ASSURED

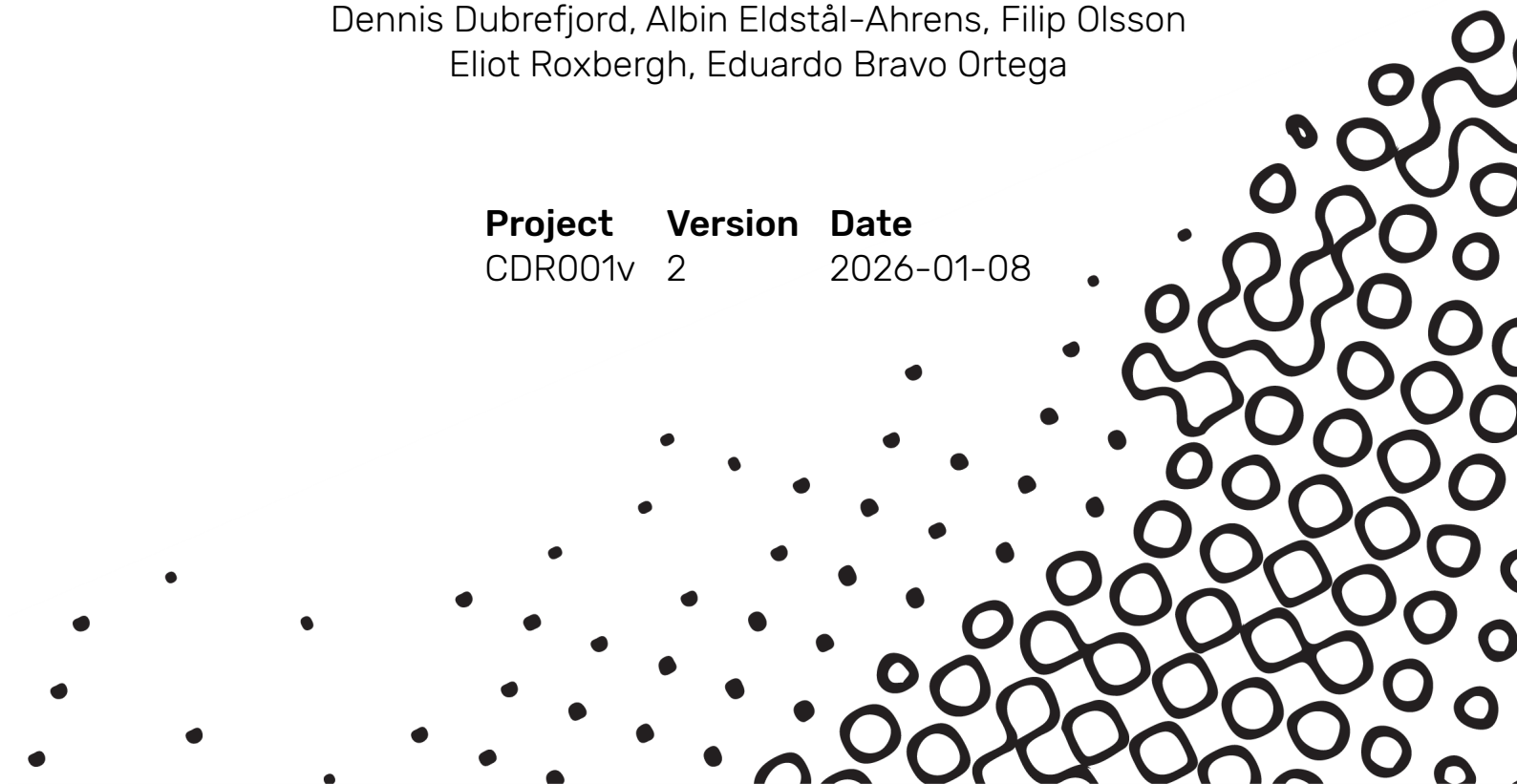
SECURITY CONSULTANTS

Report

Center for Digital Resilience - Link Penetration Test

Dennis Dubrefjord, Albin Eldstål-Ahrens, Filip Olsson
Eliot Roxbergh, Eduardo Bravo Ortega

Project	Version	Date
CDR001v	2	2026-01-08



ASSURED
SECURITY CONSULTANTS

REPORT

Project	Version	Date
CDR001v	2	2026-01-08

Executive summary

Between 2024-10-07 and 2024-10-22 Assured Security Consultants performed an application penetration test on behalf of the Center for Digital Resilience (CDR). The test covered CDR Link, a human rights helpdesk application used to provide various types of support for activists and other communities exposed to rights violations.

The test included the Link application itself, its integrations with common chat networks WhatsApp and Signal, as well as the deployment and hosting infrastructure underlying a typical Link instance.

This report lists the security issues found, along with recommendations for fixing or mitigating them. In our conclusions we discuss the issues and address apparent patterns in areas where security is lacking.

A verification test was performed from 2025-12-12 to 2025-12-17 to validate fixes and mitigations in response to the original test.

Observations in this report are tagged to indicate their status at the time of verification:

- FIXED for verified fixed findings.
- PARTIALLY FIXED for findings where we found partial mitigation in effect.
- REMAINING where the finding was verified still valid.

Observations were made with the following risk severity assessments (number of issues):

Critical 1 | High 3 | Medium 10 | Low 6 | Note 10

One attack chain of critical severity was identified, which allowed an attacker to gain full access to Link via a malicious ticket submission. This vulnerability was reported and fixed in a short timeframe, and is no longer present in the production instances of Link at the time this report is published.

A number of lower-severity vulnerabilities were also identified, with impacts ranging from spoofed information to session handling weaknesses.

Assured would like to thank Sarah Sloan and Darren Clarke, Ana Custura, and Iain Learmonth from the Center for Digital Resilience for their support during this test. We are happy to answer any questions and provide further advice.

Assured would also like to thank the Open Technology Fund (OTF) for funding the test.

ASSURED
SECURITY CONSULTANTS

REPORT

Project	Version	Date
CDR001v	2	2026-01-08

Contents

1 Introduction

1.1 Background

Assured AB (Assured) was contracted to perform a penetration test of Link, a helpdesk application developed and managed by the Center for Digital Resiliency (CDR). Link runs atop well-known open-source helpdesk software Zammad[1]. Link provides a simplified user experience for support Agents to receive support requests via channels such as E-mail, Whatsapp or Signal. Agents can then manage support requests and respond to the original creator as needed.

1.2 Constraints and disclaimer

This report contains a summary of the observations made during the project period. This report should not be considered as a complete list of all vulnerabilities, security flaws and/or misconfigurations.

1.3 Project period and staffing

Assured started the penetration test on 2024-10-07 and finished on 2024-10-22.

Verification of CDR's fixes and mitigations took place between 2025-12-12 and 2025-12-18.

This report was last reviewed on 2026-01-08.

Involved in the penetration testing were Assured security consultants Dennis Dubrefjord, Albin Eldstål-Ahrens, Filip Olsson and Eliot Roxbergh.

The verification test was performed by Assured security consultant Eduardo Bravo Ortega.

[1] `https://zammad.com`

1.4 Risk rating

1.4.1 OWASP Risk Rating Methodology

In this report we have assessed the severity of issues and identified vulnerabilities according to the OWASP Risk Rating Methodology [1].

Table 1: OWASP Risk Rating overall severity model

Overall risk severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

As Table 1 visualizes, the overall risk assessment is determined from a combined likelihood and impact of an identified vulnerability or security issue. A value from 0 to 9 is assessed for each variable, where 0-2 is determined LOW, 3-5 is MEDIUM and 6-9 is HIGH.

Likelihood is dependent on attributes related to threat actors and the identified vulnerability, with factors such as: the skill level and motivations of the threat agents; how easily the vulnerability can be found and exploited, and; how likely an exploit may be detected.

Impact depends on technical and business factors, such as: level of loss of confidentiality, integrity, availability and accountability; potential financial damage; potential brand damage, and; potential violations of privacy.

Please note that the severity assessment is made by Assured consultants and ratings may differ from the resource owners' ratings.

1.5 Assured Security Consultants

Assured Security Consultants (Assured AB[2]) was founded in 2015 with the mission to provide premier technical cybersecurity services as an independent consultancy, not affiliated with any vendor. Our team of experienced and dedicated cybersecurity specialists perform penetration testing, red team activities, secure design, embedded development, advisory, training and similar services.

We are committed to the security community as OWASP chapter leaders, event organizers and podcasters. We also take active part in security research projects into areas such as cryptography and automotive security.

[2]Assured AB, Org.nr. 556985-8276, registered in Sweden. www.assured.se

2 Scope and methodology

2.1 Key risks and threat model

The threats facing Link and its users includes a complex mixture of high-level adversaries, such as state actors, state-sponsored actors, oppressive regimes and criminal organizations. Due to the sensitive nature of the Link userbase, testing was centered not only on common vulnerabilities such as access control, but additional focus was also placed on the protection of Personally Identifiable Information (PII).

An additional threat is that the platform is made unavailable and prevent its users from accessing it. Government threat actors have multiple means to perform attacks to successfully realize these threats. These include DNS and network manipulation, social engineering, or DDoS attacks.

2.2 Scope

The test was divided into two separate scopes, carried out in parallel. The first scope covers the Link application itself, including two custom messaging bridges (Signal and Whatsapp) implemented by CDR. The second scope was concerned with the deployment and infrastructure surrounding the application when a new application instance is created.

2.2.1 Application penetration test

The Link application consists of a web front-end and API wrapper around Zammad, a well-known helpdesk software suite. Link includes its own web portal, login system and ticket management interface, but uses Zammad in the back-end for storage, access control and other features. In addition to this front-end, CDR had also developed custom message bridges between Zammad and popular messaging platforms WhatsApp and Signal. These bridges were also included in the test scope.

Testers had full access to the Link source code, which is made available for public access at `https://gitlab.com/digiresilience/link`. Accounts with Administrative and Agent privileges were provided for the test.

2.2.2 Audit of CDR Link deployment

The test environment consisted of a single Virtual Machine hosted at GreenHost. The machine runs Debian Linux and hosts the application and its various dependencies in separate Docker containers.

Testers were given full administrative access to the test server, as well as access to deployment scripts and playbooks.

2.3 Methodology

2.3.1 Application penetration test

Testing of Link was performed by using two approaches: Source code review and dynamic tests against the running test environment. The source code was reviewed using automated static analysis and manual inspection. Dynamic tests were mainly carried out manually, aided by automation tools as listed below.

The Link interface is only intended to be accessed by Administrators and Agents, not by regular users ("Customers" in Zammad parlance). After finding a way to still create a Customer user account, tests were performed from this position as well.

Testing covered various common types of vulnerabilities, ranging from login weaknesses to access control and information leakage as well as threats against the back-end service.

2.3.2 Audit of CDR Link deployment

The system hosting the test environment at `assured.cdr.link` was inspected for configuration issues which could reduce the security posture of Link when deployed for public use. Since the application components are run in separate Docker containers, privilege escalation tests were carried out from two perspectives: i) A compromised application inside a container, and ii) An attacker having escaped a container and attempting to elevate privileges.

In addition to privilege escalation vulnerabilities, the system was inspected for potential information leaks and avenues of lateral movement. One example of this is internally exposed services, which may be accessible to an attacker who has gained a limited foothold inside an application container.

2.3.3 Tools used

- Burp Suite Professional (`https://portswigger.net/burp`)
- Semgrep (`https://semgrep.dev`
- Brakeman (`https://brakemanscanner.org/`)
- testssl (`https://testssl.sh`)
- Trivy (`https://github.com/aquasecurity/trivy`)
- LinPEAS (`https://github.com/peass-ng/PEASS-ng`)
- CDK (`https://github.com/cdk-team/CDK`)

2.4 Limitations

At the time of the test, the Signal integration was not yet functional in the test environment. The source code of the integration was inspected, but no dynamic tests were performed.

3 Observations

This chapter details the observations made during the test. The first section covers vulnerabilities related to the Link application itself, and its interaction with Zammad. The second section covers observations related to the deployment and infrastructure of a Link instance.

3.1 Application

3.1.1 CRIT FIXED SSRF in API gateway

Likelihood: HIGH (8), Impact: HIGH (8)

Note: This vulnerability was reported to CDR during the test, and immediately mitigated. Assured has verified the fix, and that the vulnerability is no longer present in the test environment. Within 24 hours of reporting, the fix had been deployed to production instances of Link.

Any request starting with /zammad redirects to the Zammad system URL, and simply appends everything after /zammad to form (and then performs a request to) a URL on the form `http://zammad-nginx:8080[USER-INPUT]`.

If a user inputs a string starting with an @ sign, the resulting URL will be interpreted differently and allow the user to control the full target host and path. For example, by providing the input `@assured.se`, the URL that Link redirects to becomes `http://zammad-nginx:8080@assured.se`, which is treated as an authentication attempt to the host `assured.se`, with the username `zammad-nginx` and the password `8080`. The original request headers are added to this request, meaning that the session cookies of the user can be retrieved by redirecting the request to an attacker-controlled URL. Moreover, we could also use the vulnerability as an SSRF, sending the request to any system, internal or external. Since the response of this request is sent back to the end user, it would be possible to retrieve any data sent from the system (such as tokens).

Figure 1 shows the request received by our system, after we manipulated the redirect as outlined above. Note in particular the leaked Cookies, which belong to the Link user who visited the malicious URL.

By tricking a Link user into visiting a malicious link URL, such as https://assured.cdr.link/zammad@attacker.net, an attacker is able to redirect the user's request to an external destination. This allows for a full session takeover, since the user's session token is included in the request.

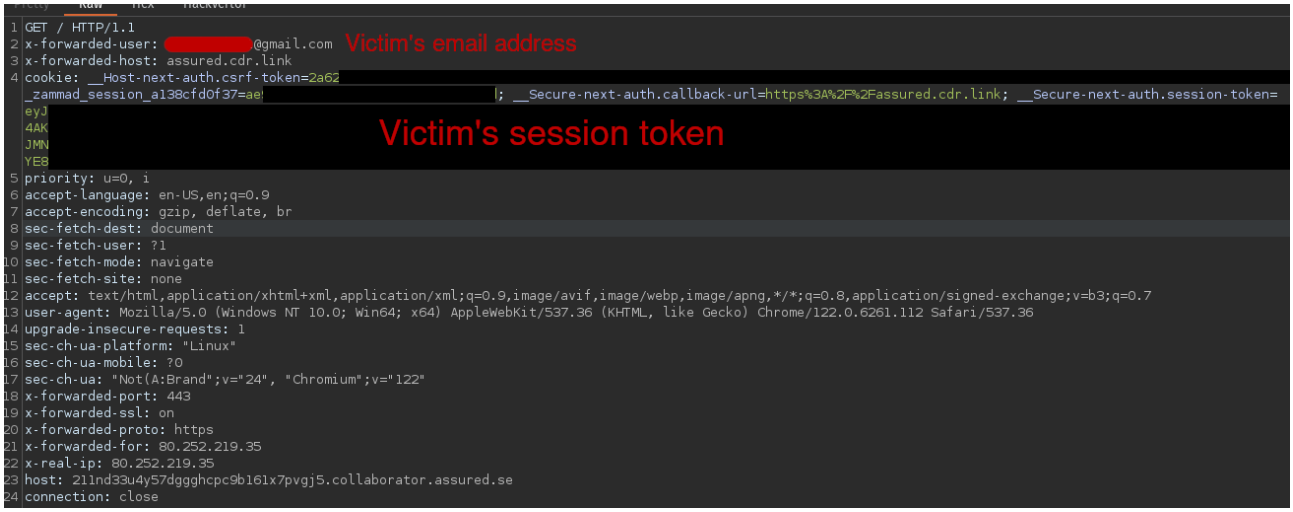


Figure 1: The attacker receives a redirected request, which includes the user's session token

When combined with Finding 3.1.2, this can be done automatically and invisibly to the user; by injecting an iframe in a ticket and setting the iframe to use the SSRF, the session token of any Link agent who views the ticket is sent to the attacker without user interaction.

When combined with Finding 3.2.5, an SSRF can give an attacker access to sensitive resources in a cloud hosting environment such as Amazon AWS. The ability to insert arbitrary headers allows an attacker to circumvent common protection mechanisms for these APIs, intended to prevent access from "simple" SSRF attacks.

We recommend adding a trailing slash to the template URL, i.e. making it `http://zammad-nginx:8080/` to prevent the user from being able to control the host portion of the URL.

Since this vulnerability is both of high impact and can be exploited via an email or WhatsApp message (See Finding 3.1.2) without authentication, we highly recommend investigating whether any similar attack has already been carried out in the production environments. A clear indicator is either an @ URL in the access logs of Zammad or any HTML in plaintext tickets.

Due to the additional risk that malicious tickets have been edited or removed after successful attack, we also recommend investigating the email inboxes associated with ticket submission to find potentially malicious payloads.

An attacker who gains Administrative privileges will also be able to execute code server-side (see Finding 3.1.15), and can potentially use this to further eliminate evidence in email or WhatsApp histories.

3.1.2 HIGH FIXED HTML injection in ticket chat

Likelihood: HIGH (7), Impact: MEDIUM (5)

Verification note: Responses where the contents of a ticket are displayed, both in Link and the Zammad interface, are properly HTML encoded.

Plaintext emails sent to Zammad are included directly in their ticket conversations without filtering. This means that an attacker can send an email with malicious contents and thereby modify the contents of the page viewed by an Agent. Figure 2 shows a plaintext (i.e. not HTML-encoded) email sent to the support system. Figure 3 shows the same message being viewed in Link by an agent. The HTML elements are rendered by the browser, including a link with a malicious target.

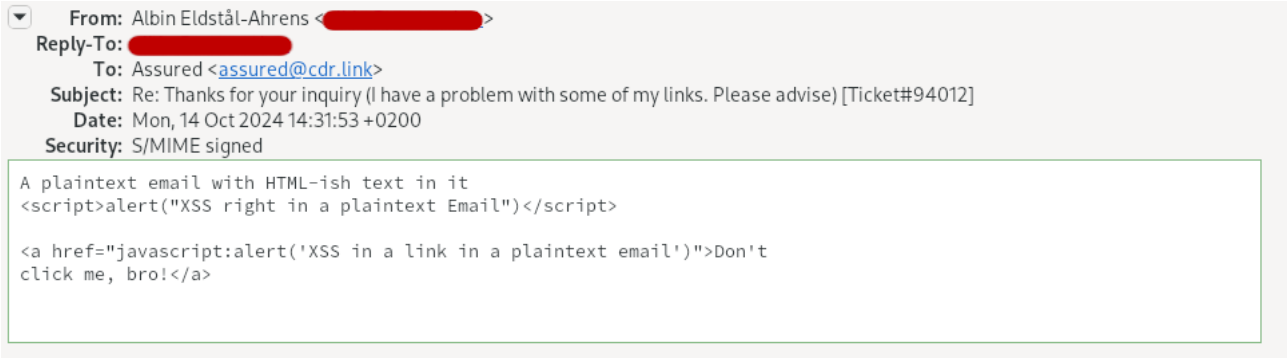


Figure 2: A plaintext email with HTML in the text

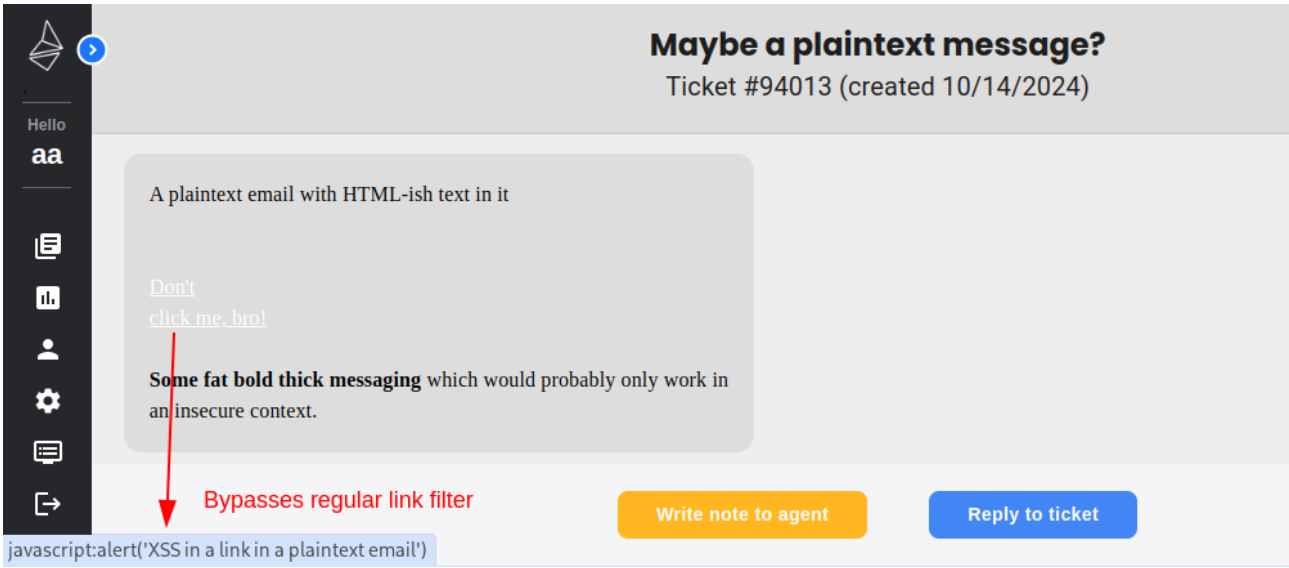


Figure 3: The malicious message rendered in Link. JavaScript is not executed.

Similar links are properly neutralized by Zammad when they originate from HTML-encoded email. When a message is sent as plaintext, this sanitization does not take place and the malicious text is insecurely included in the page DOM by Link.

The same vulnerability applies to WhatApp messages, whose text is also insecurely included in the DOM without sanitization.

The CSP (Content Security Policy) of the application successfully prevents this attack from executing JavaScript. An attacker can still insert malicious HTML and CSS stylesheets to alter the contents of the page (e.g. render a fake document on top of the actual one) and use this for phishing or other types of attacks.

We recommend sanitizing all user input before using it in a sensitive context such as HTML content. Using insecure methods such as the JavaScript built-in `innerHTML` may allow an attacker to inject malicious scripts or markup.

3.1.3 HIGH FIXED Potential timing attack in login

Likelihood: HIGH (6), Impact: MEDIUM (5)

Verification note: Basic Auth has been disabled. Only SSO through a Google account, along username and password, is possible at the time of verification.

The Link and Zammad APIs allow HTTP Basic Auth, where a header is included in each request containing an encoded email address and password. A small timing discrepancy was observed between an invalid email/password and a valid email with an invalid password.

This may allow an attacker to test whether a given email address is registered in the system for an Administrator, Agent or Customer. Repeated attempts against the same email address will quickly lock the account, but this is not likely to prevent the timing attack from being successful.

We recommend disabling Basic Auth entirely, since it is not used in the application directly. If this is not feasible, we recommend notifying the Zammad development team of the vulnerability in order for a constant-time authentication method to be implemented.

3.1.4 MED FIXED Possible to create Zammad customer account via email

Likelihood: MEDIUM (5), Impact: MEDIUM (3)

Verification note: The endpoints under /api/v1 are now locked behind authentication.

When a ticket is entered into Zammad via any channel (such as email, WhatsApp, or Signal), a Zammad customer account is automatically established for the originating person. No password is associated with the account, and Link does not expose a login form to use it. It is still possible to abuse the `/api/v1` passthrough between Link and Zammad to perform a password reset and thus activate the Zammad account.

An attacker can do this by the following process:

1. Create a ticket via email
2. Call the endpoint `/api/v1/signshow` to receive a CSRF token and Zammad session cookie
3. Call the endpoint `/api/v1/users/password_reset` to send a reset token via email
4. Call the endpoint `/api/v1/users/password_reset_verify` to set a password
5. Call the endpoint `/api/v1/signin` to sign in
6. Use the Zammad session cookie to call arbitrary Zammad endpoints via `/api/v1`

The account gets Customer access only, and can only interact with its own tickets this way. In the test environment it was possible to alter some metadata of tickets (e.g. the title) but not the ticket contents (body, attachments, and so on)

This gives an attacker a small foothold inside the application; it allows for an unauthorized user to gain a low-privilege account in Zammad and interact with its API. If vulnerabilities are identified in Zammad, they may be possible to exploit using this Customer account.

We recommend disabling the password reset functionality in Link by refusing to route to the associated endpoints. In addition, investigate options to prevent login from Customer accounts altogether. In general, the entire `/api/v1/` redirect should be locked down behind authorization checks to prevent access to sensitive Zammad endpoints by unauthenticated or Customer-level users.

3.1.5 MED PARTIALLY FIXED User IP address shows internal IP address

Likelihood: MEDIUM (4), Impact: MEDIUM (5)

Verification note: Mail notifications that disclosed the internal ip address are no longer sent. But user session address remains to be an internal ip instead of the user's IP.

Zammad helpdesk does not seem to store the actual user IP, instead an internal address is shown (specifically `172.18.0.9`). This makes it harder for users and administrators to identify suspicious logins. The IP address is leaked to user in the web UI, as well as in the "Zammad Helpdesk login from a new device" email, as shown in Figure 4. Additionally, this email is sent several times as the application is used, even if it is from the same device (Figure 5).

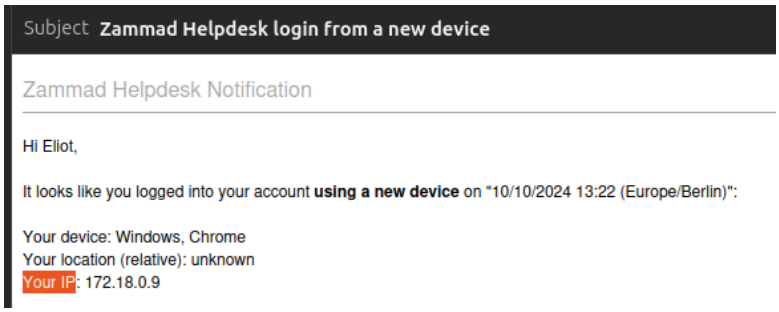


Figure 4: Users receive email showing internal IP address upon login

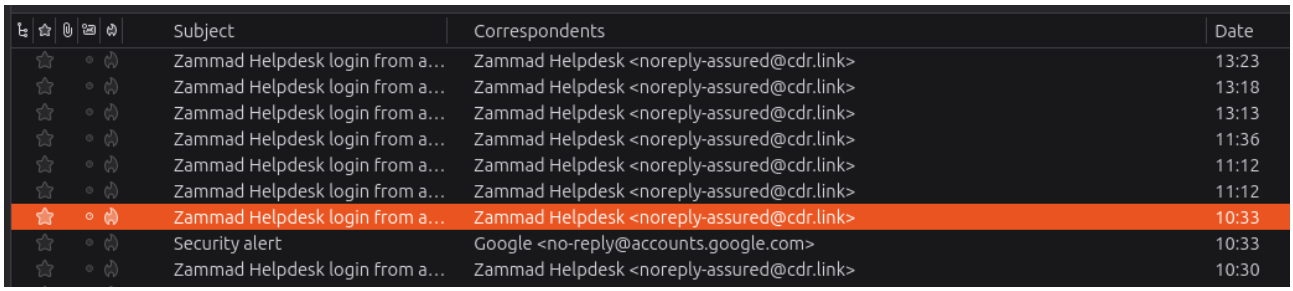


Figure 5: The user receives many emails even if the same device is used throughout

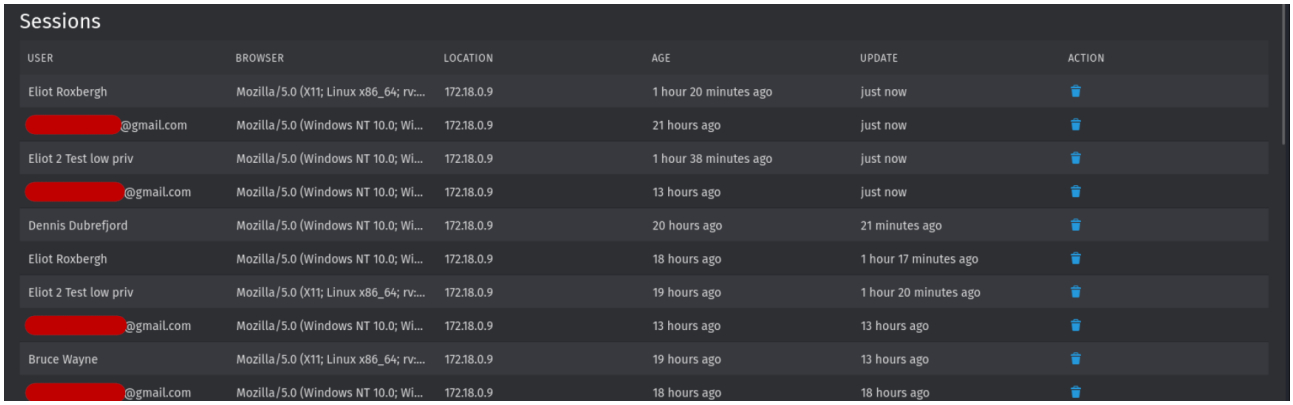


Figure 6: The administrator's view: storing only an internal IP address obscures the actual IP from which users connect.

External users thereby gain limited insight into internal IP addresses used, and are not able to identify unusual login locations. Similarly, administrators cannot see the actual user login IP addresses from the web UI (as shown in Figure 6). It is possible that server-side logging may also be partially faulty in the same manner, preventing proper auditing after a breach, although this was not the case in the system logs we observed.

The email notification itself is an account security measure intended to alert users to unusual account activity. Excessive emails and incorrect contents make this mechanism less effective, since users will rapidly learn to ignore messages and cannot trust their contents.

We recommend ensuring that the proper user IP address is logged and shown to users. Make sure the proper IP is stored in logs. Minimize the number of emails sent to users, for example by only notifying them when an IP address is observed which has not been seen in use recently.

3.1.6 MED FIXED Link authentication token long validity

Likelihood: MEDIUM (3), Impact: MEDIUM (5)

Verification note: Authentication token duration has been reduced to four days.

The session token used for access to Link has a long validity (one month) and can be refreshed indefinitely. The session token can also be used to request a new Zammad token for the same account, allowing a user with a valid token to maintain access forever.

If an attacker is able to steal a user's Link cookie (e.g. using Finding 3.1.1), it allows the attacker to maintain persistent access to the system indefinitely.

We recommend implementing a more secure flow with a short-lived session token and a separate refresh token. The refresh token should be protected from unauthorized access (i.e. if it is a cookie, it should have HTTP-only set) and if possible it should also be limited to a different scope (e.g. cookie domain) than the session token. The refresh token should only be included in requests to a bespoke token refresh endpoint, to minimize the risk of leaking.

This setup would allow the more exposed session token to have a shorter validity, while still allowing client-side code to obtain a new session token without forcing the user to repeat the login process.

3.1.7 MED FIXED Link session not invalidated by Device removal

Likelihood: MEDIUM (3), Impact: MEDIUM (5)

Verification note: Removing a device now destroys the corresponding session. Furthermore, concurrent sessions of the same user are not allowed.

The Link user profile exposes a list of logged-in device sessions and allows the user to remove sessions as needed. This is an important security feature, which gives each user an overview of their logged in sessions and an opportunity to detect and invalidate malicious logins. Sessions removed from the interface are not properly invalidated, and thus their associated session tokens remain active and functional even after explicit removal.

An attacker who is able to steal a user's session token is able to both hide their activity (by "removing" the device from the list of visible sessions) and maintain access even if a user explicitly attempts to end the session. This gives a user no recourse upon detecting a malicious or forgotten session.

We recommend deactivating user sessions server-side upon requested deletion from the session list. If this for some reason is not possible, the feature should be disabled as to not mislead users.

3.1.8 MED FIXED Link session not invalidated on logout

Likelihood: MEDIUM (3), Impact: MEDIUM (5)

Verification note: Signing out properly invalidates the session token.

User sessions are not invalidated server-side when the user logs out of Link.

An attacker who is able to steal a user's session token (e.g. using Finding 3.1.1) is able to continue using the same token even after the user explicitly logs out.

We recommend deactivating sessions server-side upon explicit user logout. This is important, since the session token itself may remain in cookie storage, caches, proxy logs, etc. after a user signs out.

3.1.9 MED FIXED Zammad cookie not removed client-side on logout

Likelihood: MEDIUM (3), Impact: MEDIUM (5)

Verification note: Signing out from Link logs the user out from Zammad, as well as removing the session cookie.

During normal use of Link, a logged-in user will have two separate access tokens: one for Link and one for the underlying Zammad system. When the user explicitly logs out of Link, the Zammad token remains stored client-side in a cookie.

This means that after logging out of Link, the machine remains "logged in" to Zammad with the same privileges. If a system is shared between Agents, their login sessions may leak between users. If a Link Administrator has signed into a system, their Zammad administrator privileges will be granted to the next Agent to log in as well.

We recommend explicitly clearing the Zammad cookie when a user logs out of Link. In addition, consider setting a short validity on the Zammad cookie, since the user's Link token can be used to fetch a new Zammad token as needed.

3.1.10 LOW REMAINING HTML injection in voice webhook

Likelihood: LOW (2), Impact: MEDIUM (3)

Verification note: Affected lines of code have not implemented the Rails built-in mechanism to prevent this issue.

The webhook implementation used for the Voice channel in Link is vulnerable to HTML injection from a number of its parameters. Figure 7 shows the affected code location, where an HTML fragment is created as a string by including values from webhook parameters.

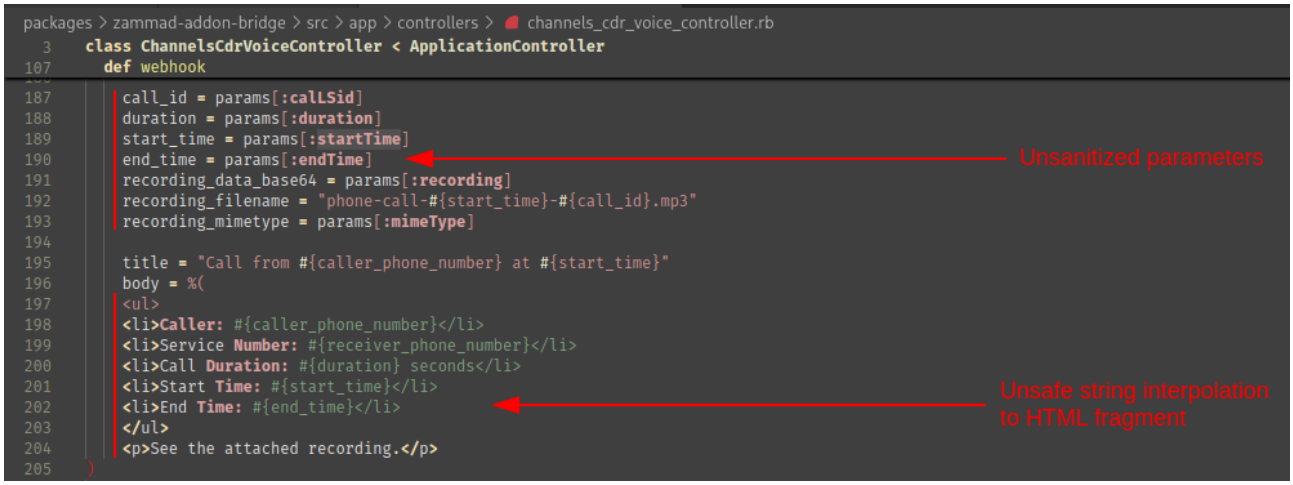


Figure 7: Code injection vulnerability in voice webhook

The webhook requires a valid token to call, and the CSP of the application successfully prevents any malicious JavaScript execution. The vulnerability could allow an attacker to spoof parts of the document, for example to forge a login form for phishing purposes.

We recommend sanitizing user input before including it in sensitive contexts such as HTML. Rails has built-in support for this through the `ActionView::Helpers::SanitizeHelper` class.

3.1.11 LOW FIXED Account lockout may cause denial of service

Likelihood: MEDIUM (5), Impact: LOW (2)

Verification note: Basic Auth is disabled and the /api/v1 endpoints are now locked behind authentication.

HTTP Basic Auth is enabled, which allows a user to attempt authentication by calling any API endpoint. After five failed attempts, the associated account is locked and must be unlocked by an administrator.

An attacker who has mapped out the email addresses of agents and administrators of a Link instance can effectively halt the system by locking the accounts. This does not affect the ability of users to submit new tickets since they do not require login.

We recommend disabling HTTP Basic Auth, since it is not used in the Link setup. In addition, consider requiring MFA for sensitive accounts to prevent this type of abuse through other channels (e.g. SSO provider). In general, the entire `/api/v1/` redirect should be locked down behind authorization checks to prevent access to sensitive Zammad endpoints by unauthenticated or Customer-level users.

3.1.12 LOW PARTIALLY FIXED Zammad API allows for password and token authentication

Likelihood: MEDIUM (3), Impact: LOW (2)

Verification note: The conditions that led to this finding remain mostly unchanged. The main difference being the removal of token authentication.

An admin or agent can create a password or a personal access token to access Zammad directly. This creates a second method of access to the application, bypassing the SSO login on Link. No multi-factor authentication is used.

We also note that access tokens by default have no expiry.

Application access without multi-factor authentication is prone to password spraying or brute-force attacks. As this login can include powerful administrative access, this creates a clear risk to the application.

Moreover, even if an acceptable password policy is in place on Zammad, an admin is not limited by this and can change users' passwords, creating a potential risk of weak credentials. However, even if the password policy could be improved, multi-factor authentication is the main concern as this greater protects against phishing attacks and password reuse issues.

We recommend implementing MFA as part of user authentication. MFA should be mandatory at least for administrators and forced on logins from new devices and for certain sensitive actions.

3.1.13 NOTE PII of users available to Agents

Two endpoints exist which allow users with Agent privileges in Link to list personal information about all users. This includes other Agents and Administrators, as well as "customers" who have submitted tickets but not otherwise used Link. Figure 8 shows an Agent who is not a member of the "users" group (no access to any tickets) listing the first and last names as well as email address of a user who has only submitted tickets via email. Figure 9 shows an Agent listing users with a different endpoint, and revealing the WhatsApp account number of a user who has submitted a ticket.

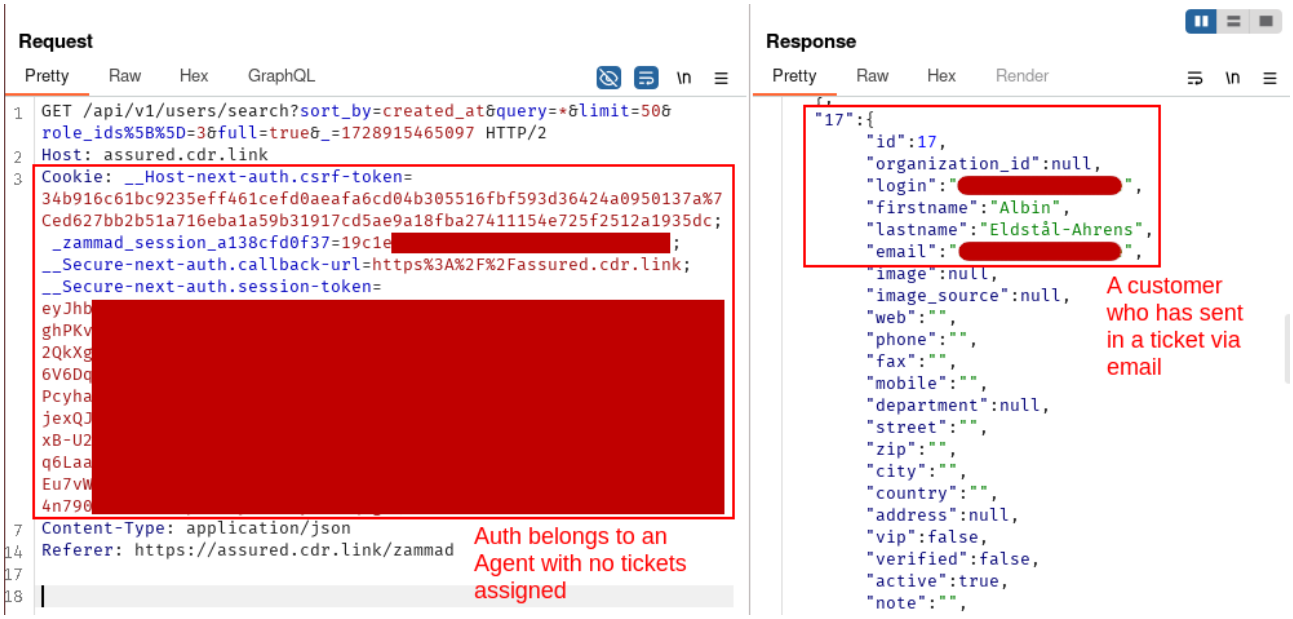


Figure 8: A support Agent is able to list PII of a support customer

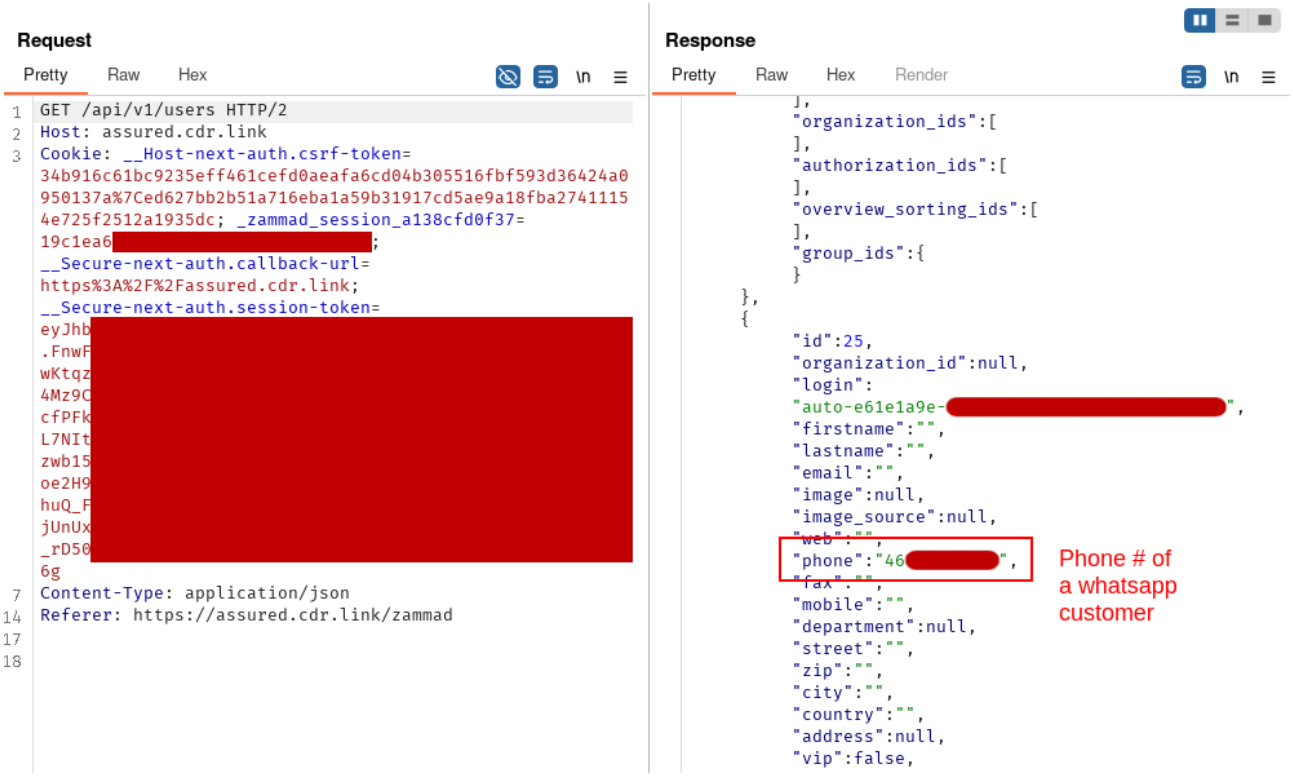


Figure 9: An Agent listing the WhatsApp contact number of a ticket submitter

If the identity of support users is intended to be secret from some Agents (for example using Zammad groups such as "Users"), this can be circumvented by performing Zammad API calls.

We recommend investigating if the sensitive endpoints need to be limited. This may be possible by implementing a filtering proxy to the Zammad API, or it may be implemented by patching Zammad itself to limit the permissions of Agent users.

3.1.14 NOTE User token permissions may be set arbitrarily

A personal access token can be set (via `https://assured.cdr.link/profile`) by Admins and Agents. This request may however be modified in any way (shown in Figure 10), whereby an agent can add restricted administrative permissions (or completely arbitrary text) to the created token (shown in Figure 11). However, this did **not** have any impact for the application, and the user was still limited according to their permissions.

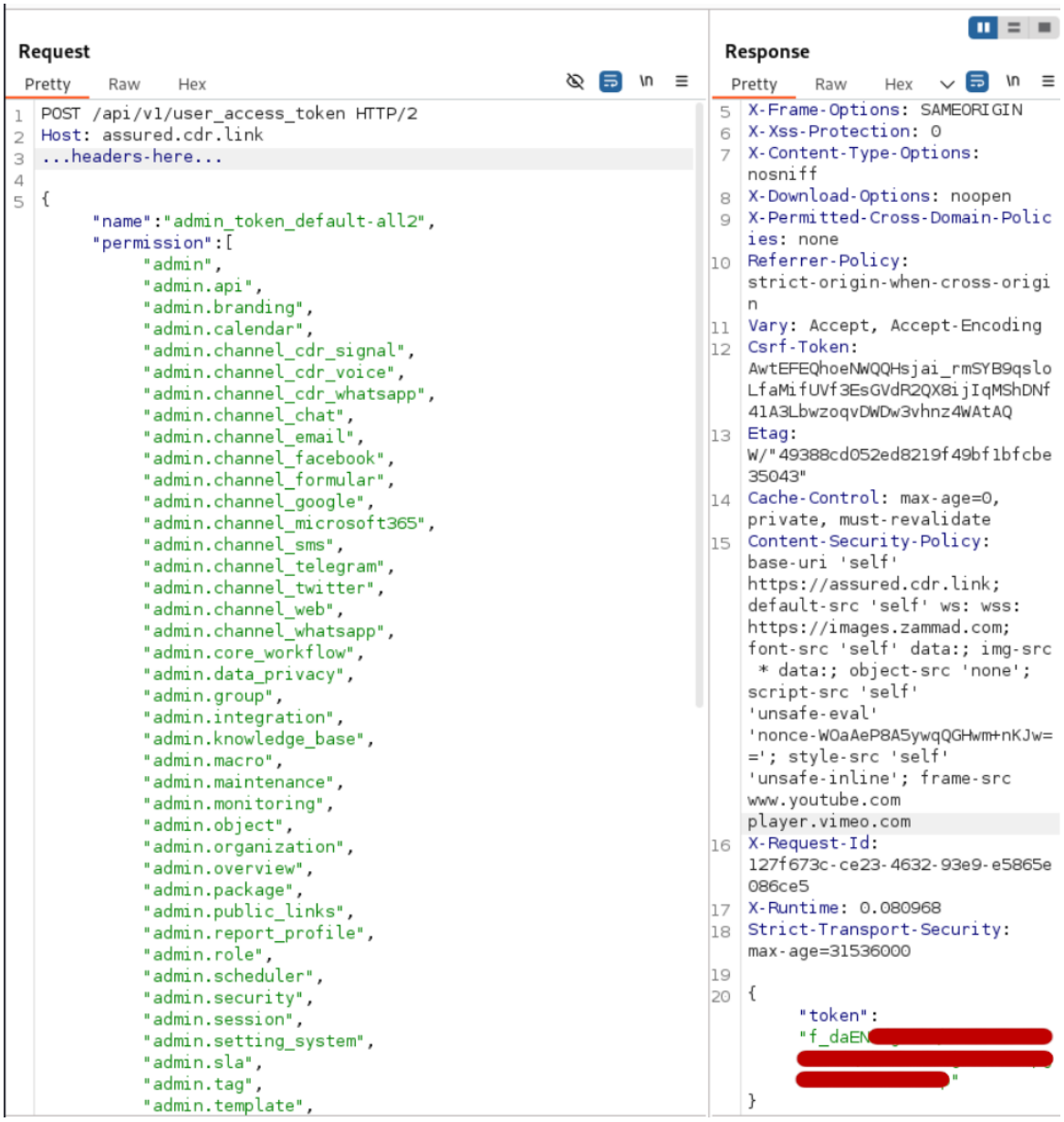


Figure 10: Creating an "Admin" access token as an Agent

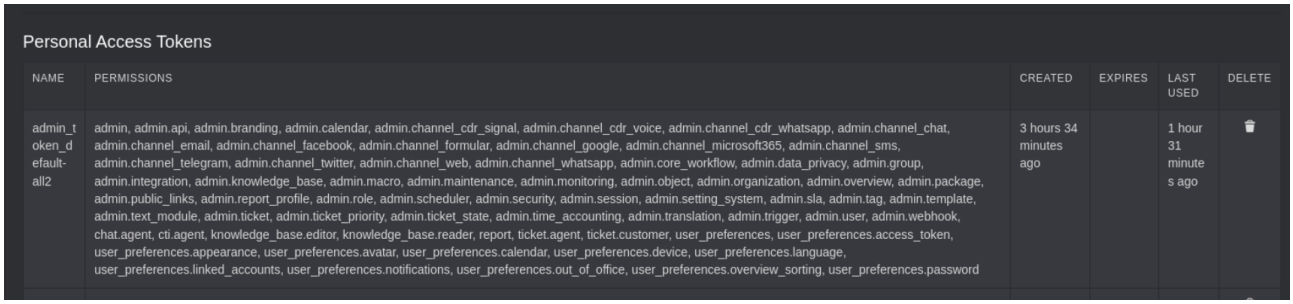


Figure 11: The new user access token shows that custom values are accepted and stored

We recommend ensuring server-side, in addition to client-side, that fields have the expected data, to prevent inconsistencies or risk of future misconfiguration that could enable a vulnerability.

3.1.15 NOTE Code execution via Zammad admin privileges

Any user who has Administrative privileges in Zammad is able to install addons via the web API. This allows the user to execute arbitrary code in the Zammad container.

By extension, in combination with Finding 3.2.6, an attacker can move laterally from almost any container to code execution in the Zammad container by first creating an administrative account and then uploading a malicious addon.

3.1.16 NOTE Zammad report shows detailed error logs

When an administrator account was logged in and viewing the CDR Link web interface at `https://assured.cdr.link/zammad#report` and pressed "Closed", a detailed error log was shown including basic information about internal service `http://opensearch:9200/`, namely:

```
Unable to process GET request to elasticsearch URL 'http://opensearch:9200/
    zammad_production_ticket/_search'. Check the response and payload for detailed information:

Response:
#<UserAgent::Result:0x0000774cefb31600 @success=false, @body="{\"error\":{\"root_cause\":[{\"
    type\":\"query_shard_exception\",\"reason\":\"No mapping found for [close_at] in order to
    sort on\",\"index\":\"zammad_production_ticket\",\"index_uuid\":\"DX7ZeqEVShqx5kER9d6epQ
    \"}],\"type\":\"search_phase_execution_exception\",\"reason\":\"all shards failed\",\"phase
    \":\"query\",\"grouped\":true,\"failed_shards\":[{\"shard\":0,\"index\":\"
    zammad_production_ticket\",\"node\":\"ICFOCFjlQvu1zhZ_ETUpGQ\",\"reason\":{\"type\":\"
    query_shard_exception\",\"reason\":\"No mapping found for [close_at] in order to sort on
    \",\"index\":\"zammad_production_ticket\",\"index_uuid\":\"DX7ZeqEVShqx5kER9d6epQ\"}}]},\"
    status\":400}", @data=nil, @code="400", @content_type=nil, @error="Client Error: #<Net::
    HTTPBadRequest 400 Bad Request readbody=true>!", @header={"content-type"=>"application/json;
     charset=UTF-8", "content-length"=>"585"}>

Payload:
{"size":100,"query":{"bool":{"must":[{"range":{"close_at":{"from":"2023-12-31T23:00:00Z","to
    ":"2024-12-31T22:59:59Z"}}},{"bool":{"must":[{"bool":{"must_not":[{"term":{"state.name.
    keyword":"merged"}}]}}]}}]}},"sort":[{"close_at":{"order":"desc"}},"_score"]}

Payload size: 0M
```

The logs we observed were only viewable by an administrator and could be expected behavior. However, if such logs were to become available to normal users it would mean unwanted information disclosure, as detailed logs and internal server information could be useful for an attacker.

3.1.17 NOTE Outdated third-party dependency

Trivy reports that one third-party dependency has a known vulnerability, with a low severity (`CVE-2024-47764`).

The package in question is `cookie` of version 0.5.0 and 0.6.0, which are included several times in `package-lock.json`. This vulnerability was only recently reported, on Oct 4th, and seems to have no impact in the application.

We recommend updating the vulnerable dependency and establishing a vulnerability management process to identify and address vulnerabilities in third-party components. Establish a process for software and library updates, to prevent applications falling out of maintenance. Software Composition Analysis (SCA) and Supply Chain Management (SCM) processes can be useful in tracking dependencies, licenses and versions used by your software.

3.1.18 NOTE Strict-Transport-Security header included twice

The HTTP Strict-Transport-Security (HSTS) header is used to enforce HTTPS for clients and avoid HTTP-downgrade attacks.

Due to a misconfiguration, several endpoints return two HSTS headers, while the client only expects one. An example of this is `/overview/recent`, where the following response header was observed by the client:

```
Strict-Transport-Security: max-age=63072000; includeSubDomains; preload
.....
Strict-Transport-Security: max-age=31536000
```

If the second entry were to be used by a client, it would disable the two security features `includeSubDomains` and `preload`. As clients are only supposed to parse the first HSTS value (according to RFC6797[3]), we deem this to be unlikely. Still, having two different HSTS headers may increase the risk of confusion for system owners and developers, who believe a certain HSTS value is used when in fact it is not.

We recommend ensuring that only one HSTS header is set.

[3] `https://www.rfc-editor.org/rfc/rfc6797#section-8.1`

3.1.19 NOTE Weaker TLS ciphers allowed

The TLS configuration is overall good, and only supports TLS v1.2 and v1.3. It does however allow for cipher suites without forward secrecy, and weaker cipher suites such as `TLS_RSA_WITH_AES_128_CBC_SHA`[4]. This does not currently have any impact, and it may be needed for compatibility reasons. That being said, this configuration could potentially be additionally hardened.

The following supported cipher suites were reported by testssl:

```
TLSv1.2 (server order)
 xcca8   ECDHE-RSA-CHACHA20-POLY1305       ECDH 253   ChaCha20    256      TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
 xc02f   ECDHE-RSA-AES128-GCM-SHA256       ECDH 253   AESGCM      128      TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
 xc030   ECDHE-RSA-AES256-GCM-SHA384       ECDH 253   AESGCM      256      TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
 x9e     DHE-RSA-AES128-GCM-SHA256         DH 4096    AESGCM      128      TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
 x9f     DHE-RSA-AES256-GCM-SHA384         DH 4096    AESGCM      256      TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
 xc027   ECDHE-RSA-AES128-SHA256           ECDH 253   AES         128      TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
 xc028   ECDHE-RSA-AES256-SHA384           ECDH 253   AES         256      TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
 xc013   ECDHE-RSA-AES128-SHA              ECDH 253   AES         128      TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
 xc014   ECDHE-RSA-AES256-SHA              ECDH 253   AES         256      TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
 x67     DHE-RSA-AES128-SHA256             DH 4096    AES         128      TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
 x33     DHE-RSA-AES128-SHA                DH 4096    AES         128      TLS_DHE_RSA_WITH_AES_128_CBC_SHA
 x6b     DHE-RSA-AES256-SHA256             DH 4096    AES         256      TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
 x39     DHE-RSA-AES256-SHA                DH 4096    AES         256      TLS_DHE_RSA_WITH_AES_256_CBC_SHA
 x9c     AES128-GCM-SHA256                 RSA        AESGCM      128      TLS_RSA_WITH_AES_128_GCM_SHA256
 x9d     AES256-GCM-SHA384                 RSA        AESGCM      256      TLS_RSA_WITH_AES_256_GCM_SHA384
 x3c     AES128-SHA256                     RSA        AES         128      TLS_RSA_WITH_AES_128_CBC_SHA256
 x3d     AES256-SHA256                     RSA        AES         256      TLS_RSA_WITH_AES_256_CBC_SHA256
 x2f     AES128-SHA                        RSA        AES         128      TLS_RSA_WITH_AES_128_CBC_SHA
 x35     AES256-SHA                        RSA        AES         256      TLS_RSA_WITH_AES_256_CBC_SHA

TLSv1.3 (no server order, thus listed by strength)
 x1302   TLS_AES_256_GCM_SHA384            ECDH 253   AESGCM      256      TLS_AES_256_GCM_SHA384
 x1303   TLS_CHACHA20_POLY1305_SHA256      ECDH 253   ChaCha20    256      TLS_CHACHA20_POLY1305_SHA256
 x1301   TLS_AES_128_GCM_SHA256            ECDH 253   AESGCM      128      TLS_AES_128_GCM_SHA256
```

[4] https://ciphersuite.info/cs/TLS_RSA_WITH_AES_128_CBC_SHA

3.2 Deployment

3.2.1 HIGH FIXED Shared administration account

Likelihood: MEDIUM (3), Impact: HIGH (6)

Verification note: SSH access and sudo access is managed via FreeIPA/IdM. Moreover, privileges are managed in a fine-grained manner by limiting which commands can be executed, and which users and groups can a user impersonate through a sudo policy.

The deployment playbook configuration file `ansible-cdr-link/inventory/group_vars/all.yml` adds SSH keys for a number of people to log in as admin on the host system. This account has full access to run arbitrary commands as root using sudo.

Sharing a single account for administrative tasks makes it more difficult to accurately perform audit logging. If one of the private keys is compromised, the shared account makes it more difficult to trace malicious operations and perform incident response.

We recommend using separate accounts for all administrative users, and preventing elevation to the single root account. By forcing administrators to use personal accounts for all sensitive operations, audit logging becomes possible.

3.2.2 MED FIXED Docker guest runs as root

Likelihood: MEDIUM (3), Impact: MEDIUM (5)

Verification note: Containers have been migrated to Podman. These containers are no longer running as root.

A number of containers, listed in Example 1, run processes with root privileges.

Example 1: Containers running as root

```
nginx_proxy
nginx_proxy_gen
nginx_proxy_le
zammad-zammad-nginx-1
zammad-zammad-railsserver-1
zammad-zammad-scheduler-1
zammad-zammad-websocket-1
```

While the processes execute in the confinement of Docker containers, the `root` user (UID 0) still has elevated privileges within the container. The account itself is the same as the root account on the host machine, but with certain capabilities stripped.

In the presence of other container configuration vulnerabilities, root access can allow an attacker to elevate privileges on the host system. If a kernel vulnerability is found, it is likely that it can be used to escape the container.

In addition, a root docker container with RW volumes mounted can be used to elevate privileges outside the container. If an attacker has a low-privilege foothold in the host and can run code inside the container, these two factors can be used to gain root privileges on the host as shown in Example 2.

Example 2: Example of host privilege escalation using a root container

```
# Running inside the container
# /usr/share/nginx/html is an RW volume
root@container:/usr/share/nginx/html/assured# cp /usr/bin/chmod root_chmod
root@container:/usr/share/nginx/html/assured# chmod 4755 root_chmod

# The root_chmod executable is now setuid root
# This is also true in the host:
admin@host:~$ sudo ls -lah /var/.../nginx_proxy_html/_data/assured/root_chmod
-rwsr-xr-x 1 root root 63K Oct 9 07:50 /var/.../nginx_proxy_html/_data/assured/root_chmod
```

By using the root privileges inside the container to make a binary setuid root, that binary can then be run with root privileges on the host as well.

We recommend running container guests with non-root privileges. Where possible, use a UID which does not overlap with user accounts on the host system. In addition, we recommend using the `nosetuid` option on shared filesystems to prevent privilege misuse.

3.2.3 MED FIXED Docker daemon runs as root

Likelihood: MEDIUM (3), Impact: MEDIUM (5)

Verification note: The Podman daemon runs as a custom user assigned to a specific group instead of root.

The docker daemon can be executed as root (default) or as a regular user. Running the docker daemon as root increases the impact if an attacker would compromise the host system through docker container breakout attacks.

An attacker that successfully performs a breakout from a docker container on the host, can execute system commands and code as the root user. Making the full system compromised.

We recommend to evaluate if running the docker daemon in rootless mode is suitable for your design. If rootless mode is not possible, consider using Podman[5] as an alternative to docker.

[5] https://podman.io/

3.2.4 MED FIXED Containers with write privileges to configuration volumes

Likelihood: MEDIUM (3), Impact: MEDIUM (4)

Verification note: Containers have been migrated to Podman. Podman container configuration files, like /usr/share/containers/containers.conf and /etc/containers/containers.conf are read-only inside of the container.

Some Docker containers have volumes mounted from the host, containing application configuration which are mounted with RW enabled, giving the application write privileges in a configuration directory on the host.

Example 3 lists volumes which are mounted with RW enabled, giving the application write privileges in a configuration directory on the host.

Example 3: Containers and volumes with Read-Write access

```
nginx_proxy: nginx_proxy_conf
nginx_proxy: nginx_proxy_vhost
nginx_proxy_gen: nginx_proxy_conf
zammad-zammad-railsserver-1: zammad-database.yml
```

An attacker who uses an application vulnerability to gain a limited foothold (e.g. the ability to append data to an arbitrary file) within a container may use these write privileges to gain extended access (e.g. Remote Code Execution) or persistence. A malicious update to the nginx configuration may allow for transparent (i.e. without losing TLS) hijacking of web traffic.

We recommend mounting configuration and other volumes with read-only permissions, unless it is strictly necessary for application functionality.

3.2.5 MED FIXED Potential cloud metadata access

Likelihood: MEDIUM (4), Impact: MEDIUM (4)

Verification note: Link and Zammad are now deployed in SR2's public cloud https://docs.cdr.link/-docs/hosted/. SR2 has seemingly no metadata API to abuse.

Docker containers are not prevented from accessing external resources, such as the metadata API of hosting providers like AWS.

If the cloud service is configured to allow it, an attacker who gains a foothold inside an application container (e.g. by exploiting a vulnerability in Zammad) can gain access to the metadata API of the cloud provider. Access to the metadata API may give an attacker access to credentials and other sensitive information.

We recommend blocking access from containers to the cloud provider metadata API, for example using firewall rules. The implementation of this must be aware of the specific cloud host used for an instance, which may increase the complexity of deployment.

3.2.6 LOW REMAINING Postgres database accessible without password

Likelihood: LOW (1), Impact: HIGH (8)

Verification note:
This is an issue that the development team is currently aware of and is taking steps to remediate, like implementing an argument in the deployment script that will override both host and local authentication for new databases. Furthermore, awareness on the fact that manual intervention will be required for already existing ones is also being taken into account. The severity has been lowered since it would require escalation to root through other means in order to access the database.

The PostgreSQL database in the container `zammad-zammad-postgresql-1` is accessible internally on the host using the account name "postgres" and a blank password. The user has access to the zammad database, including user information as shown in Example 4.

Example 4: Access to the postgres database from a container. Emails have been redacted.

```
root@d6b81d8cc1ed:/# psql -U postgres --no-password
psql (16.4 (Debian 16.4-1.pgdg120+2))
Type "help" for help.

postgres=# \c zammad_production
You are now connected to database "zammad_production" as user "postgres".
zammad_production=# select email,password from users;
          email           |          password
--------------------------+-----------------------------------
                          |
 a____@sr2.uk             | $argon2id$v=19$m=65536,t=3,p=4$Skb3kS...
 support@_______.com      |
 da______@red_____.com    | $argon2id$v=19$m=65536,t=3,p=4$JzuDcT...
 __________@gmail.com     |
 ___________@assured.se   |
 ______________@gmail.com | $argon2id$v=19$m=65536,t=3,p=4$ftiNcE...
 _______@sr2.uk           |
 _________@gmail.com      |
 _________@assured.se     | $argon2id$v=19$m=65536,t=3,p=4$6fetiM...
```

The corresponding port is exposed to the host system, but could not be tested due to the host system missing a postgres client. The database is **not** exposed to the internet.

An attacker who has a foothold in one of the application containers (e.g. Link or one of the messaging bridges) is able to connect to the zammad database and read sensitive information from it. It is also possible from this position to add new users to Zammad, including users with administrative privileges.

The database contains other possibly sensitive tables:

- `http_logs`, which is empty in the test system, suggesting that logs are not collected here
- `store_files`, which contains information about file uploads

- `ticket_articles`, which contains text from support tickets

We recommend disabling the password-less postgres account, and relying only on the (already present) database credentials which are secure and unique per instance.

3.2.7 LOW FIXED Redis database accessible without authentication

Likelihood: LOW (2), Impact: MEDIUM (3)

Verification note: Redis requires a password for authentication now. Furthermore, access to redis is only allowed from its corresponding container.

A redis instance is running in its own container, and requires no authentication for connecting clients. The server is accessible to the `nginx-proxy` docker network, which includes the majority of the running containers.

The database was empty during the test, and is likely only used for temporary storage in Zammad. There is a risk that applications like Zammad trust data from the Redis database, and therefore applies less stringent input sanitization to such data.

We recommend enabling authentication on all services, even internal ones. Redis supports password authentication, and this should be enabled to prevent the use of the redis database for potential data leakage or pivoting between containers.

3.2.8 LOW REMAINING /proc filesystem mounted in container

Likelihood: LOW (2), Impact: MEDIUM (3)

Verification note: The /proc filesystem is still mounted in the containers.

A number of containers, listed in Example 5, have the `/proc/` filesystem of the host mounted inside the guest:

Example 5: Containers with the `/proc` filesystem mounted

```
nginx_proxy_gen
zammad-opensearch-1
zammad-signal-cli-rest-api-1
zammad-zammad-memcached-1
```

If a guest process has root privileges and read access to the host's `/proc/` filesystem, this can give an attacker information that is useful for container breakout.

We recommend not mounting the /proc filesystem inside containers if it is not necessary for application functionality.

3.2.9 NOTE Docker UID shared with administrative user

The test system runs application containers with various identities. In some cases, the UID 1000 is used for the guest processes. In the host system, this UID belongs to the user admin who has unrestricted sudo rights (effectively root privileges on the host). Example 6 shows the containers using UID 1000:

Example 6: Containers running with UID 1000

```
zammad-bridge-whatsapp-1
zammad-bridge-worker-1
zammad-link-1
zammad-opensearch-1
zammad-signal-cli-rest-api-1
```

If an attacker finds a limited container breakout, the foothold as `admin` in the host system provides a direct path to privilege escalation and subsequently a full system takeover. It is unlikely that such a breakout will occur; kernel vulnerabilities which allow breakout tend to either require root privileges inside the container or immediately give root privileges in the host.

We recommend avoiding UID reuse between host and containers. If possible, assign UIDs to containers which have no privileges in the host.

3.2.10 NOTE Docker socket mounted inside containers

Two Docker containers (`nginx_proxy_le` and `nginx_proxy_gen`) have the host's Docker control socket mounted inside the container. This serves as a method for an attacker to break out of the container and access the host system, as well as gain root privileges in the host. Listing 7 shows the mount for the `nginx_proxy_gen` container.

Example 7: An excerpt from the container state, showing the mounted control socket

```
{
  "Type": "bind",
  "Source": "/var/run/docker.sock",
  "Destination": "/tmp/docker.sock",
  "Mode": "ro",
  "RW": false,
  "Propagation": "rprivate"
}
```

The two containers have no exposed ports and do not accept user input. It is therefore deemed unlikely that this has any security impact.

3.2.11 NOTE SSH root login inconsistency

The root user has an SSH public key configured for login. The SSH server configuration contains two conflicting directives as shown in Example 8:

Example 8: Excerpt from sshd configuration

```
PermitRootLogin no
AllowUsers root admin
```

The AllowUsers directive comes from the link-ops repository, in group_vars/all.yml:

Example 9: Excerpt from all.yml

```
ssh_allow_users: root admin
```

Testing confirms that root is **not** permitted to log in.

This inconsistency may lead to confusion about the system configuration.

We recommend explicitly disallowing direct root login and using personal administration accounts as outlined in Finding 3.2.1.

4 Conclusions and recommendations

Assured security consultants performed a penetration test of Link, a human rights help desk application developed by the Center for Digital Resilience (CDR).

Link is a front-end and API wrapper for well-known Open-Source helpdesk software Zammad, and deploys its own instance of Zammad in the back-end. Link also adds communication bridges for popular chat applications WhatsApp and Signal. The test included these bridges, the Link software and the deployment/hosting security of Link in a bespoke testing environment.

One critical vulnerability was identified, which allowed an unauthorized attacker to perform a partial or complete take-over of a Link instance via email or WhatsApp. The vulnerability was immediately reported and fixed by CDR, and the fix has been deployed to all instances of Link at the time this report is published.

Other vulnerabilities were found in the application code itself, with impacts ranging from minor information leaks to spoofing which could aid in phishing attacks against help desk Agents. In addition to this, a small number of integration issues were found between Link and Zammad, which expose Zammad functionality intended to be disabled or hidden by Link.

A number of observations in this report touch upon account and session security. Session handling was found to be lacking in several regards, making it difficult for a user to manage their active login sessions or prevent, detect and counteract leaks of an active session.

The deployment was found to be sound, with a small number of issues resulting in an increased risk of privilege escalation or lateral movement after an attacker is able to gain a foothold on the application server. Application components are kept separated from each other with industry standard containerization mechanisms, and the report contains a number of recommendations for hardening this configuration further.

In addition, some observations were made which affect audit logging. The recommendations related to this can aid in investigations after a possible breach, both in terms of limiting the damage and assessing the extent of compromise.

Our recommendations can be summarized as follows:

Recommendations for web application findings

- Sanitize all user input before using it in sensitive contexts such as HTML fragments.
- Consider requiring Multi-Factor Authentication for sensitive accounts.
- Ensure proper user IP address is logged and shown to users. Send *"new device"* emails only on new user-agent and/or IP address.
- Limit personal access token expiration. Do not allow tokens without expiry.

Recommendations for back-end findings

- Disable the passwordless postgres account.
- Avoid mounting the /proc filesystem inside containers, where possible.

References

[1] OWASP, "OWASP Risk Rating Methodology." https://owasp.org/www-community/OWASP_Risk_Rating_Methodology, 2023.