Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest-Report Keycloak 8.0 Audit & Pentest 11.2019

Cure53, Dr.-Ing. M. Heiderich, M. Rupp, Dr. N. Kobeissi, BSc. C. Kean, MSc. S. Moritz, B. Walny, BSc. T.-C. Hong

Index

Introduction

"Keycloak is an open source Identity and Access Management solution aimed at modern applications and services. It makes it easy to secure applications and services with little to no code."

From https://www.keycloak.org/about.html

This report documents the results of a security assessment targeting the Keycloak 8.0 complex. Carried out by Cure53 in November 2019, this project entailed a source-code assisted penetration test and a broader review of the security posture. It must be underscored that the assessment was initiated and funded by REWE Digital GmbH.

In terms of scope, Cure53 was tasked with examining the Keycloak software and its running instance that was spun up by REWE Digital GmbH for the Cure53 team to test on. The Keycloak team was also involved in the project, especially since Cure53 set up a Slack channel for all involved parties. On Slack, the Cure53 team, the Keycloak team and the REWE team could not only delineate the bounds of this project, but later used it to discuss progress and emerging findings during the actual assessment process.

It can be clarified that the methodology used in this project was essentially a white-box approach. This is due to the nature of the project in that Keycloak is available as Open Source Software. To further facilitate completion of this investigation, Cure53 was given an environment on GCP, being granted ownership privileges for testing. In addition, Cure53 utilized a new version of the RIPS source code scanning platform to inspect the Java sources and see if static code analysis translates to finding more issues.

Addressing the resources, the total time budget available for the project stood at twenty-one days. The majority of the time allocated to this assessment was invested into manual code analysis and penetration testing, while one day was dedicated to scanning of the Keycloak sources and evaluating the results. A team of seven Cure53 testers was comprised to perform the tests and audits.

To best address various objectives, this assessment of the Keycloak was divided into three work packages (WPs). In WP1, the focus was placed on classic web security issues. Consequently, Cure53 inspected the application and code for classic injections, such as SQLi, XSS, as well as in regard to RCE, deserialization attacks, arbitrary file read, SSRF, CSRF issues and XML-parsing bugs. Next, in WP2, Cure53 zoomed in on the ACL, RBAC and privilege escalation flaws. The main tasks revolved around proper enforcement of access rules, as well as object ownership and visibility - both across different users and across different types of users. Finally, cryptography and information flow took center stage in WP3. In this work package, Cure53 audited and inspected the cryptographic code and primitives used by Keycloak. Furthermore, protocols and information flow during several types of authorization processes were analyzed.

All work took place in late November 2019. Specifically, the scope was prepared in Calendar Week (CW) 46, executed in CW47 and finalized in CW48 2019. Cure53 kept the REWE and Keycloak teams up to date regarding the test status in the aforementioned Slack channel. Moreover, the testers live-reported tickets into the bug tracker made available by Keycloak. Over the course of the project, all involved parties were exchanging questions and feedback in a prompt and productive fashion.

Ten security-relevant findings surfaced during this Cure53 assessment. The spotted problems were classified as five actual vulnerabilities and five general weaknesses with lower exploitation potential. One finding, a persistent XSS in the Keycloak client settings page, was given a *High* severity score. All other flaws were considered to represent less pressing risks, receiving *Low* or even just *Informational* marks. It is also worth noting that the majority of the identified problems are trivially easy to fix. Foreshadowing the conclusions, this can be seen as quite a good result. However, it needs to be stated that Keycloak is an incredibly complex software with vast amounts of features. Thus, a complete coverage of all features and sources could not be reached in the given timeframe. While the existence of unidentified risks on the scope cannot be excluded, the impression made by security posture of Keycloak is nevertheless very positive.

In the following sections, the report will first present the areas featured in this project's scope in more detail. It then moves on to dedicated, chronologically discussed tickets, which present the discoveries one-by-one. Alongside technical aspects like PoCs, Cure53 furnishes mitigation advice, so to ascertain that the Keycloak and REWE Digital GmbH teams can move forward with all necessary information at hand. The report closes with a conclusion in which Cure53 summarizes this autumn 2019 project and issues a verdict about the security premise of the investigated Keycloak 8.0 scope.

Scope

- **Keycloak Open Source Version as set up by REWE Digital:**
  - Running Keycloak 8.0 Environment was provided for Cure53
    - *URL removed from public report*
  - Test-users with *admin* role were provided for Cure53
  - Test-users with other roles were created by Cure53
- **Relevant sources were available on GitHub:**
  - https://github.com/keycloak/keycloak/tree/8.0.0
- **Documentation was available at:**
  - https://www.keycloak.org/documentation.html

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *KC-01-001*) for the purpose of facilitating any future follow-up correspondence.

KC-01-001 Web: Stored XSS in *Client settings* via *application links* (*High*)

During the assessment of the Admin Console application, it was found that links to external applications, which are called *application links*, do not get validated properly. Therefore, they are prone to Stored XSS attacks. The affected *BaseURL* parameter within the *Clients settings* page reachable from the Admin Console application accepts any characters, making it possible to insert URLs with the *javascript:* scheme at the beginning. AngularJS mitigates the attack and prepends *unsafe:* within the Admin Console application. However, it turns out that the account page (see *PoC #1*) and the error page (see *PoC #2*) fail to handle it properly.

Thus, a malicious user with permissions to create *application links* within the *clients settings* page has the capacity to forge JavaScript URLs to trick users from other realms. Under some circumstances, it is possible to change the email from the admin account to get access to the Admin Console of the master level (see *PoC #3*).

Affected Request (add JavaScript URL):
```
PUT /auth/admin/realms/test/clients/a79caaa0-51f7-4113-8c7b-8467df17591b
HTTP/1.1
Host: auth-service.test.example.com
Content-Type: application/json;charset=utf-8
Authorization: Bearer [...]
```

```
{"id":"a79caaa0-51f7-4113-8c7b-
8467df17591b","clientId":"test","rootUrl":"","adminUrl":"","baseUrl":"javascript
:alert()","surrogateAuthRequired":false,"enabled":true[...]
```

PoC #1 (via account settings):
https://auth-service.test.example.com/auth/realms/test/account/applications

Resulting HTML:
```
<td>
   <a href="javascript:alert()">
      test
   </a>
</td>
```

PoC #2 is working too if an admin is already authenticated

PoC #2 (via error page on the test area):
https://auth-service.test.example.com/auth/realms/test/protocol/openid-connect/auth?client_id=test&redirect_uri=https%3A%2F%2Fauth-service.test.example.com%2Fauth%2Fadmin%2Fmaster%2Fconsole%2F%23%2Frealms&state=c6c3a871-4b90-4e39-bb88-0b953f2bf928&respons

Resulting HTML:
```
<div id="kc-error-message">
   <p class="instruction">Invalid parameter: redirect_uri</p>
   <p><a id="backToApplication" href="javascript:alert()">« Back to
Application</a></p>
</div>
```

The following Proof-of-Concept was created to demonstrate a privilege escalation via a malicious JavaScript URL. This succeeds in getting the highest-level permissions within a Keycloak instance. Therefore, the admin from the *master realm* needs to click on the *Back to Application* link from the following error page. This will alter his or her email address to an attacker-controlled one (in this case *foo@attacker.com*). If the password reset functionality is activated on the *master realm*, the attacker is able to receive a password reset link. From there, it would be possible to set a new password and obtain access to the Admin Console from the *master realm*.

Please note that this attack also works if the admin account is protected via an OTP token. The password reset page allows to set up a new device so as to enable alternative reset of the forgotten password. Additionally, the functionality to change the email (which is enabled by default) has to be available as well.

PoC #3 (via error page on the test realm)
https://auth-service.test.example.com/auth/realms/test/protocol/openid-connect/auth?client_id=privesc&redirect_uri=https%3A%2F%2Fauth-service.test.example.com%2Fauth%2Fadmin%2Fmaster%2Fconsole%2F%23%2Frealms&state=c6c3a871-4b90-4e39-bb88-0b953f2bf928&resp

It is recommended to ensure that the given *BaseURL* always begins with the protocols of *https://* or *http://*. What is more, it is recommended to introduce a whitelist that contains characters allowed within these URLs, protecting the project from further attacks. A check should fail if the URL contains characters that are not on the whitelist.

KC-01-004 Web: Front-end limited access control on *Account Log* (*Info*)

It was discovered that the *Log* page of the *Keycloak Account Management* area, which contains information about the user's actions, remains accessible and offers previously saved data. This happens even if this function is disabled and does not appear in the application's menu. It was noted that this restriction is only present in the user front-end web interface and the user is still able to access the required page by reaching a specific uniform resource locator (URL) on the web server.

The following endpoint demonstrates the current behavior and proves access to the *Account Log* page happening from the outside of the links available in the user-menu:

https://auth-service.test.example.com/auth/realms/master/account/**log**

At least during this assessment, Cure53 could not find any ways to obtain information from the affected endpoint without a user *authorization* token being in play. As a result, this issue was ranked with the "*Info*" priority.

Despite the ranking, it is vital to underline that proper access-control should never rely only on the assumption that a user does not know how to reach various endpoints. Client-side validation should be seen as serving the purpose of assisting the user in following the rules. By no means can it constitute a final validation step within the broader process. Consequently, it is necessary to implement a server-side validation, which should enforce the rules and reply with an error message when the rules are violated.

KC-01-006 Web: Open Redirect/Phishing via *X-Forwarded-Host* header (*Info*)

It was found that the SSO endpoints on *auth-service.test.example.com* can be influenced by providing an *X-Forwarded-Host* header that points to an attacker-controlled website. This allows malicious adversaries to perform Phishing attacks against platform users, specifically by redirecting them to attacker-controlled websites from trusted domains.

Notably, while the *X-Forwarded-Host* header is considered trustworthy, browser plugins - such as Adobe Reader - are capable of forging and sending this header. A rogue PDF can initiate opening of a URL and force the browser into sending an affected header controlled by an attacker. Having power over the headers, an adversary can abuse the injection to transparently redirect their possible victim to a different page and execute Phishing or similar attacks from there. On the one hand, the overall severity of this issue can be considered "*Info*" due to a greatly limited exploitability. On the other hand, the attack remains possible and is therefore worth-fixing.

PoC Request:
```
GET /auth/realms/test/login-actions/authenticate?execution=71469e62-9d9b-4cd1-
b368-53281aeecb85&client_id=account&tab_id=8C_peLA7UKE HTTP/1.1
Host: auth-service.test.example.com
X-Forwarded-Host: evil.com
Cookie: AUTH_SESSION_ID=e934d48d-980c-47db-b655-e4a5edcc060b.keycloak-server-
deployment-69cdcc9974-wsrb7;
```

Server Response:
```
<form id="kc-form-login" onsubmit="login.disabled = true; return true;"
action="https://evil.com/auth/realms/test/login-actions/authenticate?
session_code=uSdzXvssZxiI0fYf47FALwoqeu_-Tcp7sO8bS4uOU64&amp;execution=71469e62-
9d9b-4cd1-b368-53281aeecb85&amp;client_id=account&amp;tab_id=8C_peLA7UKE"
method="post">
```

It is recommended to validate the *X-Forwarded-Host* header more strictly or completely devoid it of user-input. If this is not possible, a whitelist of trusted values should be defined, ensuring that anything absent from the whitelist is consistently rejected or ignored by the system.

Note requested by Keycloak team: *Cure53 was given access to a test system that did not validate the hostname.*

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

KC-01-008 OAuth: *Refresh* token not rotated after use (Low)

During an acquisition of a new *access* token, the *refresh* token is not rotated. According to the OAuth 2.0 specification:

> *"When client authentication is not possible, the authorization server SHOULD deploy other means to detect refresh token abuse. [...] the authorization server could employ refresh token rotation in which a new refresh token is issued with every access token refresh response. The previous refresh token is invalidated but retained by the authorization server."*[1]

Since Keycloak allows a client to not require the *client_secret* when using a *refresh* token (i.e. a public client), the lack of rotation could allow abuse of a leaked *refresh* token. It is recommended to rotate a used *refres*h token. In addition, it is also advised to employ a means to detect whether the *refresh* has been abused. Put simply, this means a capacity to monitor token rotation, with the main idea being that:

> *"If a refresh token is compromised and subsequently used by both the attacker and the legitimate client, one of them will present an invalidated refresh token, which will inform the authorization server of the breach."*

Note requested by Keycloak team: *Refresh token rotation is in fact available yet not enabled by default. The test-system Cure53 operated on did not enable that setting.*

KC-01-009 Crypto: Password hashing vulnerable to GPU-based attacks (Low)

It was found that various components of Keycloak used the *PBKDF2* password hashing function in order to produce hashes that were resistant against brute-force or dictionary attacks. In one instance, the number of *PBKDF2* iterations was set to 20,000. In another, 27,500 iterations were used, while in yet another instance, the number was set to 30,000. While these numbers of iterations are deemed sufficient according to recommendations issued by NIST, it is still the case that *PBKDF2* is vulnerable to GPU and ASIC-based attacks[2]. These can drastically speed up dictionary and brute-force attacks on *PBKDF2* hashes and render the scheme insufficiently future-proof.

Affected Files:
server-spi-private/src/main/java/org/keycloak/credential/hash/Pbkdf2PasswordHashProviderFactory.java
server-spi/src/main/java/org/keycloak/models/PasswordPolicy.java

[1] https://tools.ietf.org/html/rfc6749#section-10.4
[2] https://link.springer.com/chapter/10.1007/978-3-319-24192-0_2

Affected Code:
```
public class Pbkdf2PasswordHashProviderFactory implements
PasswordHashProviderFactory {
       public static final String ID = "pbkdf2";
       public static final String PBKDF2_ALGORITHM = "PBKDF2WithHmacSHA1";
       public static final int DEFAULT_ITERATIONS = 20000;

[...]

public class PasswordPolicy implements Serializable {
public static final String HASH_ALGORITHM_ID = "hashAlgorithm";
public static final String HASH_ALGORITHM_DEFAULT = "pbkdf2-sha256";
public static final String HASH_ITERATIONS_ID = "hashIterations";
public static final int HASH_ITERATIONS_DEFAULT = 27500;
```

It is recommended to instead use a modern password hashing function such as *scrypt*[3], which is significantly more resistant to GPU and ASIC-based attacks due to drawing its strength from memory read/write speeds.

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

KC-01-002 Web: Login IP address spoofable via *X-Forwarded-For* header (*Info*)

It was discovered that the Keycloak application offers the possibility to monitor and show the actual IP of the user through the *Account Management* page and different areas in the Admin Console. However, it was noticed that the application uses the *X-Forwarded-For* header as a source of this IP address. As this header may be controlled by the user, it is possible to submit arbitrary values and, thereby, easily spoof the real IP address.

The following example demonstrates the current behavior.

Log In request:
```
POST /auth/realms/master/login-actions/authenticate?session_code=[...]
_id=account&tab_id=GDrDoIgxJfa HTTP/1.1
Host: auth-service.test.example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:70.0)
Gecko/20100101 Firefox/70.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
```

[3] https://www.tarsnap.com/scrypt.html

```
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Cookie: [...]
X-Forwarded-For: 127.0.0.6

[...]
```

Output HTML on Sessions page:

```
<td>127.0.0.6</td>
```

It should be noted that *X-Forwarded-Host* header is by default not considered secure and the developers recommend to set a fixed hostname for this value. User-controlled *X-Forwarded-For* headers should never be used as the sole source of acquiring the IP addresses of users.

***Note requested by Keycloak team:** Cure53 was given access to a test system where the setting* PROXY_ADDRESS_FORWARDING *was set to* true*. By default, this setting is set to* false*.*

KC-01-003 Web: General HTTP security headers missing (*Info*)

It was found that pages on the admin console of the application are completely missing general HTTP security headers in HTTP-responses. This does not directly lead to a security issue, yet it might aid attackers in their efforts to exploit other problems. The flaws unnecessarily make the servers more prone to Clickjacking, channel downgrade attacks and other similar client-based attack vectors.

It is recommended to review the following list of headers to prevent various headers-related flaws:

- **X-Frame-Options**: This header specifies whether the web page is allowed to be framed. Although this header is known to prevent Clickjacking attacks, there are many other attacks which can be achieved when a web page is frameable[4]. It is recommended to set the value to either **SAMEORIGIN** or **DENY**.
- **X-Content-Type-Options**: This header determines whether the browser should perform MIME Sniffing on the resource. The most common attack abusing the lack of this header is tricking the browser to render a resource as an HTML document, effectively leading to Cross-Site-Scripting (XSS).
- **X-XSS-Protection**: This header specifies if the browser's built-in XSS auditors should be activated (enabled by default). Not only does setting this header prevent Reflected XSS, but also helps to avoid the attacks abusing the issues on

[4] https://cure53.de/xfo-clickjacking.pdf

the XSS auditor itself with false-positives, e.g. Universal XSS[5] and similar. It is recommended to set the value to either **0** or **1; mode=block**.

- **Strict-Transport-Security**: Without the HSTS header, a MitM could attempt to perform channel downgrade attacks using readily available tools such as *sslstrip*[6]. In this scenario the attacker would proxy clear-text traffic to the victim-user and establish an SSL connection with the targeted website, stripping all cookie security flags if needed. It is recommended to set up the header as follows:

  Strict-Transport-Security: max-age=31536000; includeSubDomains;

It is vital to reiterate the importance of having all HTTP security headers set at a specific, shared and central place for all application's areas. Overall, missing security headers is a bad practice that should be avoided. It is recommended to add the mentioned headers to every server response*,* including error responses like *4xx* items*.*

KC-01-005 Web: Missing input validation in IDP *authorization* URLs (*Info*)

During the assessment of the Admin Console application, it was found that almost every *authorization* URL that points to an IDP server lacks proper input validation. There is no need to allow a wide range of characters because a malicious user might be able to use them to craft deep links and, as a consequence, introduce further attack scenarios on affected clients.

For example, it is possible to set an *authorization* URL like *skype:+49030123456?call* within the settings page for *Identity Providers*. If a client starts the authorization flow via this registered IDP, s/he will be forwarded to the URL via *Location* header and the *Skype* client is opened after confirming the dialog box (see below). This can introduce further attacks in case a vulnerable application callable via deep link is registered on the system.

Affected Response (Redirect to the *authorization URL*):

```
HTTP/1.1 303 See Other
Location: skype:+49030123456?call?scope=openid&state=uMqbueX5-
12KlF0H6RVJAiqFpAWNLnCes_GivBv_zZI.XksfjYy5DvQ.security-admin-
console&response_type=code&client_id=1233&redirect_uri=https%3A%2F%2Fauth-
service.test.example.com%2Fauth%2Frealms%2Ftest%2Fbroker%2Fkeycloak-oidc
%2Fendpoint&nonce=407a48bf-5630-4fd1-a253-5084591ac894
```

[5] http://www.slideshare.net/masatokinugawa/xxn-en
[6] https://moxie.org/software/sslstrip/

It is advised to make sure that *authorization* URLs always begin with the *https://* protocol. Additionally, it is recommended to introduce a whitelist that contains characters allowed within these URLs to safeguard the project against further attacks. A check should fail if the URL contains characters that are not on the whitelist.

Note: *The severity of this issue was downgraded from Medium to Info. Deep links need user-interaction on browsers and this will stop most of the attacks. Only on mobile devices, no user-interaction is required to get the links processed.*

KC-01-007 Web: Potential XSS via unrecognized field in Keycloak *(Info)*

A potential *POST*-based Reflected XSS was found in the *OpenID Connect Dynamic Client Registration* endpoint in Keycloak. The endpoint reflects unfiltered user-input in its error message, with HTML content-type. However, this issue is not directly exploitable as it requires the request's content-type to be *JSON.* Browsers are not able to carry this out cross-origin because an SOP violation would take place. At the same time, browser/ plugin bugs that allow one to bypass this restriction could be possible.

Sample Request/Response:

```
POST https://auth-service.test.example.com/auth/realms/master/clients-
registrations/openid-connect HTTP/1.1
Content-Type: application/json
Host: auth-service.test.example.com
Content-Length: 27

{"<svg onload=alert(1)>":1}

HTTP/1.1 400 Bad Request
Content-Type: text/html;charset=UTF-8
Content-Length: 134
Date: Thu, 21 Nov 2019 09:31:29 GMT
Via: 1.1 google
Alt-Svc: clear

Unrecognized field "<svg onload=alert(1)>" (class
org.keycloak.representations.oidc.OIDCClientRepresentation), not marked as
ignorable
```

It is recommended not to have error messages rendering as HTML (i.e. having the content-type of *text/html)*, but to rather use plain-text (*text/plain*) instead. Additionally, HTML characters should be HTML-encoded in the response to prevent MIME-sniffing issues.

KC-01-010 Crypto: Kerberos employs outdated cipher suites *(Info)*

It was found that the Kerberos implementation that is loaded by Keycloak employs somewhat outdated cipher suites. These cipher suites are used in order to perform authenticated key exchanges and establish secure sessions under the Kerberos protocol. When a cipher suite is weak enough on any of the secure channel protocols, it can expose the key exchange process to being compromised by an active attacker. Some details on this matter are provided next.

1. The *AES128-CTS-HMAC-SHA1-96* cipher suite employs 96-bit HMACs, which provide significantly less entropy than the 128-bit HMACs. It is relatively much closer to the 80-bit "*safety zone*" for HMACs.
2. The *DES-CBC-MD5* cipher suite utilizes the DES block cipher, which is trivially easy to brute-force.

Affected File:
util/embedded-ldap/src/main/java/org/keycloak/util/ldap/KerberosEmbeddedServer.java

Affected Code:
```
private static final String DEFAULT_KDC_ENCRYPTION_TYPES = "aes128-cts-hmac-
sha1-96, des-cbc-md5, des3-cbc-sha1-kd";
```

While many implementations of Kerberos do not currently appear to support more secure cipher suites, some modern implementations appear to support more secure cipher suites, such as *aes256-cts-hmac-sha384-192*.[7] Migrating to a modern implementation would allow upgrading the employed cipher suite and could effectively increase the available security parameters for establishing a secure session.

***Note requested by Keycloak team:** The feature is actually meant to be used for testing purposes only and will, to make that more clear, likely be moved into a different folder.*

[7] http://web.mit.edu/kerberos/krb5-latest/doc/admin/conf_files/kdc_conf.html#encryption-types

Conclusions

This Cure53 security-centered assessment of the Keycloak application broadly concludes on a positive note. After spending twenty-one days on the scope in November 2019, seven members of the Cure53 testing team can attest to the fact that various components and aspects of the Keycloak 8.0 applications withstood their scrutiny. Having been commissioned to perform this assessment by REWE Digital GmbH, Cure53 can express a conviction that the Keycloak project is on the right track from a security standpoint.

Some of the project's objectives and tasks should be commented on in more detail before proceeding to more specific points. The basic idea behind this Cure53 investigation was to find out whether the existing functionality of the integrated application components can be deemed strong enough to withstand attacks by malicious users with different types and levels of access. The top priorities enveloped verifications of both classical problems and vulnerabilities, and possible logic and crypto and weaknesses that could lead to unauthorized access to data, eventually bringing the application or its components down.

During this assessment, a lot of tests were performed and aimed at the Keycloak application's components. Cure53 initially looked for weakness at the source code-level, but also reinforced their conclusions through additional tests on the provided test environment. The vast majority of these checks failed to identify security problems of any sort. To give a more concrete picture, no issues were identified in the following application aspects: the implementation of the OIDC, implemented access control matrix, the 2FA implementation, Java specific issues, possible logic weaknesses and various file handling actions.

The access control for creating, updating and deleting items across the Keycloak application was found to be secure, Cure53 testers did not reveal any problems linked to ACL despite many attempts. This was confirmed with vertical and horizontal access control investigations as well as across different realms. Further, Cure53 meticulously analyzed scope items where a malicious user could potentially escalate privileges or obtain access to sensitive data. The testers observed that the application clearly determined what could be seen by the user and verified whether certain features were enabled for the user prior to returning output.

After completing crypto and information flow parts of the audit, it can be said that the cryptographic primitives and APIs for the supported authentication frameworks were evaluated as implemented securely. They follow the authentication frameworks' specifications and a high level of confidence was obtained in the soundness of the cryptographic primitives and protocols exposed by Keycloak. However, a minor issue was identified in the choice of password hashing function, which is not resistant to modern GPU and ASIC-based attacks (see KC-01-009).

A general check of the 2FA flow has shown that the 2FA implementation employed on Keycloak can be judged as resistant to brute-force attacks and other flaws in this area. No common implementation of the OIDC issues was found during the project. This stems from the fact that *Keycloak* is highly configurable, allowing users to determine if it is necessary to introduce a minor compromise to security in exchange for compatibility or real-life usage. While the underlying OAuth implementation is nearly fully compliant to the specification, a minor problem was identified in the realm of rotating a used *refresh* key (KC-01-008). Overall, Keycloak application is developing nicely in terms of IdP security.

The Keycloak API and its source code were subject to in-depth examination. The overall impression made by the code is quite good as no major flaws could be found. As mentioned above, the regular expressions employed on Keycloak were examined for the possibility of ReDoS attack vectors. Yet, no vulnerable RegEx was identified during this engagement. Java-specific and more severe vulnerabilities, such as deserialization issues, XXE in SAML handling or plain command injections, have not been found, confirming successful application of defensive coding practices.

All file handling actions, such as the upload of keys, configuration files or similar, are done in a secure manner, leaving no room for exploitation. Actions which rely on files present on the filesystem, such as a password blacklist, can only be configured via direct access to the server. Further, the Keycloak application validates the input from most of the parts correctly, e.g. parsing configuration files for setting up a *SAML Identity Provider* or reflected error messages get sanitized accordingly. However, the missing input validation for *Application Links* enables malicious *admin* users to introduce XSS attacks via prepending a *javascript:* scheme (see KC-01-001). Ultimately, it turned out that acquiring access to the *admin* user from the *master realm* was a realm possibility, provided that a prepared link is clicked on.

The missing input validation for *Authorization* URLs points to shortcomings in an external *Identity Provider* (KC-01-005). This could be utilized to introduce further attack scenarios, especially on mobile devices whereat no user-interaction is required for processing deep links within the system. Finally, a potential XSS due to unencoded HTML characters and the incorrect choice of MIME-type was found in the API endpoint response (KC-01-007).

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Summing up, the overall security of the application components is particularly solid: it is evident that they are not only sturdy but also well-maintained by the in-house team at Keycloak. The absence of obvious problems, and the low number of findings in total, serve as powerful indicators of the strengths exhibited by the examined application. This conclusion holds in spite of intensive and dedicated searches for various pathways to compromise that the Cure53 team members sought out. Even though Cure53 ultimately documented ten items, the testers really needed quite some time to spot flaws. It should be underscored that even with white-box methodologies and extensive coverage, the Keycloak reached a rare result of findings being few and far between.

Nevertheless, looking at the items stemming from this assessment, it can be observed that they still represent classic and well-known problems. This not only makes them easily exploitable but also positions them as fully preventable. At the same time, a considerable majority of the disclosed issues either do not pose major threats or have manageable severity levels. Thus, Cure53 recommends to address the unveiled problems as stepping stones to accomplishing better security milestones of the application. Fixing the reported issues, which appear to have slipped past the generally careful development, will help achieve that goal. It should be noted that even issues that seem minor at first sight tend to accumulate, potentially leading to new attack chains being formed.

In Cure53's expert opinion, the results documented in this November 2019 report affirm that Keycloak is a mature and solid project. Even though it cannot be ruled out that some components of the application are suffering from issues similar to the ones described in the tickets - mostly because of the size and complexity of this project, Cure53 is confident about Keycloak moving forward securely. The outcomes show that the developer team at the Keycloak entities designs and proposes features with high awareness about the field of security. Summing up, with absence of serious problems and a generally limited presence of risks, the application makes a stable impression in relation to the core security aspects.

Cure53 would like to thank Stian Thorgersen and Bruno Oliveira from the Keycloak team as well as Matthias Jambor and Sebastian Kriesten of REWE Digital GmbH for their excellent project coordination, support and assistance, both before and during this assignment. Special gratitude also needs to be extended to REWE Digital GmbH for funding this project.