From:
Antoine MOUTIER
Bastien ROBERT
Victor HUCKEL
Antonin LEDEE
Flavie ROBILLART

Smábanki platform security assessment

To:
Alexandre THIROUX

14/02/2024

Table of Content

Executive Summary

Ce rapport met en évidence les vulnérabilités identifiées sur le site smabanki.ovh et l'impact potentiel de leur exploitation sur la sécurité des données et des utilisateurs. L'évaluation réalisée a permis de détecter plusieurs failles critiques, notamment des problèmes liés à la confidentialité, à la gestion des virements ou encore à la gestion des communications. Chaque évaluation est accompagnée de recommandations afin de corriger les problèmes. L’adoption rapide de ces mesures est primordiale pour garantir la sécurité du site et préserver la confidentialité ainsi que l'intégrité des données qu'il héberge.

+ long

méthodo

vulns

conclusion

SMA-001 - Negative Transfer Vulnerability

Severity	Critical
Location	Payment Gateway / Transfer Endpoint
Impact	The system currently allows transferring funds with negative amounts, which can bypass balance checks and potentially enable unauthorized withdrawals.

Technical Details

When the amount is changed to a negative value in the transfer request, the transaction still goes through even if the source account has no funds. This indicates insufficient validation on the server side and can lead to fraudulent or inaccurate transactions, causing data inconsistencies.

Request	{ "sourceAccountId": "SMAJ730582", "recipientId": "3f41f250-1bb6-4b5a-9cec-a7a4826fa000", "description": "aa", "destinationAccountId": "SMAP71303847", "amount": "-784" }
Response	 The transaction is recorded as if it were valid, deducting a negative sum without verifying adequate funds.

Figure 1 : Request and Response during a negative transaction

Recommendation:

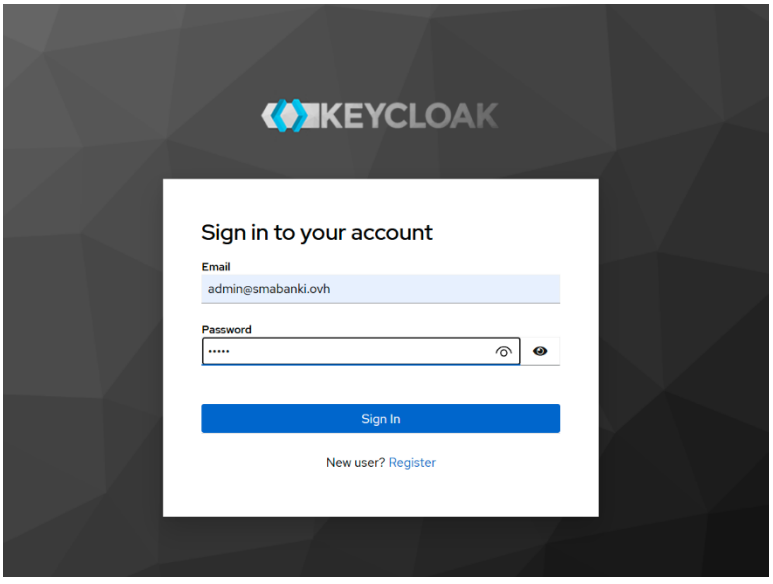
Strict checks should be put in place so negative amounts are rejected unless they're clearly allowed. Before any transfer, the system should confirm there's enough money in the source account. All attempts with negative amounts should be logged all attempts with negative amounts and alert the team. Finally, the security controls must be reviewed and updated to ensure no one can bypass these checks.

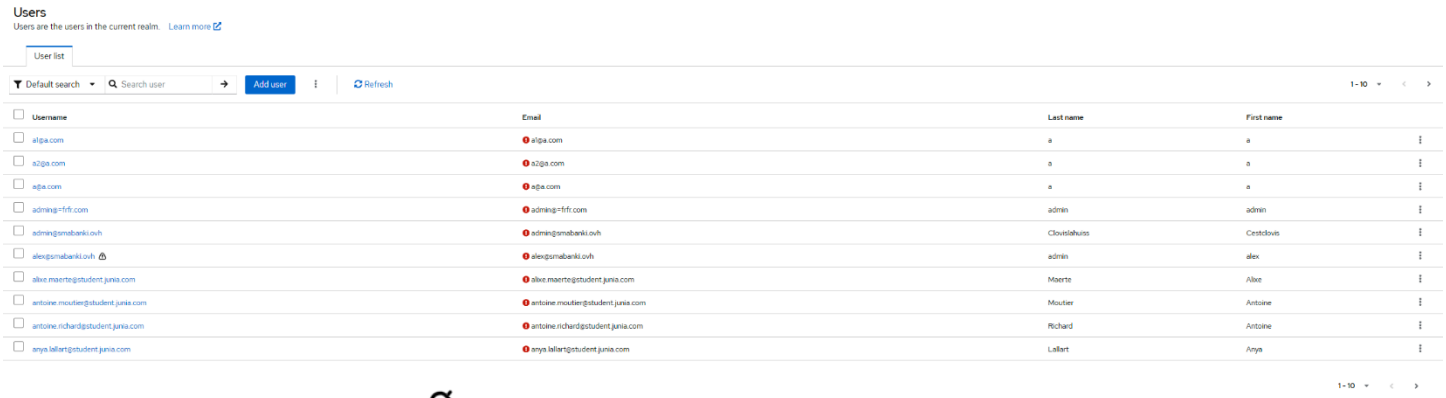
SMA-002 - Keycloak Admin Compromise

Severity	Critical
Location	identity.smabanki.ovh/admin/master/console/
Impact	The attacker can exploit security vulnerabilities to access, modify, or delete user data, potentially compromising sensitive information. We may also escalate privileges, granting unauthorized users admin rights and increasing control over the system. Additionally, they can inject a backdoor, allowing persistent access and enabling them to collect authentication data for future attacks.

Technical Details

We noticed that the admin email is publicly listed on the "About Us" page, making it easily accessible to anyone browsing the site. Additionally, the password is simply "admin," which is far too weak and predictable, allowing easy access to the admin account. We also observed that the system unnecessarily interacts with Keycloak during account creation, making requests even when just switching to the login page.





Recommendation:

We recommend strengthening security by changing the admin password to a more complex one, at least 10 characters long, including numbers and symbols. Admin email should also be hidden to prevent easy access. Additionally, modifying the account creation page could help avoid unnecessary interactions with Keycloak, reducing potential security risks.

SMA-003 - Insecure Communications

Severity	**Critical**
Location	~~website~~ N/A
Impact	The website uses the HTTP protocol instead of HTTPS, which exposes sensitive data such as credentials and tokens to interception. An attackers can perform Man-in-the-Middle attacks. This increases the risk of credential theft, session hijacking, and data manipulation.

Technical Details:

The production environment does not enforce HTTPS. Credentials and authentication tokens are transmitted over an insecure channel, making them vulnerable to sniffing.
And no HTTP Strict Transport Security is implemented, meaning users can still access the site over HTTP even if HTTPS is available.

Recommendation:

All HTTP traffic should be redirected to HTTPS. HTTP Strict Transport Security must be implemented to ensure users cannot access the site over an insecure connection.

SMA-004 - Path Traversal Vulnerability

Severity	**Critical**
Location	Controller/document.js :34 /api/documents/view Endpoint
Impact	The system allows unauthorized access to arbitrary files on the server by manipulating the uuid parameter. An attacker can exploit this vulnerability by reading sensitive system files, including authentication credentials, configuration files, and other confidential data. This can lead to data leakage, system compromise, and privilege escalation if critical files are exposed.

Technical Details

The endpoint /api/documents/view takes a Base64-encoded uuid parameter, which is decoded and directly used to construct a file path without proper validation. This allows a path traversal attack, enabling access to files outside the intended directory.

In this case, a request was made using Burp Suite with the following payload:

Request	GET /api/documents/view?uuid=Li4vLi4vLi4vLi4vLi4vLi4vLi4vLi4vZXRjL3Bhc3N3ZA== HTTP/1.1
Response	HTTP/1.1 200 OK Server: nginx/1.27.4 Date: Fri, 14 Feb 2025 15:39:19 GMT Connection: keep-alive X-Powered-By: Express Access-Control-Allow-Origin: * Content-Length: 1172

	cm9vdDp4OjA6MDpyb290Oi9yb290Oi9iaW4vYmFzaApkYWVtb246eDoxOjE6ZGFlbW9u Oi91c3Ivc2JpbjovdXNyL3NiaW4vbm9sb2dpbgpiaW46eDoyOjI6YmluOi9iaW46L3Vzci9z

This Base64-encoded string :

Li4vLi4vLi4vLi4vLi4vLi4vLi4vZXRjL3Bhc3N3ZA==

decodes to:

../../../../../../../../etc/passwd

As a result, the server responded with the contents of /etc/passwd, confirming that arbitrary file access is possible. The response contained system user information, proving that the attacker could read critical system files.

Recommendations:

The application must strictly validate and sanitize the uuid parameter before using it to construct file paths. It should reject any input containing ../ sequences to prevent directory traversal. A whitelist of allowed files should be enforced, ensuring that only expected and authorized files can be accessed. Additionally, secure path resolution methods such as path.join() with strict directory constraints should be used.