a new report from the list of templates available from the “Reporting” menu.



Upon downloading and fetching the response file in [Burpsuite](https://portswigger.net/burp/communitydownload) the file structure shows that JesperReports 6.10.0 module was used to generate the file itself.

A screenshot of a computer

Description automatically generated

This information can also be found more easily by parsing the generated pdf report with tools that have the ability to read \*exif data like [exfiltool](https://exiftool.org/).

A screenshot of a computer

Description automatically generated

We can also see that this module uses another library called iText v. 2.1.7 which is outdated and known to be vulnerable to XEE attacks, briefly described on the [CVE-2017-9096](https://nvd.nist.gov/vuln/detail/CVE-2017-9096).

XXE attacks tend to be relatively dangerous as it is relatively easy to reproduce and can result in the ability to read content of local files stored on the backend machine or even in some cases remote code execution.

The tester tried to reproduce the attack by injecting some malicious XML data under several fillable fields in a customized report. The goal of the code is to read the local file *“/etc/passwd”* that stores information about the users on the backend server.



But all the data is treated as string and no HTML code is parsed.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

In this case the tester holds no access to the backend template and the user input seems pre-processed before sent back to the backend JesperReport module. For this reason, the vulnerability has been categorized respectively from Medium to Low.

*\*Exif: Exchangeable Image File Format (EXIF) is a standard that defines specific information related to an image or other media.*

## Debug Messages Revealing Unnecessary Information

Severity: **LOW**

The backend is revealing unnecessary information about its version via *“X-Powered-By”* header in its responses sent back to clients.

A screenshot of a computer

Description automatically generated

Being too verbose with debugging information can pose a risk, as adversaries tend to use this information to tailor the exploit and be very specific about what their attacks are aiming for.

## Improper Restriction of XML External Entity Reference

Severity: **INFORMATIONAL**

The pentester didn’t notice any rate limiting applied on the backend API, specifically not on sensitive functions like the session login that can be target of potential brute forcing attacks.

# Recommendations

## Overview

As stated in the introduction, several issues were found, and specifically, one allowed the tester to gain remote code execution on the webserver.

The remediation measures described in the following paragraphs may not be universally suitable or applicable; careful tailoring may be necessary.

Additionally, the customer should ensure that all remediation steps and mitigating controls are carefully planned and tested to prevent service disruptions or data loss.

## Remedies

Ranked in falling order of priority.

### Improper Neutralization of Special Elements Used in a Template Engine

The first identified vulnerability can be tackled in two steps, the first is to apply a sanitization over the user input before passing the data back to the template engine. This step would be the key to neutralize the command execution function and other risky characters described in the previous sections.

The second step is updating Freemarker to the latest version possible. Specifically, the tester wasn’t able to determine the exact version used by the backend, but most of the exploits available in the wild are based on the [CVE-2021-25770](https://nvd.nist.gov/vuln/detail/CVE-2021-25770).

Nonetheless, all the code execution and sandbox bypass exploits are known to be working for Freemarker version below 2.30.0.

### Permissive Cross-domain Policy with Untrusted Domains

The actual CORS policy should be addressed, and no request should be possible to external untrusted domains. The following picture shows the error that the tester got when testing the victim cookie exfiltration, the error message complains about the “[Access-Control-Allow-Origin](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Origin)” policy that is missing.

A screenshot of a computer

Description automatically generated

In this specific example we can identify that the [SameSite](https://owasp.org/www-community/SameSite) policy is not set as “Strict”.

A screenshot of a computer

Description automatically generated

The main goal of a [SameSite](https://owasp.org/www-community/SameSite) policy is to mitigate the risk of cross-origin information leakage. The actual policy can be set as ([Strict, Lax, or None](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie#attributes)), where None poses no protection, and Strict is the more restrictive option. Lastly the Lax option seen as a way in between and means that the cookie is not sent on cross-site requests, such as on requests to load images or frames, but is sent when a user is navigating to the origin site from an external site.

Going a bit deeper into details to fulfil the policy condition, two sites are considered the same when the schema, domain, and port match (http://mysite.com:80 & http://mysite.com:80). A mismatch would occur when the schemas are different, such as (https://mysite.com:443 & http://mysite.com:80).

In addition, using the “*HttpOnly*” option which optional by default and forbids JavaScript from accessing the cookie, for example, through the [Document.cookie](https://developer.mozilla.org/en-US/docs/Web/API/Document/cookie) property.

It is important to mention that upon [SameSite](https://owasp.org/www-community/SameSite) policy implementation, a configuration of the CORS policy will be required. This policy is used to loosen up the [SameSite](https://owasp.org/www-community/SameSite) by creating exclusions. For example, it specifies which domains are allowed to make cross-origin requests via the [Access-Control-Allow-Origin](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Origin) option, as seen previously.

### Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

To address the Cross-Site scripting vulnerability identified in the ticketing function the customer should pre-process the user input data and sanitize unwanted words and special escape characters used typically in these kinds of attacks.

Blacklist filtering should be applied both on the front-end and back-end to minimize the risk of storing unwanted code. In order to archive this, the customer might use libraries that are designed for this specific task, example [DomPurify](https://github.com/cure53/DOMPurify).

Another popular defense against XSS attacks is using CSP (Content Security Policy). This browser security mechanism aims to disarm attacks by defining specific sources from which the JS code can be loaded. It can be configured with the specific option *“*[*script-src*](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/script-src)*”*, allowing you to specify either the same origin (aka “self”) or specific domains.

### Improper Restriction of XML External Entity Reference

Consider upgrading the JesperReports library used by the reporting generator functionality to a version greater than [6.20.1](https://github.com/TIBCOSoftware/jasperreports/releases/tag/6.20.1) where the iText 2.1.7 has been replaced with OpenPDF 1.3.30 for PDF export.

### Debug Messages Revealing Unnecessary Information

The customer should be able to address this issue by editing the JSP configuration inside the *\*WildFly* management console and specifically disabling the “X-Powered-By” header.

*\*WildFly: formerly known as JBoss AS, or simply JBoss, is an application server written by JBoss, now developed by Red Hat.*

### Missing rate limiting on the API backend

Consider implementing rate-limiting techniques for backend APIs whenever possible, particularly for sensitive functions like session login. This helps prevent possible brute-force attacks that could impact day-to-day performance. Additionally, deploying a robust \**Web Application Firewall (WAF)* at the backend acts as a gate, effectively blocking anomalies in traffic and common web attacks.

*\** *Web Application Firewall (WAF): A WAF or web application firewall helps protect web applications by filtering and monitoring HTTP traffic between a web application and the Internet.*