**Pentesting Report**

Table of Contents

[1 Executive summary 2](#_Toc167870037)

[2 Customer contacts 2](#_Toc167870038)

[3 IPS contacts 2](#_Toc167870039)

[4 Method 2](#_Toc167870040)

[5 Scope 3](#_Toc167870041)

[6 Assessment 3](#_Toc167870042)

[6.1 Summary of findings 3](#_Toc167870043)

[6.2 Identified Devices 4](#_Toc167870044)

[6.3 Overview 4](#_Toc167870045)

[7 Technical findings details 5](#_Toc167870046)

[7.1 *Improper authorization on BECS frontend* 5](#_Toc167870047)

[7.2 Dependency on Vulnerable Third-Party Component 8](#_Toc167870048)

[7.3 Use of Password Hash with Insufficient Computational Effort 8](#_Toc167870049)

[7.4 Cleartext Transmission of Sensitive Information 13](#_Toc167870050)

[7.5 Improperly Controlled Modification of Object Prototype At-tributes ('Prototype Pollution') 14](#_Toc167870051)

[7.6 Missing rate limiting 15](#_Toc167870052)

[8 Recommendations 15](#_Toc167870053)

[8.0 Overview 15](#_Toc167870054)

[8.1 Remedies 15](#_Toc167870055)

[8.1.1 Improper authorization on BECS frontend 15](#_Toc167870056)

[8.1.2 Dependency on Vulnerable Third-Party Component 15](#_Toc167870057)

[8.1.3 Use of Password Hash with Insufficient Computational Effort 16](#_Toc167870058)

[8.1.4 Cleartext Transmission of Sensitive Information 16](#_Toc167870059)

[8.1.5 Improperly Controlled Modification of Object Prototype At-tributes ('Prototype Pollution') 16](#_Toc167870060)

[8.1.6 Missing rate limiting 16](#_Toc167870061)

# Executive summary

PacketFront Software Solutions AB (“PacketFront Software” herein) contracted IP-Solutions to perform a Penetration Test of the [BECS](https://pfsw.com/becs/)\* 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 2024-05-14 to 2024-05-20 with a limited knowledge of the internal network and the BECS product itself. In addition, the customer provided a low-privilege 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 deter-mine 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 BECS 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* |
| Medium | 1 | Improper authorization on BECS frontend via the token becsrole. |
| Medium | 2 | Dependency on Vulnerable Third-Party Component as Jquery and Bootstrap framework vulnerable to XSS. |
| Medium | 3 | Use of Password Hash with Insufficient Computational Effort in xml config files. |
| Low | 4 | Cleartext Transmission of Sensitive Information via obsolete git remote port. |
| Low | 5 | Improperly Controlled Modification of Object Prototype Attributes ('Prototype Pollution') on RequireJS library. |
| Informational | 6 | Missing rate limiting on the API backend. |

## Identified Devices

|  |  |  |
| --- | --- | --- |
| *Hostname* | *IP* | *Details* |
| Becs-tomber-docker-pentest | 151.236.205.230 | BECS demo instance. |

## Overview

The tester found PacketFront’s BECS application to be well implemented where no critical finding has been discovered that led to an actual breach of the environment. 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 an improper authorization on the frontend web portal where the *becsrole* token could be edited to gain access to extra functions that otherwise would be visible only to administrative users.

The second finding was related to third-party libraries used by the application found to be outdated and vulnerable to *\*XSS* attacks.

The next issue was related to the use of weak password hash (*MD5*) exportable as part of a xml configuration file used by devices on the management portal. MD5 password hashes are notoriously weak and potentially crackable with enough computational power.

The use of a decommissioned git service that allows unencrypted communication being accessible by anyone on the same network is described in the fourth finding.

The fifth finding is about a *\*prototype pollution* vulnerability on all the RequireJS versions. 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.

*\*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.*

*\*Prototype pollution: Prototype pollution is a JavaScript vulnerability that enables an attacker to add arbitrary properties to global object prototypes, which may then be inherited by user-defined objects.*

# Technical findings details

## *Improper authorization on BECS frontend*

Severity: **MEDIUM**

The tester performed a login to the web portal accessible via *http://<customerip>:8089/* with the low privileges user’s credentials provided by the customer before the testing exercise.

A city skyline with a sunset

Description automatically generated

The login request is sent back to the backend API and upon successful processing a token is sent back to the frontend. This token looks something like this.A screenshot of a computer

Description automatically generated

The official documentation states that there should be 2 different roles assigned to a user either administrator or read-only which is our case.

A close-up of a computer screen

Description automatically generated

Then the tester tried to edit the cookie to elevate the permission by changing the token to match the user group administrators.

A screenshot of a computer

Description automatically generated

This change actually gives access to the logged-in user to extra menus in the website that otherwise weren’t available before.

Before token editing:

A screenshot of a computer

Description automatically generated

After token editing:

A screenshot of a computer

Description automatically generated

It is important to mention that this change isn’t enough to totally elevate the permissions of a low-privileged user. Every request initiated from the frontend is sent back & forth to the backend API where the *sessionid* is checked back for its permissions (our user held no write permissions aka read-only).

Several other tests were performed to elevate the permissions also on the backend without any success.

## Dependency on Vulnerable Third-Party Component

Severity: **MEDIUM**

The application uses several third-party libraries that are either outdated or suffer from documented vulnerabilities.

A white background with black lines

Description automatically generated

Specifically, the jQuery (ver. 1.12.1) is susceptible to several XSS (Cross-site scripting) vulnerabilities as stated here: <https://security.snyk.io/package/npm/jquery-ui/1.12.1>

Nonetheless the affected version is dated September/2016.

The other library Bootstrap framework (ver. 3.3.0) is susceptible to XSS (Cross-site scripting) vulnerabilities as well, as stated here: <https://security.snyk.io/package/npm/bootstrap/3.3.0>

Same applies for this library which is dated October/2014.

## Use of Password Hash with Insufficient Computational Effort

Severity: **MEDIUM**

The node import/export via XML file function accessible from the website which can be used to easily populate the structure hold password hashes of the managed devices (prior to this testing only one device was added to the portal).

The configuration can be exported easily via a wizard:

A screenshot of a computer

Description automatically generated

And the export file can be downloaded via the built-in file management function.

A screenshot of a computer

Description automatically generated

The XML configuration file holds the password hashes of the nodes saved in the management portal and the ssh username and many more.

A screenshot of a computer program

Description automatically generated

Specifically, there is a password hash saved by using a MD5(Crypt) hash type which is notoriously weak.

A screen shot of a computer

Description automatically generated

From an initial analysis seems like the built-in string encryption tool is used to generate the password hashes.

A screenshot of a computer

Description automatically generated

The first 2 hashes are generated via Openssl tool by using a preshared key that is saved locally on the backend.

A screenshot of a computer

Description automatically generated

Where instead the MD5(Crypt) seems generated by sending the command *openssl passwd -1 PASSWORD.* In the reproduction example a very short password(“a”) has been encrypted via the built-in function and successfully decrypted in matter of seconds.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Even if the specifical password exported from the node configuration hasn't been decrypted it is relevant to mention that with the appropriate computing power and in case of shorter passwords used this attack is likely feasible.

## Cleartext Transmission of Sensitive Information

Severity: **LOW**

The backend uses a deprecated git service running over the port 9418/TCP, originally used to remotely reach repositories on self-hosted git solutions.

Nowadays git providers strongly suggest avoiding this service as it is unencrypted and easily accessible by anyone being on the same network.

The repositories can be easily accessed without any authentication required:  
A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

Even if stated in the official documentation that this is part of the standard configuration used to manage HA (High-Availability) implementations, it is still possible to stumble upon password hashes seen in the previous paragraph.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## Improperly Controlled Modification of Object Prototype At-tributes ('Prototype Pollution')

Severity: **LOW**

The solution uses a third-party library RequireJS (vers. 1.12.1) that suffers from prototype pollution. Prototype pollution are a JavaScript vulnerabilities that enables an attacker to add arbitrary properties to global object prototypes, which may then be inherited by user-defined objects.

Reference: [https://security.snyk.io/vuln/SNYK-JS-REQUIREJS-5416713](https://security.snyk.io/vuln/SNYK-JS-REQUIREJS-5416713%20)

A white background with black lines

Description automatically generated

## Missing rate limiting

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, none of the issues found during testing resulted in critical privilege escalation or access to unrestricted data. However, 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 authorization on BECS frontend

Implementing token validation can prevent potential logic bugs in the frontend and avoid unintentionally exposing hidden functions on the website. One effective remediation approach is to implement JSON Web Tokens (JWT), which use signatures and encryption via public/private keys to protect against cookie tampering.

### Dependency on Vulnerable Third-Party Component

Maintaining third-party components is crucial for maintaining a strong security posture and preventing exploitation of known vulnerabilities by adversaries.

### Use of Password Hash with Insufficient Computational Effort

Avoid using password hashes generated by weak cryptographic functions like MD5, especially if they can be obtained from clear-text files. Instead, consider using stronger cryptographic functions such as SHA2 or BCRYPT. Additionally, enforce a robust password policy that includes at least 16 alphanumeric characters and, if feasible, implement password retention policies.

### Cleartext Transmission of Sensitive Information

Avoid using unencrypted services that might contain sensitive information. Specifically, consider implementing *\*Git* repositories that require authentication either with an account or SSH keys.

*\*Git: Git is a distributed version control system that tracks versions of files. It is often used to control source code by programmers collaboratively developing software.*

### Improperly Controlled Modification of Object Prototype At-tributes ('Prototype Pollution')

Although *\*prototype pollution* is often unexploitable as a standalone vulnerability, it lets an attacker control property of objects that would otherwise be inaccessible.

Please refer to the productor’s knowledge base of how to update to newer version that might fix this issue.

*\*Prototype pollution: Prototype pollution is a JavaScript vulnerability that enables an attacker to add arbitrary properties to global object prototypes, which may then be inherited by user-defined objects.*

### Missing rate limiting

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.*