HACKMANIT

Security Analysis Report: *SSH CA*

Version: 1.3
24.02.2025

Conrad Schmidt
Phone: +49(0)234 / 54459996 | E-Mail: conrad.schmidt@hackmanit.de

Project Information

Customer:	WAYF / Danish e-Infrastructure Cooperation Asmussens Allé, Building 305 DK-2800 Lyngby, DENMARK
Contact:	Mads Freek Petersen
Commissioned to:	Hackmanit GmbH Universitätsstraße 60 (Exzenterhaus) 44789 Bochum, Germany
Project executive:	Conrad Schmidt Phone: +49(0)234 / 54459996 Fax: +49(0)234 / 54427593 E-Mail: conrad.schmidt@hackmanit.de

Initial Penetration Test

Project members:	Conrad Schmidt (Hackmanit GmbH) Niklas Conrad (Hackmanit GmbH) Karsten Meyer zu Selhausen (Hackmanit GmbH) Juraj Somorovsky (Hackmanit GmbH)
Project period:	

Retest

Project members:	Conrad Schmidt (Hackmanit GmbH) Juraj Somorovsky (Hackmanit GmbH)
Project period:	2024-10-28 – 2024-10-30

Version of the report: 1.3

This report was technically verified by Prof. Dr. Juraj Somorovsky.
This report was linguistically verified by Karsten Meyer zu Selhausen.

Contents

1 Summary

Hackmanit GmbH was commissioned by WAYF / Danish e-Infrastructure Cooperation to perform a security analysis of their application SSH CA. The analysis was performed remotely with a total expense of 7 man-days – including documentation and writing of this report. It consisted of a penetration test, as well as, a threat analysis, and a code review. Additionally a retest was performed for a total expense of 1,5 man-days.

Weaknesses. During our analysis we identified 14 potential security threats. Three of these threats lead to weaknesses which should be addressed; two rated as *Critical* and one rated as *High*.

The two weaknesses rated as *Critical* both describe a method in which an attacker could obtain an SSH certificate with the identity of a victim for their public keys – one time using a flaw in SSH CA and one time exploiting a flaw in MyAccessID. The weakness rated as *High* describes a denial-of-service (DoS) attack, in which an attacker could block access to the SSH CA for other users using simultaneous HTTP connections.

Top Weaknesses:

Risk Level	Finding	Reference
C01	CSRF Attack: Obtaining a Certificate With a Victim's Identity	Section 8.1, page 22
C02	CSRF Attack at MyAccessID: Obtaining a Certificate With a Victim's Identity	Section 8.2, page 24
H01	Denial-of-Service via Simultaneous HTTP Connections	Section 8.3, page 27

Recommended Actions. We recommend implementing the countermeasures described for each weakness, especially the ones rated as *Critical*, since they can be used by an attacker to gain access to a victim's account. For the third weakness rated as *High* countermeasures should be applied as well, since it could lead to a DoS attack.

Since C02 is not a weakness in SSH CA but rather in MyAccessID we recommend to forward the issue to MyAccessID to enable them to fix it on their side.

Retest. WAYF followed our recommendations and implemented countermeasures for every identified weakness, as well as the recommendation. We conducted a retest and can confirm that all identified weaknesses were successfully fixed.

For C02, WAYF contacted MyAccessID, since they had to implement a fix. Measures in form of a consent page were implemented from their side. They also followed our recommendation to remove the ability to skip the consent page in the device auth grant.

All weaknesses and recommendations have therefore been resolved.

Structure. The report is structured as follows: In Section 2, the timeline of the penetration test is listed. Section 3 introduces our methodology and Section 4 explains the general conditions and scope of the penetration test. In Section 5, the scenario of the security analysis is described in detail. Section 6 contains an analysis of threats affecting the SSH CA. Section 7 provides an overview of the identified weaknesses, as well as, further recommendations and information. In Section 8, all identified weaknesses are discussed in detail and specific countermeasures are described. Section 9 summarizes our recommendations resulting from observations of the application.

2 Project Timeline

The security analysis was carried out remotely from 2024-06-03 to 2024-06-07. The concept and implementation of the SSH CA developed by WAYF was examined by four people with the entire effort of 7 man-days – including documentation and writing of this report.

From 2024-10-28 to 2024-10-30, a retest of the described weaknesses was performed. Additionally a new feature, called “Global Unix Username” was added to the thread analysis. The retest was performed by one person with the effort of 1,5 man-days – including updating the report.

3 Methodology

Among others, the following tools were used for the penetration test:

Tool	Link
Mozilla Firefox	https://www.mozilla.org/de/firefox/
Google Chrome	https://www.google.com/intl/de_ALL/chrome/
Burp Suite Professional	https://portswigger.net/burp
Self-developed tools	-

Risk Rating. Each weakness has its own CVSS 3.1 base score rating (*Common Vulnerability Scoring System Version 3.1 Calculator*).[1,2] Based on the CVSS 3.1 base score, the following weaknesses assessment is performed:

0.0 – 3.9:	Low
4.0 – 6.9:	Medium
7.0 – 8.9:	High
9.0 – 10.0:	Critical

4 General Conditions and Scope

In the scope of the penetration test was the application “SSH CA” from WAYF. WAYF provided Hackmanit with the source code of a proof-of-concept application as well as access to a live test implementation.

The threat analysis, as well as the penetration test, was carried out for the test implementation accessible at the following URL:

https://sshca.deic.dk

[1] https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator
[2] https://www.first.org/cvss/v3.1/user-guide

For logging in with the default certificate authority we were provided with accounts for *Grand Unified* and additionally used existing accounts for logging in with *MyAccessID*.

To facilitate the threat analysis and penetration tests a code review of a proof-of-concept application was conducted. The source code was available in the following GitHub repository (commit: e8b33d6c83356c4b7cecc909110f4c01b33df9c9):

https://github.com/wayf-dk/sshca/

Part of the scope was the SSH CA web application as well as the SSH server that is part of the application. Although the login process carried out with the IdP was not part of the scope, the start of the single sign-on (SSO) workflow as well as the validation of the provided tokens, were tested.

5 Scenario Description

"SSH CA" developed by WAYF is a tool used to issue SSH certificates based on an identity provided via SSO. The general concept is as follows:

- The user chooses a certificate authority (CA) they want a certificate from.
- The user identifies themselves with an IdP trusted by the SSH CA.
- The user provides an SSH public key to the SSH CA.
- The SSH CA signs the public key with the desired CA and returns the SSH certificate to the user.
- The user can use this SSH certificate to connect to a server that supports it and trusts the issuing CA.

The SSH CA supports three different ways to authenticate and get an SSH certificate:

- Using the MyAccessID Python client[3]
- Using the "web first"-flow
- Using the "SSH first"-flow[4]

In the following sections every flow is explained separately.

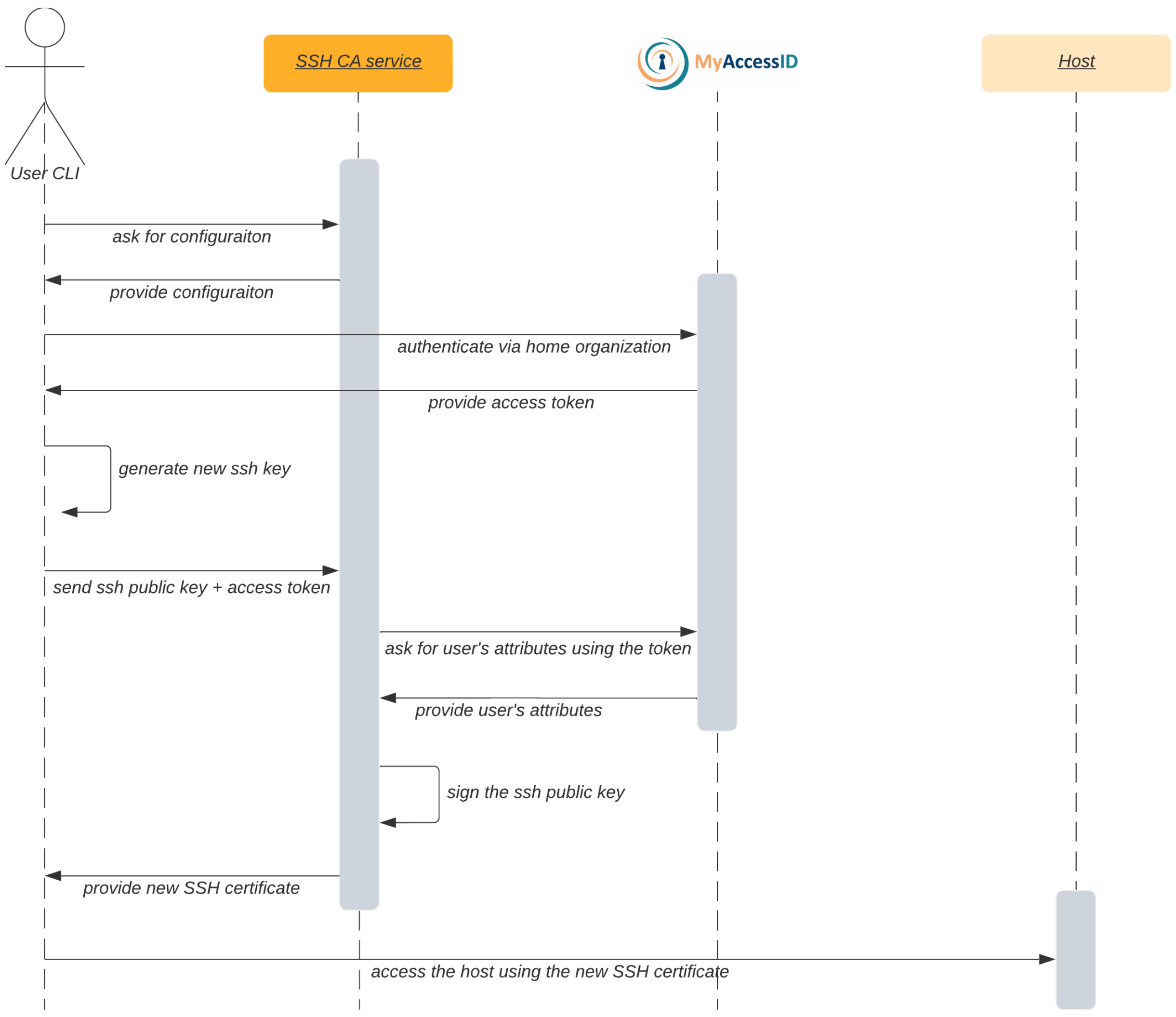

User CLI
SSH CA service
MyAccessID
Host
ask for configuraiton
provide configuraiton
authenticate via home organization
provide access token
generate new ssh key
send ssh public key + access token
ask for user's attributes using the token
provide user's attributes
sign the ssh public key
provide new SSH certificate
access the host using the new SSH certificate

Figure 1: Flow of the Python client. Image provided by WAYF.

5.1 Python Client Flow

MyAccessID provides a simple Python client for obtaining a valid SSH certificate using an identity from MyAccessID and a configurable CA. The client can be installed using `pip` and can be run using the command `generate_ssh_cert`. For this test we configured `https://sshca.deic.dk/MyAccessIdAcc` as the CA.

With the Python client the flow works as follows:

1. The user starts the flow by running the Python client CLI application.
2. The client asks the configured CA for configuration details.

[3] https://pypi.org/project/ssh-ca-cli/
[4] The “SSH first”-flow was removed after the initial test and can no longer be used.

3. The client starts a OAuth device authorization grant [1] with the configured IdP. The user authenticates at the IdP.
4. The IdP sends an access token to the client.
5. The client generates a new SSH key pair on the user's device.
6. The client sends the generated SSH public key alongside the access token to the SSH CA.
7. The SSH CA retrieves the user's attributes by calling the *UserInfo* endpoint of the IdP with the provided access token.
8. The SSH CA signs the provided SSH public key and sends the resulting SSH certificate back to the client.

5.2 "Web first"-flow

Using the "web first"- and "SSH first"-flow a user can use the SSH CA service without any client application, simply by connecting to an SSH server and using the web interface to authenticate using an IdP. The SSH CA uses different SSO flows depending on the chosen CA. These flows are labeled "Variant A" and "Variant B" in the flow charts.

The "web first"-flow works as follows:

1. The users visits the web page of the SSH CA application and chooses a CA.
2. The SSH CA generates a session token and provides the user with a command to start an SSH session containing the session token.
3. Authentication Variant A:
   a) The SSH CA starts the OAuth device authorization grant with the MyAccessID IdP.
   b) The user authenticates at the IdP.
   c) The IdP sends the access token to the SSH CA.
   d) The SSH CA retrieves the user's attributes by calling the *UserInfo* endpoint of the IdP with the provided access token.
4. Authentication Variant B:
   a) The SSH CA starts an OpenID Connect (OIDC) implicit flow with the Grand Unified IdP. The user authenticates at the IdP.
   b) The IdP sends an ID token to the SSH CA via a redirect in the user's browser.
   c) The SSH CA verifies the ID token's validity and extracts the user's identity.
5. The user starts an SSH connection using the command containing the session token provided by the SSH CA.
6. The SSH CA signs the public key the user sent along when connecting via SSH and returns the resulting SSH certificate via the SSH connection.

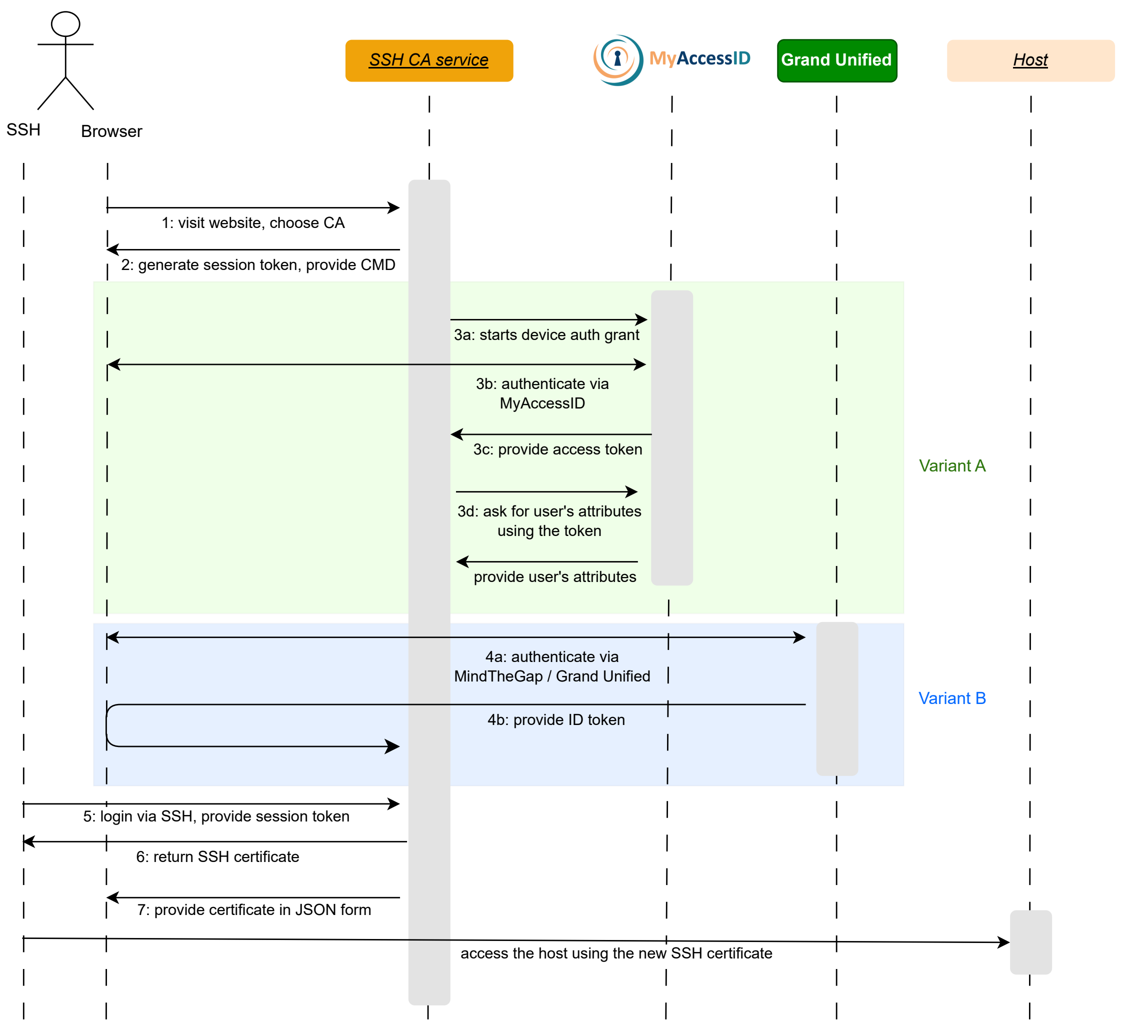

SSH
Browser
SSH CA service
MyAccessID
Grand Unified
Host
1: visit website, choose CA
2: generate session token, provide CMD
3a: starts device auth grant
3b: authenticate via MyAccessID
3c: provide access token
3d: ask for user's attributes using the token
provide user's attributes
Variant A
4a: authenticate via MindTheGap / Grand Unified
4b: provide ID token
Variant B
5: login via SSH, provide session token
6: return SSH certificate
7: provide certificate in JSON form
access the host using the new SSH certificate

Figure 2: "Web first"-flow. Image created by Hackmanit.

7. The SSH CA also provides a JSON version of the SSH certificate's contents in the user's browser.

5.3 "SSH first"-flow

The "SSH first"-flow is similar to the "web first"-flow. Instead of visiting a website, the user starts by establishing an SSH connection with the application first.

The "SSH first"-flow works as follows:

1. The users connects to the SSH server of the SSH CA application either choosing a CA and IdP directly or choosing it in the next step via the browser.

2. The SSH CA generates a session token and provides the user with a link containing the session token.
3. The user visits the URL and either has to choose a CA and an IdP or is directly redirected to the authentication at the IdP chosen in step 1.
4. Authentication Variant A:
   a) The SSH CA starts the OAuth device authorization grant with the MyAccessID IdP.
   b) The user authenticates at the IdP.
   c) The IdP sends the access token to the SSH CA.
   d) The SSH CA retrieves the user's attributes by calling the *UserInfo* endpoint of the IdP with the provided access token.
5. Authentication Variant B:
   a) The SSH CA starts an OIDC implicit flow with the Grand Unified IdP. The user authenticates at the IdP.
   b) The IdP sends an ID token to the SSH CA via a redirect in the user's browser.
   c) The SSH CA verifies the ID token's validity and extracts the user's identity.
6. The SSH CA signs the public key the user sent along when connecting via SSH and returns the resulting SSH certificate via the still open SSH connection.
7. The SSH CA also provides a JSON version of the SSH certificate's contents in the user's browser.

Retest. The "SSH first"-flow was removed after the initial test, to address C01.

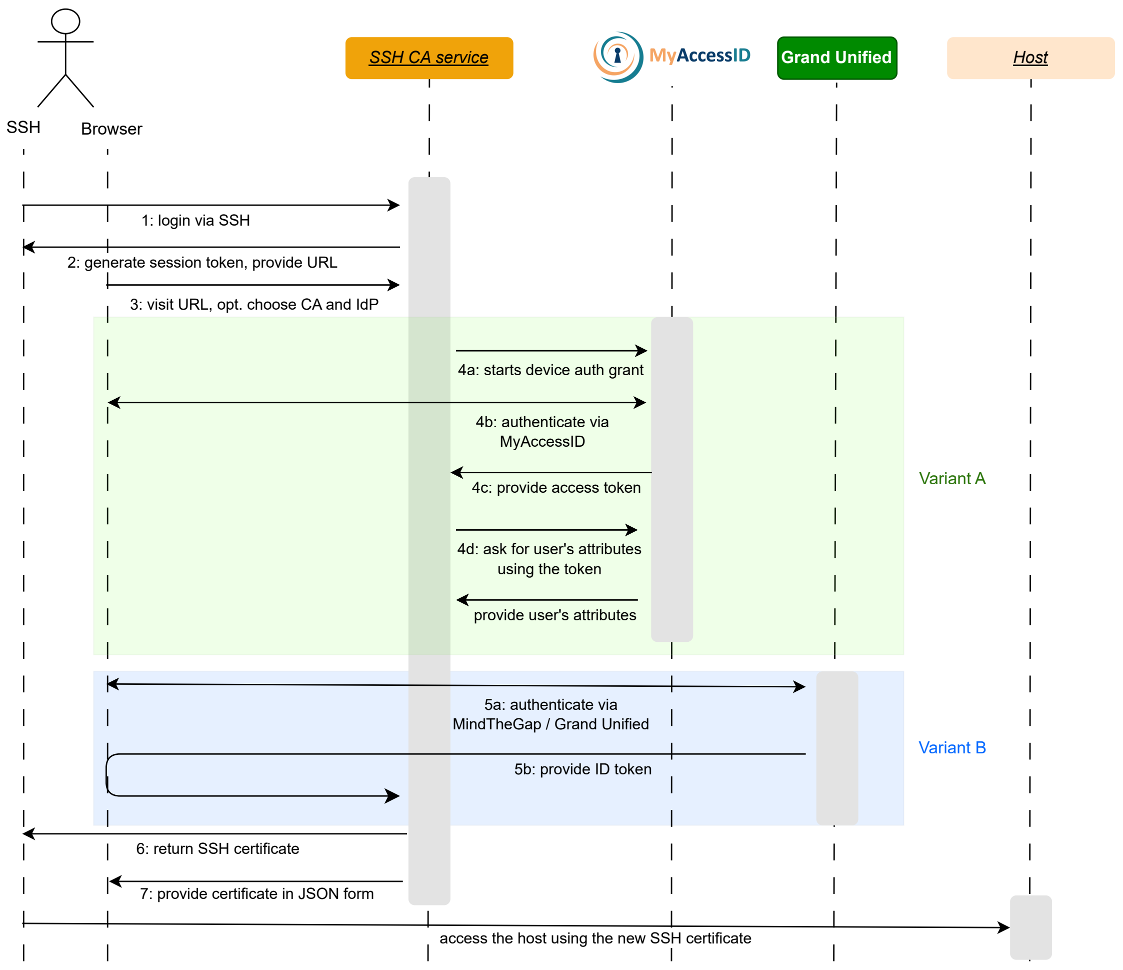

SSH
Browser
SSH CA service
MyAccessID
Grand Unified
Host
1: login via SSH
2: generate session token, provide URL
3: visit URL, opt. choose CA and IdP
4a: starts device auth grant
4b: authenticate via MyAccessID
4c: provide access token
4d: ask for user's attributes using the token
provide user's attributes
Variant A
5a: authenticate via MindTheGap / Grand Unified
5b: provide ID token
Variant B
6: return SSH certificate
7: provide certificate in JSON form
access the host using the new SSH certificate

Figure 3: “SSH first”-flow. Image created by Hackmanit.

6 Threat Analysis

6.1 Guessing Session Tokens

Threat Idea. An attacker could guess the session token of other users. This would result in the following attacks:

- In the first scenario, the victim would start the flow via the browser. The SSH CA service would generate a session token that the attacker guesses. The attacker would than connect via SSH before the victim does. When the victim finishes logging in, the attacker gets a signed certificate via SSH.

- In the second scenario, the victim would start the flow via browser or SSH. The attacker would again guess the generated session token and use the `/feedback` endpoint to get information about the victim, including its public key and subject information.

Analysis Result. After practical tests and a code review we concluded that guessing the session token is highly unlikely and therefore not a serious threat. The session token ist a cryptographically random chosen 8 byte code, has a lifetime of 30 seconds, and can only be used once.

Recommendation. *No recommended actions.*

6.2 Reusing a Session Token to Generate a Certificate

Threat Idea. In the "web first"-flow the SSH CA generates a session token that is used to continue the flow in an SSH connection. The SSH CA issues an SSH certificate for the public key used in this SSH connection after the user authenticated in the browser. If the user already authenticated, the SSH CA issues the SSH certificate immediately when the session token is provided in an SSH connection.

An attacker could try to reuse a stolen session token in a new SSH connection. If the SSH CA allows a session token to be used multiple times it might send an SSH certificate to the attacker in this SSH connection. Depending on the implementation the SSH certificate might either be the same certificate issued for the session token before or a new SSH certificate for the public key of the attacker. In both cases the attacker obtains a certificate without authenticating.

Analysis Result. After practical tests and a code review we concluded that it is not possible to use a session token multiple times. The SSH CA removes a session token after it has been used from its list of valid session tokens. Therefore, providing a session token a second time in an SSH connection results in the SSH CA rejecting the session token.

Recommendation. *No recommended actions.*

6.3 Reusing a Session Token to Obtain a Certificate's Contents

Threat Idea. When an SSH certificate is issued by the SSH CA it can be obtained using the corresponding session token in two different ways: Either the SSH certificate can be obtained directly via an SSH connection or information about the SSH certificate's contents can be obtained by calling the `/feedback` endpoint.

An attacker could try to call this endpoint using a stolen session token. If the SSH CA allows a session token to be used multiple times at this endpoint the contents of an SSH certificate would leak to the attacker. The information might contain private information about the victim.

Analysis Result. After practical tests and a code review we concluded that it is not possible to use a session token multiple times. The SSH CA removes a session token after it has been used from its list of valid session tokens. Therefore, calling the `/feedback` endpoint with a session token a second time results in the SSH CA rejecting the unknown session token and redirecting to the start page to start a new flow with a new session token.

Recommendation. *No recommended actions.*

6.4 IdP / CA Mix Up

Threat Analysis. An attacker could try to mix up IdPs and CAs to use an IdP for a CA not meant to be used with it. This could lead to a certificate signed by a CA with an identity controlled by an unauthorized IdP.

Analysis Result. We tried using the MyAccessID IdP for logging in with the "Default CA" CA as well as Grand Unified for logging in with the "MyAccessIDAcc" CA via "web first"- and "SSH first"-flows. Both did not succeed. We could not use MyAccessID as an IdP for the "Default CA" CA as only IdPs from a predefined list at `wayf.wayf.dk` were allowed. We could in term not use Grand Unified as an IdP for the "MyAccessIDAcc" CA, as the `idp` parameter was not allowed.

Recommendation. *No recommended actions.*

6.5 Using arbitrary IdPs

Threat Analysis. An attacker could try to use a malicious IdP under their own control to redirect victims to different web pages or to return fake identities to the SSH CA.

Analysis Result. It was not possible to use values other than the ones in the list of *MindTheGap* as `-idp` parameter in the SSH connection or `entityID` parameter in the web part of the flows, as both parameters are sent to `wayf.wayf.dk` for verification. Therefore, this attack is not possible.

Recommendation. *No recommended actions.*

6.6 Obtaining a Certificate With a Foreign Identity

Threat Idea. The SSH certificates issued by the SSH CA contain the identity of the user in the `ValidPrincipals` field. Servers which allow to connect using SSH certificates can use the `ValidPrincipals` field to authenticate the user and make authorization decisions. The SSH CA allows users to use different IdPs for authenticating.

If an attacker would be able to obtain an SSH certificate for their own public key with a victim's identity they could use this certificate to impersonate the victim when connecting to servers.

6.6.1 Obtaining a Certificate With a Foreign Identity by Making a Victim Authenticate at the SSH CA

Threat Idea. An attacker can try to leverage the fact that the flow for obtaining an SSH certificate is split into two parts: one web part and one SSH part. The attacker can start the flow by initiating an SSH connection and utilize a cross-site request forgery (CSRF) attack to make the victim execute the web part.

Analysis Result. During the practical tests it was possible to successfully execute a CSRF attack to obtain a certificate with a victim's identity. Details can be found in C01.

Recommendation. We recommend adding a mandatory user interaction to the flow to prevent the CSRF attack. *See Countermeasures in* C01 *for details.*

6.6.2 Obtaining a Certificate With a Foreign Identity Using the "Python Client"-Flow

Threat Idea. In the "Python Client"-flow with MyAccessID the OAuth device authorization grant [1] is used. In this grant the user needs to access a "verification URI" at the IdP and enter a "user code" before authenticating and authorizing the access. Instead of making the user enter the user code manually the grant also supports to access a different URI which already contains the user code.

An attacker can lure a victim into accessing a malicious website and execute the following attack: The attacker starts a device authorization grant and redirects the victim to the URI containing the user code. If the victim has an active session at the IdP and no consent-page is displayed they are automatically authenticated and access is granted. The attacker can now obtain an access token associated with the victim's identity. Afterwards, they can use this access token to obtain an SSH certificate with the victim's identity at the `/sign` endpoint of the SSH CA.

Analysis Result. During the practical tests it was possible to successfully execute a CSRF attack to obtain a certificate with a victim's identity. Details can be found in C02. However, the weakness itself is not rooted in the SSH CA but in the MyAccessID IdP, which is not in the scope of this analysis.

Recommendation. We recommend adding a mandatory user interaction when using the OAuth device authorization grant to prevent the CSRF attack. *See Countermeasures in* C02 *for details.*

6.6.3 Manipulating the Tokens to Contain an Arbitrary Identity

Threat Idea. When using the Grand Unified IdP an ID token is used to determine the user's identity. When using the MyAccessID IdP an access token is used retrieve the user's attributes from the IdP by calling the *UserInfo* endpoint. If the signatures of these tokens would not be verified correctly, an attacker could manipulate the payload of a token to contain an arbitrary identity. If a manipulated ID token is accepted by the SSH CA or a manipulated access token is accepted by MyAccessID IdP an attacker could obtain certificates for arbitrary identities.

Analysis Result. The practical tests showed that the signature of the tokens was verified correctly. Both the SSH CA and MyAccessID IdP recognized the invalid signature and returned an error for the following attacks:

- Invalidating the signature
- Using the `none` algorithm
- Server-side Request Forgery (SSRF) attack using the `jku` or `x5u` header parameters
- Using a custom key to sign the token

Recommendation. *No recommended actions.*

6.7 Denial-of-Service Attack via Multiple Open Connections

Threat Idea. An attacker could open multiple simultaneous connections to the SSH CA application either via HTTP or via SSH to exhaust the server's resources; thus executing a DoS attack. That way the server would not be able to accept new connections and would therefore be unusable by legitimate users. This attack leverages the fact, that an SSH connection with the application as well as an HTTP connection to the `/feedback` endpoint can stay open for up to 30 seconds, until the session token expires.

Analysis Result. In our analysis we did not manage to exhaust the server's resources with up to 300 simultaneous SSH connections. However, the attack could still be possible with more connections.

We did indeed succeed in blocking the server for legitimate users with around 200 simultaneous HTTP connections. Details can be found in H01.

Recommendation. We recommend adding the countermeasures described in H01 to prevent a DoS via HTTP. Additionally, we recommend not accepting simultaneous SSH connections from the same user to impede an attack via SSH.

6.8 Token Confusion

Threat Idea. The SSH CA supports different IdPs to authenticate the user. These IdPs issue tokens to the SSH CA. An attacker can try to use a token issued by one IdP to complete a flow started with a different IdP. This could result in an SSH certificate being issued for a wrong identity.

Analysis Result. The practical tests showed that the SSH CA does not accept tokens that have been issued by another IdP in another flow. Replacing the ID token issued by Grand Unified with an access token or ID token issued by MyAccessID IdP in the OIDC implicit flow resulted in an error message: `HTTP/1.1 400 Bad Request`

Sending an ID token issued by Grand Unified instead of an access token issued by MyAccessID IdP to the `/sign` endpoint resulted in an error message: `HTTP/1.1 500 Internal Server Error [...] no principal found`

Recommendation. *No recommended actions.*

6.9 Cross-Site Request Forgery (CSRF) Attacks Against the Single Sign-On Flows

Threat Idea. If the SSH CA does not correctly verify the `state` or `nonce` parameter used in the OIDC flows, an attacker can execute CSRFs attacks. These attacks might allow them to make the victim authenticate at the SSH CA without the victim interacting and afterwards leverage the victim's authenticated state for further attacks.

Analysis Result. The practical tests showed that the SSH CA correctly verified the `state` or `nonce` parameter. Any attempts to manipulate the value of these parameters resulted in an error message displayed by the SSH CA instead of a successful login of the victim.

Recommendation. *No recommended actions.*

6.10 Code Injection Into the Website

Threat Idea. If an attacker could manipulate data that gets reflected on the website of the SSH CA it could lead to a cross-site scripting (XSS) attack. Data that is injected into the website includes:

- the session token from the URL
- subject data from the IdP in the certificate's content information
- the public key from the user in the certificate's content information
- the `verification_uri_complete` when using the device authorization grant at the MyAccessID IdP

Analysis Result. We analyzed every potential point for code injections as mentioned above and found no weaknesses.

The analysis revealed that the session token cannot be used for injection attacks, as it cannot be created by the attacker or reflect arbitrary characters. The subject data, as well as the public key, could be manipulated but are injected into the website via JavaScript's `innerText` function, which provides sufficient protection against injection attacks. The `verification_uri_complete` field can be manipulated by a malicious IdP, however since the `html/template` package is being used, each text inserted into the template via Go is properly escaped.

Recommendation. *No recommended actions.*

6.11 Path Traversal at `GET /www/`

Threat Idea. A path traversal attack tries to access files outside of the web root folder. This would lead to information disclosure. In this case an attacker would try to change the path so that `http.ServeFileFS(w, r, Config.WWW, r.URL.Path)` (*sshca.go L.113*) would call an arbitrary file (e.g., `/etc/passwd`).

Analysis Result. The practical tests showed no successful way to open other files. Only when requesting `/www/index.html` the server responded with a 301 status code. WAYF confirmed to us, that there are no other files in the exposed directory and that the files are stored inside a container inside the binary file instead of the actual filesystem of the host.

Recommendation. As WAYF confirmed the use of a containerized filesystem stored inside the binary, this threat is already mitigated and no further action is necessary.

6.12 Collision of Global Unix Username

Threat Idea. SSH CA provides a "Global Unix Username" as a principal inside the certificate that is provided after the login. This username principal can be used by a service that trusts the SSH certificates issued by SSH CA. When a user connects to a supported host using the certificate, the host can use the provided username without having to do additional mapping or relog the user into its correct account name. This way, the users can log into different hosts using the same global username.

The global username provided by SSH CA is created differently depending on whether MyAccessID or another IdP, such as Grand Unified is used, as seen in Listing 1.

- **MyAccessID**: After logging in, MyAccessID provides an identity string in the form of `[uuid]@acc.myaccessid.org`.[5] To create the username, the uuid is used but without the dashes. E.g.: `c253971a-ab5c-401a-a976-6b8f0f17af3b@acc.myaccessid.org` becomes `c253971aab5c401aa9766b8f0f17af3b`.

[5]We assume that a UUID version 4 is used: https://en.wikipedia.org/wiki/Universally_unique_identifier#Version_4_(random)

- **Grand Unified**: After logging in with a different IdP, such as Grand Unified, an identity string in the form of `someuser@organisation.xyz` is provided. The global username is then generated by hashing the whole identity string with SHA256, trimming it to 24 Bytes and then encoding it with Base64Url. If the string starts with a dash, it is cut off. E.g.: `Conrad.Schmidt@hackmanit.de` becomes `BVmhgw6I3-pofQc-f7zh4-AC-pfMuxnT`. We call this method “HashedPrincipal”.

```go
  func usernameFromPrincipal(principal string, ca CaConfig) (username string) {
 if ca.HashedPrincipal {
  hashed := sha256.Sum256([]byte(principal))
  username = strings.TrimLeft(base64.RawURLEncoding.EncodeToString(hashed[:24]), "-")
      // - (dash) not allowed as 1st character
  return
 }
 if ca.MyAccessID {
  username = strings.ReplaceAll(principal[:36], "-", "")
  return
 }
 return
}
```

Listing 1: Code for generating the global usernames as seen in *sshca.go* from commit *fb4cce0*.

An attacker could try to generate a certificate with a global username in the principal, that is already connected to a different identity (collision of usernames). This way they could log into an account of a victim with the same global username as the attacker.

Analysis Result. A collision of usernames would be possible in two different ways:

1. Two users using HashedPrincipals with different identities get the same stripped hash value and therefore the same username.
2. A user using MyAccessID gets the same username as a user using HashedPrincipals.

For **scenario 1**, SHA256 is used as a hashing algorithm, which is generally considered safe for cryptographic use. For a hash function with an output of n bits, finding a collision (two distinct inputs that produce the same hash output) generally requires $2^{\frac{n}{2}}$ operations due to the birthday paradox.[6] Since in this case, the hash output length is reduced, from 256 bits to 192 bits, the work required to find a collision would also be reduced, requiring only 2^{96} operations. This makes the hash weaker against collision attacks, but at this size still safe enough. However, the users do not have the ability to change their identity strings, making a brute force attack impossible. A collision could therefore only happen by chance, which would be nearly impossible with the potential number of users of the application. Dropping the first character if it is a dash does not alter the chances either, since it just creates a shorter username that can not be created using a different method.

In **scenario 2**, a user using HashedPrincipals would have to have a username, that is by chance in the same format as a username generated through MyAccessID. This means, it can only contain the letters a to f and numbers, and it can not contain – or _. If we assume that enough users of that schema exist, it would still be necessary that the MyAccessID UUID collides with one of them. The total length of random bits in a UUID is 122, making an attack possible with

[6]https://en.wikipedia.org/wiki/Birthday_problem

2^{61} operations. Since the UUIDs by MyAccessID cannot be chosen by the users a brute force attack is not possible and a collision could only happen by chance. Since the number of users is much lower than the needed 2^{61}, this is nearly impossible.

It should also be noted, that as stated by WAYF, CAs are not meant to be used by both MyAccessID and other IdPs using HashedPrincipal. This scenario should therefore not be possible in practice.

We therefore conclude that a collision of usernames is not feasible.

Recommendation. *No recommended actions.*

7 Overview of Weaknesses, Recommendations, and Information

Risk Level	Finding	Reference
C01	**CSRF Attack: Obtaining a Certificate With a Victim's Identity:** An attacker can obtain an SSH certificate for a victim's identity and use this certificate to impersonate the victim.	Section 8.1, page 22
C02	**CSRF Attack at MyAccessID: Obtaining a Certificate With a Victim's Identity:** An attacker can obtain an SSH certificate for a victim's identity and use this certificate to impersonate the victim.	Section 8.2, page 24
H01	**Denial-of-Service via Simultaneous HTTP Connections:** An attacker is able to block legitimate users from accessing the SSH CA using simultaneous HTTP connections.	Section 8.3, page 27
R01	**Information Disclosure: Server Header:** It is advised to hide information such as the `Server` header in HTTP responses as this information might be helpful for attackers.	Section 9.1, page 29

Risk Definitions:

Critical Risk	Weaknesses classified as *Critical* can be exploited with very little effort by an attacker. They have very large negative effects on the tested system, its users and data, or the system environment.
High Risk	Weaknesses classified as *High* can be exploited with little effort by an attacker. They have a major negative impact on the tested system, its users and data, or the system environment.
Medium Risk	Weaknesses classified as *Medium* can be exploited with medium effort by an attacker. They have a medium negative impact on the tested system, its users and data, or the system environment.
Low Risk	Weaknesses classified as *Low* can only be exploited with great effort by an attacker. They have little negative impact on the tested system, its users and data, or the system environment.
Recommendation	*Recommendation* identifies measures that may increase the security of the tested system. Implementation is recommended, but not necessarily required.
Information	Observations classified as *Information* are usually no weaknesses. Examples of these observations are unusual configurations and possibly unwanted behavior of the tested system.

8 Weaknesses

In the following sections, we list the identified weaknesses. Every weakness has an identification name which can be used as a reference in the event of questions, or during the patching phase.

8.1 C01 CSRF Attack: Obtaining a Certificate With a Victim's Identity

Exploitability Metrics		**Impact Metrics**	
Attack Vector (AV)	**Network**	Confidentiality Impact (C)	**High**
Attack Complexity (AC)	**Low**	Integrity Impact (I)	**High**
Privileges Required (PR)	**None**	Availability Impact (A)	**High**
User Interaction (UI)	**Required**	Scope (S)	**Changed**
Subscore: **2.8**		Subscore: **6.0**	
Overall CVSS Score for C01: 9.6			

Note: *The specific impact of this weakness on confidentiality, integrity, and availability highly depends on the servers that can be accessed with the victim's identity. This CVSS score represents the worst case.*

General Description. The SSH CA uses the claim `eduPersonPrincipalName` provided by the IdP as the user's identity. When issuing the SSH certificate the SSH CA adds this information as the `ValidPrincipals` field to the certificate.

Servers which allow to connect using SSH certificates can use the `ValidPrincipals` field to authenticate the user and make authorization decisions.

Weakness. An attacker can leverage a CSRF attack to obtain an SSH certificate for their own public key with a victim's identity. The attack works as follows:

1. The attacker lures a victim into accessing a website controlled by the attacker.
2. The attacker starts the "SSH first"-flow with their own SSH keys.[7] The SSH CA responds with an URL for the second part of the flow in the browser (e.g., `https://sshca.deic.dk/285048f20e020ffd`).
3. Instead of accessing the provided URL themselves the attacker redirects the victim to this URL.
4. The victim needs to be authenticated by the IdP *(see more details below)*.
5. After the victim has been successfully authenticated, the SSH CA issues an SSH certificate containing the victim's identity. The SSH certificate is issued for the attacker's public key as this key was used to establish the SSH connection in step 2.
6. The SSH CA send the SSH certificate to the attacker using the SSH connection established in step 2.

[7] The attacker might select a specific CA and IdP using the `-ca` and `-idp` command line flags.

7. The attacker can use the SSH certificate to authenticate to servers and impersonate the victim.

The crucial part of the attack is step 4 – the authentication of the victim.

As a prerequisite we assume that the victim has an active session at the IdP and has used the SSH CA before.[8] Therefore, the victim does not need to log in during the attack. Depending on the implementation and policy of the IdP it might be necessary for the victim to actively confirm the authentication to SSH CA on a consent-page. However, for the two IdPs used during this analysis (Grand Unified and MyAccessID) there is no consent-page displayed when SSH CA has been used before. This means step 4 looks as follows when using these IdPs:

Grand Unified: When the victim is redirected to the URL of the SSH CA it is redirected multiple times automatically without any interaction.[9] The victim is authenticated and the attack proceeds as described above.

MyAccessID: When the victim is redirected to the URL of the SSH CA a popup for MyAccessID is automatically opened.[10] As the victim has an active session at MyAccessID they are automatically authenticated and no interaction by the victim is necessary. The popup remains open and states "You can close this browser tab and return to the application"; the attack proceeds as described above. While the victim might be surprised by the redirect to SSH CA and the popup window, it is already too late to stop the attack at this point.

Countermeasures. We recommend adding a mandatory user interaction to the flow to prevent the CSRF attack. Two possible countermeasures are:

1. **Add a "confirmation token" to retrieve the SSH certificate:** Instead of sending the SSH certificate to the initiator of the SSH connection automatically, a manual step should be added. After the user authenticated the website should display a one-time usable "confirmation token" (different from the session token used before). This "confirmation token" needs to be provided in the SSH connection to retrieve the SSH certificate. This strengthens the binding between the web browser and the SSH connection.

2. **Add a confirmation step in the web part:** When accessing the SSH CA a manual confirmation that the user indeed wants to create an SSH certificate containing their identity should be added. The usage of a consent-page is in the responsibility of the IdP and cannot be enforced by the SSH CA.[11] Therefore to ensure that user interaction is mandatory, the SSH CA itself should require interaction of the user.

Retest. To mitigate this problem, the "SSH first"-flow was removed from the application. Users now **have** to use the "Web first"-flow.

We can therefore confirm that the described attack is no longer possible, since the attacker had to start the flow by sending their SSH key first.

[8] This is not a strong assumption. The attacker will only choose a victim if it is authorized to access servers which the attacker wants to access. It is likely that the people who are authorized to access the servers are actually accessing the servers and therefore need to use the SSH CA themselves.
[9] The attacker would have selected the "Default CA" CA and the Grand Unified IdP in step 2.
[10] The attacker would have selected the "MyAccessIDAcc" CA in step 2.
[11] The SSH CA could add `prompt=consent` to the authorization request, if the IdP supports this parameter. This requests the IdP to actively ask the user for consent. More information about the `prompt` parameter can be found here: https://openid.net/specs/openid-connect-core-1_0.html#AuthRequest

8.2 C02 CSRF Attack at MyAccessID: Obtaining a Certificate With a Victim's Identity

Exploitability Metrics		Impact Metrics	
Attack Vector (AV)	**Network**	Confidentiality Impact (C)	**High**
Attack Complexity (AC)	**Low**	Integrity Impact (I)	**High**
Privileges Required (PR)	**None**	Availability Impact (A)	**High**
User Interaction (UI)	**Required**	Scope (S)	**Changed**
Subscore: **2.8**		Subscore: **6.0**	

Overall CVSS Score for C02: 9.6

Note: *The specific impact of this weakness on confidentiality, integrity, and availability highly depends on the servers that can be accessed with the victim's identity. This CVSS score represents the worst case.*

General Description. The SSH CA uses the claim `eduPersonPrincipalName` provided by the IdP as the user's identity. When issuing the SSH certificate the SSH CA adds this information as the `ValidPrincipals` field to the certificate.

Servers which allow to connect using SSH certificates can use the `ValidPrincipals` field to authenticate the user and make authorization decisions.

When using the "Python Client"-flow (see Figure 1) the python client uses the OAuth device authorization grant [1] to obtain an access token at the MyAccessID IdP. This access token is then sent to the `/sign` endpoint of the SSH CA. The SSH CA uses the access token to retrieve the user's attributes at the *UserInfo* endpoint of the MyAccessID IdP.

Weakness. An attacker can leverage a CSRF attack to obtain an SSH certificate for their own public key with a victim's identity. The attack works as follows:

1. The attacker lures a victim into accessing a website controlled by the attacker.
2. The attacker starts a device authorization grant at the MyAccessID IdP. The attacker receives an URL containing the verification URI and the user code that needs to be accessed in the browser to authenticate and authorize the access (e.g., `https://webapp.acc.myaccessid.org/oidc/verify?user_code=MLZ601A8N`).
3. Instead of accessing the provided URL themselves the attacker redirects the victim to this URL.
4. The victim needs to be authenticated by the IdP *(see more details below)*.
5. After the victim has been successfully authenticated, the attacker can retrieve an access token at the MyAccessID IdP.
6. The attacker sends this access token along with their own SSH public key to the `/sign` endpoint of the SSH CA.
7. The SSH CA issues an SSH certificate containing the victim's identity for the attacker's public key and sends the SSH certificate to the attacker.

8. The attacker can use the SSH certificate to authenticate to servers and impersonate the victim.

The crucial part of the attack is step 4 – the authentication of the victim.

As a prerequisite we assume that the victim has an active session at the IdP and has used the SSH CA before.[12] Therefore, the victim does not need to log in during the attack. Depending on the implementation and policy of the IdP it might be necessary for the victim to actively confirm the authentication to SSH CA on a consent-page. However, the MyAccessID IdP does not display a consent-page if SSH CA has been used before by the victim. This means step 4 does not need any user interaction and does only consist of multiple redirects which are executed automatically.

Note: The attack described above does not exploit a weakness in the SSH CA itself. The attack is rooted in a general weakness in the MyAccessID IdP. Thus, the MyAccessID IdP is responsible for applying countermeasures. The CSRF attack allows an attacker to obtain tokens associated with the victim's identity. These tokens can be used to impersonate the victim at any client or resource that trusts the MyAccessID IdP.

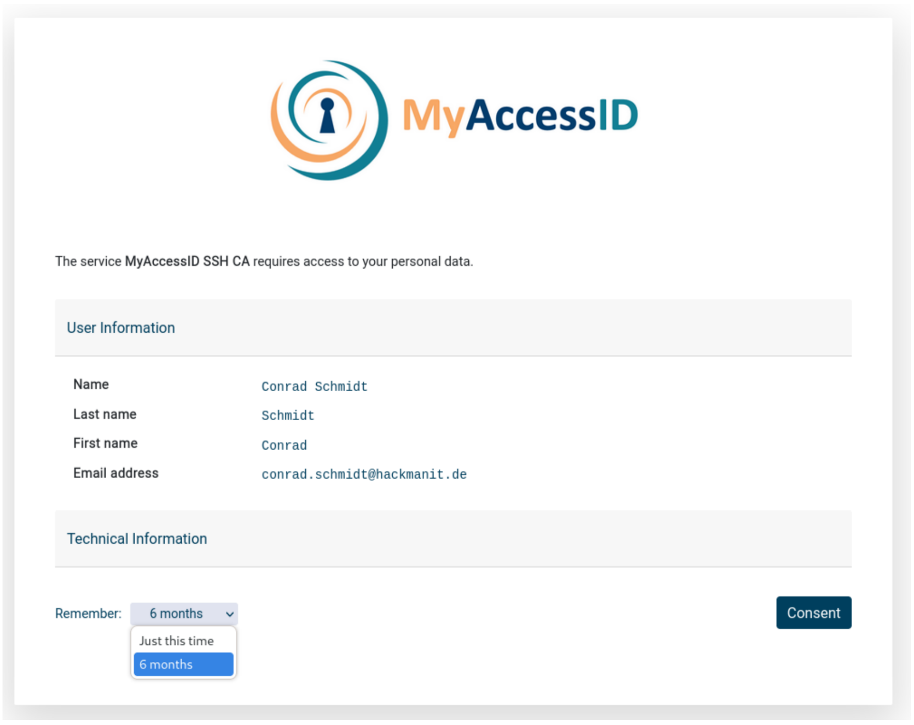
Countermeasures. We recommend adding a mandatory user interaction to the flow to prevent the CSRF attack. MyAccessID IdP should **always** display a consent page when the device authorization grant is used to ensure the user has actively participated in the flow and has given explicit consent for the authorization/login.

Retest. We can confirm that MyAccessID does display a consent page now. This consent page initially had a button to remember this choice for 6 months, as seen in Figure 4. The attack would therefore still be possible if a user selected this option. Since we assume that the victim already uses the SSH CA for themselves, it is likely that they would choose to remember their consent to prevent repeating interruptions by the consent page. However, this would mean that an attacker could still exploit the weakness.

We therefore recommend to remove the button to remember the consent decision for the device authorization grant.

MyAccessID followed this recommendation as seen in Figure 5. This weakness is therefore successfully fixed.

[12] This is not a strong assumption. The attacker will only choose a victim if it is authorized to access servers which the attacker wants to access. It is likely that the people who are authorized to access the servers are actually accessing the servers and therefore need to use the SSH CA themselves.



The service **MyAccessID SSH CA** requires access to your personal data.

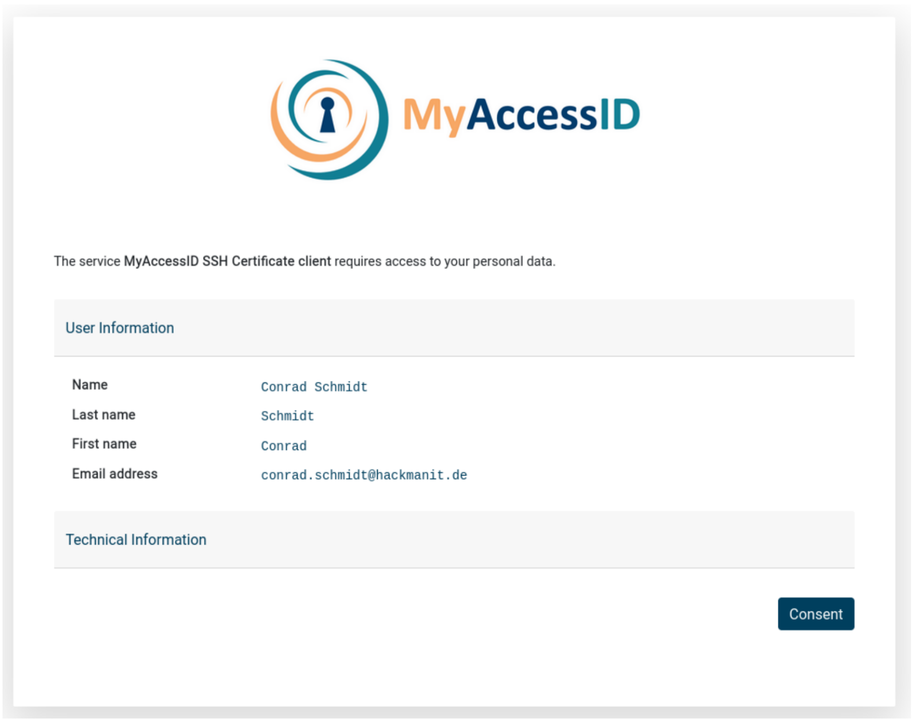
User Information

Name	`Conrad Schmidt`
Last name	`Schmidt`
First name	`Conrad`
Email address	`conrad.schmidt@hackmanit.de`

Technical Information

Remember: 6 months

Figure 4: Consent page of MyAccessID, with the choice to remember 6 months.



The service **MyAccessID SSH Certificate client** requires access to your personal data.

User Information

Name	`Conrad Schmidt`
Last name	`Schmidt`
First name	`Conrad`
Email address	`conrad.schmidt@hackmanit.de`

Technical Information

Figure 5: Consent page of MyAccessID for the device auth grant, after removing the remember-functionality.

8.3 H01 Denial-of-Service via Simultaneous HTTP Connections

Exploitability Metrics		**Impact Metrics**	
Attack Vector (AV)	**Network**	Confidentiality Impact (C)	**None**
Attack Complexity (AC)	**Low**	Integrity Impact (I)	**None**
Privileges Required (PR)	**None**	Availability Impact (A)	**High**
User Interaction (UI)	**None**	Scope (S)	**Unchanged**
Subscore: **3.9**		Subscore: **3.6**	

Overall CVSS Score for H01: 7.5

General Description. Denial-of-service (DoS) attacks aim to negatively influence the accessibility or use of a target system. Overloading the target system can be achieved in various ways. In general, an attempt is made to exhaust one or more resources of the target system. These resources can be, for example, the processing power of the CPU, the RAM or hard disk storage, or the network bandwidth.

In the case of a successful DoS attack, other users cannot access or use the target system, or only to a limited extent. Depending on the type of target system, DoS attacks can have serious consequences. For example, in the case of an online store, high financial losses may occur if the customers are unable to make purchases.

Weakness. This attack leverages the fact, that a request to the `/feedback` endpoint of the SSH CA service can take up to 30 seconds to return a response, since the application is waiting for the login to succeed or the session token to expire. A web server is restricted in the number of simultaneous connections it can accept by the maximum number of threads it can create. An attacker can open as many connections as the server can simultaneously handle, thus preventing any new connections being made by legitimate users.

To exploit this weakness the attacker has to first create a valid session token either via the “web first”- or the “ssh first”-flow. They can than use this session token to make many simultaneous GET requests to `/feedback/<token>`. Since the attacker never completes the login, those connections stay open for up to 30 seconds. Afterwards, the attacker can launch a new attack to continuously block other connections to the server.

We tested the attack and can confirm, that new HTTP connections to the server could not be made, if around 200 open connections already exist. This attack is an easy to execute DoS attack, since there are very few resources needed by the attacker. It can even be carried out by a single computer.

Countermeasures. We recommend adjusting the `/feedback` endpoint to a polling based behavior. This means returning an answer immediately, either the result or that the status is still pending. With this approach the client would have to send the request again until a proper response is delivered. This will ensure that no HTTP connection is held open for an extended period of time.

Another approach would be to limit the number of open connection per token and per IP address.

Retest. We can confirm that the recommendation was successfully implemented, and the weakness is therefore fixed. The application now uses polling to get the result, with a request every few seconds, until the SSH connection has been established.

9 Recommendations

In the following sections, we provide our recommendations to improve the security of the tested system.

9.1 R01 Information Disclosure: Server Header

General Description. Information disclosure describes an attack class where an application leaks sensitive information to an attacker. This information can include internal or technical details about the environment and functionality of the application or about frameworks and libraries (incl. their versions) used by the application. There are different ways how internal information can leak to attackers; examples are verbose error messages or unnecessary HTTP headers. The leaked information may be valuable for an attacker on its own (such as a password), or it may be useful for launching other, more severe, attacks.

The SSH CA application sends a `Server` header in its HTTP responses, as seen in Listing 2.

```
HTTP/2 200 OK
Date: Mon, 03 Jun 2024 11:35:35 GMT
Server: Apache/2.4.52 (Ubuntu)
Content-Security-Policy: frame-ancestors 'self'
Strict-Transport-Security: max-age=31536000; includeSubDomains
X-Content-Type-Options: nosniff
X-Frame-Options: sameorigin
X-Xss-Protection: 0
Content-Type: text/plain; charset=utf-8
Vary: Accept-Encoding
Content-Length: 686
Set-Cookie: sshcaid=sshca-qa; path=/; HttpOnly; Secure
```

Listing 2: Example of HTTP headers in a response from the SSH CA.

Recommendation. We recommend removing the `Server` header from the HTTP responses to not reveal the software and version the server is using.

Retest. We can confirm that the recommendation was successfully implemented. The `Server` header is no longer shown.

10 References

[1] W. Denniss et al. *OAuth 2.0 Device Authorization Grant*. RFC 8628 (Proposed Standard). Internet Engineering Task Force, Aug. 2019. url: https://tools.ietf.org/html/rfc8628.