Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Pentest-Report Raspberry Pi Web Applications, Client App & Infra 01.2025

Cure53, Dr.-Ing. M. Heiderich, E. Foudil, MSc. J. Moritz, MSc. N. Krein

Index

- Introduction
- Scope
- Test Methodology
- Identified Vulnerabilities
  - RSP-01-004 WP1: Login CSRF attack using Pi serial number (Low)
- Miscellaneous Issues
  - RSP-01-001 WP2: HTTP Origin header acceptance is too broad (Low)
  - RSP-01-002 WP2: CSP can be bypassed due to wide script-src directive (Low)
  - RSP-01-003 WP2: HTML injection in email address suggestion feature (Info)
  - RSP-01-005 WP1: Devices can initiate WebRTC connections (Info)
  - RSP-01-006 WP1: connect user gains root privileges through docker group (Info)
- Conclusions

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Introduction

“Computing for everybody - From industries large and small, to the kitchen table tinkerer, to the classroom coder, we make computing accessible and affordable for everybody.”

From https://www.raspberrypi.com/

This report describes the results of a penetration test and source code audit against the Raspberry Pi Connect web application, the Raspberry Pi ID authentication provider, and the Connect Golang client that runs on Raspberry Pi devices.

To give some context regarding the assignment’s origination and composition, Raspberry Pi Ltd contacted Cure53 in September 2024. The test execution was scheduled for January 2025, namely in CW02 / CW03. A total of twenty days were invested to reach the coverage expected for this project, and a team of four senior testers was assigned to its preparation, execution, and finalization.

The methodology conformed to a white-box strategy, whereby assistive materials such as sources, URLs, documentation, test-user credentials, as well as all further means of access required to complete the tests were provided to facilitate the undertakings.

The work was split into three separate work packages (WPs), defined as:

- **WP1**: White-box pen.-tests & code audits against Raspberry Pi Connect web & infra
- **WP2**: White-box pen.-tests & code audits against Raspberry Pi ID web app UI & API
- **WP3**: White-box pen.-tests & code audits against Raspberry Pi GoLang client

All preparations were completed in December 2024, specifically during CW50, to ensure a smooth start for Cure53. Communication throughout the test was conducted through a dedicated and shared Slack channel, established to combine the teams of Raspberry Pi and Cure53. All personnel involved from both parties were invited to participate in this channel. Communications were smooth, with few questions requiring clarification, and the scope was well-prepared and clear. No significant roadblocks were encountered during the test. Cure53 provided frequent status updates and shared their findings through the aforementioned Slack channel. Live reporting was not specifically requested for this audit, but some details about findings were shared when they were discovered, so that a handful of issues could be directly addressed during the testing phase.

The Cure53 team achieved good coverage over the scope items, and identified a total of six findings. Of the six security-related findings, one was classified as a security vulnerability, and five were categorized as general weaknesses with lower exploitation potential.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Cure53 was very pleased with the overall results of this project. The limited number of issues identified, coupled with the low severities of all identified findings, can be interpreted as a positive sign with regard to the security of the inspected applications and client software. All in all, it can be concluded that the Raspberry Pi team has demonstrated a strong commitment to security. The strong security foundation observed during this engagement will provide a solid basis for future development. It is advised that continued focus on security best practices will be crucial to maintaining this strong security posture as the application evolves.

The report will now shed more light on the scope and testing setup, and will provide a comprehensive breakdown of the available materials. Next, the report will detail the *Test Methodology* used in this exercise. This is intended to show the client which areas of the software in scope have been covered, and which tests have been executed, despite only limited findings having been made. Following this, the report will list all findings identified in chronological order, starting with the *Identified Vulnerabilities* and followed by the *Miscellaneous Issues* unearthed. Each finding will be accompanied by a technical description, Proof-of-Concepts (PoCs) where applicable, plus any fix or preventative advice to action.

In summation, the report will finalize with a *Conclusions* chapter in which the Cure53 team will elaborate on the impressions gained toward the general security posture of the Raspberry Pi Connect web application, the Raspberry Pi ID authentication provider, and the Connect Golang client that runs on Raspberry Pi devices.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Scope

- **Pen.-tests & code audits against Raspberry Pi web apps, client app & infra**
  - **WP1:** White-box pen.-tests & code audits against Raspberry Pi Connect web & infra
    - **Main URL:**
      - https://connect-staging.raspberrypi.com/
    - **SSH Access to Connect Server**
      - connect@connect-staging-1.rpi.computer
      - Cure53's public keys were shared
  - **WP2:** White-box pen.-tests & code audits against Raspberry Pi ID web app UI & API
    - **Main URL:**
      - https://id-staging.raspberrypi.com/
  - **WP3:** White-box pen.-tests & code audits against Raspberry Pi GoLang client
    - **Github URLs:**
      - https://github.com/raspberrypi/connect
      - https://github.com/raspberrypi/identity-ruby
      - https://github.com/raspberrypi/identity
      - https://github.com/raspberrypi/connect-go
    - **Branch:**
      - Main
  - **Documentation:**
      - https://www.raspberrypi.com/documentation/services/connect.html
  - **Accounts registered by Cure53:**
    - U: niko+pi1@cure53.de
    - U: johannes+pi@cure53.de
    - U: ed+pi@cure53.de
    - U: ed+pi2@cure53.de
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Test Methodology

Given the lower number of exploitable findings uncovered during this test, this section's goal is to give insight into the testing process and overall methodology employed during the testing phase. This aims to ensure that a sufficient amount of coverage was reached throughout the given scope, and gives additional transparency into the testing performed by Cure53.

- Since the Raspberry Pi Connect application was the main scope of this engagement, the penetration testing phase began with a general web security-focused review of the platform. Here, it was important to cover classic web security issues, ranging from authentication problems and the session and cookie security of the *id-staging.raspberrypi.com* platform, to output validation issues such as Cross-Site Scripting (XSS) and server-side problems such as SQL injection (SQLi) via the underlying Ruby object relations.
- To cover XSS problems, the underlying Ruby on Rails, JavaScript and ERB template code was analyzed for common pitfalls, including the use of functions such as *innerHTML* without proper sanitization for user input. Dynamic testing involved injecting malicious payloads to observe application behavior and to assess the effectiveness of existing security measures.
- This phase also evaluated the effectiveness of the Content Security Policy (CSP), which acts as an additional layer of defense alongside input sanitization and validation. A minor flaw was identified where one of the CSP directives exhibited excessive permissiveness, potentially enabling the execution of JavaScript code (RSP-01-002) in the presence of an injection. Additionally, one HTML injection vulnerability was discovered where user input was reflected without proper escaping (RSP-01-003).
- Cross-Site Request Forgery (CSRF) testing focused on ensuring the presence and correct enforcement of anti-CSRF mechanisms (such as the *csrf_meta_tags* helper). By reviewing the underlying code and manipulating CSRF tokens on the live application, Cure53 verified the way in which CSRF attacks were mitigated.
- SQLi testing centered on how user input was handled within SQL queries. The team reviewed the codebase to identify all SQL queries, and verified the use of proper sanitization techniques such as *sanitize_sql_like* when constructing SQL statements. No instances of unsanitized user input being passed to *connection.execute* were found.
- Open redirect analysis focused on how user input could influence redirects to external endpoints. Methods used for application redirection were identified through code analysis and live testing, and were carefully examined for potential vulnerabilities. No instances of user input being passed to calls of the *redirect_to* method were found.
- In order to make sure that registered devices can only be accessed by the users or organizations that actually own them, Cure53 used a dynamic and source-assisted

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

approach to enumerate every route that could access devices directly, based on their UUID.

- Here, the Connect web application relies on calling the *accessible_devices()* method on the currently authenticated user context. Cure53 then ensured that calling this method in each controller initializer is not forgotten, so that access is properly restricted for device IDs belonging to other tenants.
- At first, this was done in a dynamic manner, where two accounts were used in parallel in two different browser sessions. One account enumerated all device functionalities, while the second account automatically copied and pasted their authentication cookies into the original request.
- This allowed for very fast enumeration of potential low-hanging fruit, where such a check might have been forgotten. However, no mistakes were found in this process, thereby indicating that Access Control List (ACL) checks have been well implemented.
- Since the device authentication flow relies on randomly-generated tokens and UUIDs, the generation of these elements was carefully reviewed. It was confirmed that the database's tokens and UUIDs were generated using cryptographically secure pseudorandom number generators (CSPRNGs), thereby ensuring that future values could not be predicted.
- Moreover, the implementation of the WebRTC setup was reviewed. As part of this process, the configuration of the self-hosted TURN server was examined for common misconfigurations, such as unauthorized access to internal services. However, dynamic testing utilizing tools such as Stunner[1], as well as a manual review of the TURN server configuration both confirmed that no such issues were present. Weak DTLS ciphers are disabled, credentials are rotated every 15 minutes, and a strong shared secret key is configured, all of which contribute to a robust security posture.
- In addition, the negotiation of SDP offers and answers was evaluated. The exchange through the key-value store was deemed secure, as it was adequately protected by ACL checks, thereby preventing unauthorized access by other users or devices.
- Further, various exploitation scenarios were evaluated. Initially, the team examined whether a malicious device could establish VNC or SSH sessions with other devices within the same account or across different accounts. While proper ACL checks generally prevented such actions, one issue was identified (see RSP-01-005). When combined with a login CSRF attack (RSP-01-004), this vulnerability could potentially allow an attacker to initiate an SSH session on another device.
- Next, an evaluation was conducted to determine whether a malicious device could escalate privileges by exploiting the API keys to access the privileged user API. However, it was confirmed that the device API keys were restricted to accessing only the device-specific API, thereby preventing privilege escalation.
- Furthermore, attempts to trigger XSS from the perspective of a malicious device - with the aim of gaining access to user accounts - were unsuccessful. The JavaScript

[1] https://github.com/firefart/stunner

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

dependencies in use - including Xterm.js and noVNC - were thoroughly examined for any known vulnerabilities. This review involved checking against the latest vulnerability databases and security advisories to ensure that no security issues were present.

- To ensure comprehensive coverage of the project's attack surface, the team's analysis extended to the codebase's supply chains. Recognizing that vulnerabilities within the development and third-party component ecosystems can be exploited in order to compromise the target application, Cure53 conducted a thorough review of dependencies, third-party components, and build systems.
- This review encompassed an assessment of potential vulnerabilities arising from package name or version conflicts (Dependency Confusion[2]). Given the presence of *package.json* files, the *@raspberrypi* organization scope on the public npm registry was checked. Since it is already claimed,[3] Dependency Confusion attacks are mitigated. Automated tooling identified known vulnerabilities (CVEs) in dependency files (*connect-main/Gemfile.lock*, *connect-main/yarn.lock*, etc.). Notably, *connect-go-main* lacked findings, likely due to the GitHub Workflow that regularly updates dependencies, thereby highlighting the importance of such practices. Docker was used for *connect-main* deployment in production.
- Container dependencies, referenced URLs, and build file contents were reviewed. The container did not run as root, and no secrets were stored within the build file. CircleCI was used by multiple projects with a read-only user for the GitHub package registry and code scans performed using Brakeman.
- Cure53 additionally leveraged SSH access to the *connect* staging server under *connect-staging-1.rpi.computer*, in order to identify potential infrastructure misconfigurations that could facilitate the escalation of vulnerabilities within the target web application. The assessment also aimed to uncover vulnerabilities related to the application's deployment process.
- Operating as the user (*connect*) configured to run the application, Cure53 gathered information about the operating system, running services, and server settings. Following the identification of a potential escalation path from the application user to root privileges on the staging server, Cure53 conducted an in-depth analysis of the operational Docker containers to identify any potential container escapes.
- This analysis aimed to ensure that code execution vulnerabilities within the target web application could not be escalated to full root access on the server. Fortunately, no container escape vulnerabilities were identified, as all containers were found to be operating with default security settings enabled, and were configured to run with restricted user privileges.

[2] https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610
[3] https://www.npmjs.com/~raspberrypi?activeTab=packages

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 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., RSP-01-001) to facilitate any future follow-up correspondence.

RSP-01-004 WP1: Login CSRF attack using Pi serial number *(Low)*

Fix-Note: *This issue was fixed during the testing phase and the fix was verified by Cure53.*

The existing device authentication flow relies on device serial numbers to initiate the process. Device authentication requires the user to click a verification link, before the server issues an API key for the corresponding device based on its serial number. It was found that this authentication flow is susceptible to a CSRF attack if the attacker possesses knowledge of the victim's device serial number. The attacker must entice the victim to click a malicious verification link. Upon a successful attack, the attacker's device becomes associated with the victim's account. Both the victim's and the attacker's devices are subsequently tracked as the same database entity. Consequently, when the victim attempts to connect to their device, there is a probability that they will inadvertently connect to the attacker's device.

This allows the attacker to potentially record keystrokes to capture sensitive information. Further investigation has revealed that the attacker can also initiate an SSH session on the victim's device, as detailed in RSP-01-005.

Steps to reproduce:

1. Leak another user's Pi serial number (e.g. via public forums).
2. Generate a verification link with the leaked serial number using the following request:

   PoC request #1:

   ```
   POST /client/device HTTP/2
   Host: api.connect-staging.raspberrypi.com
   Content-Type: application/x-www-form-urlencoded

   client_id=057C9305-A173-4596-BDAC-
   9701A92F7F62&login_hint=[SERIAL_NUMBER]
   ```

3. Store the *device_code* and the *verification_uri_complete* fields from the response.
4. Trick the authenticated victim into visiting the created verification URI (e.g. *https://connect-staging.raspberrypi.com/verify/VVJK-9GT1*).
5. Poll the API key using the *device_code* with the following request:

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

PoC request #2:

```
POST /client/token HTTP/2
Host: api.connect-staging.raspberrypi.com
Content-Type: application/x-www-form-urlencoded

client_id=057C9305-A173-4596-BDAC-9701A92F7F62&device_code=[...]
```

This attack is possible because no user interaction is required when signing in devices that are already registered to the user's account. Therefore, it is recommended to require users to explicitly grant permission for authentication, even for devices already registered to their account. This measure provides users with the opportunity to detect any fraudulent login attempts and to interrupt the authentication process.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit, but which 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, while a vulnerability is present, an exploit may not always be possible.

RSP-01-001 WP2: HTTP Origin header acceptance is too broad *(Low)*

Fix-Note: *This issue was fixed during the testing phase and the fix was verified by Cure53.*

During review of the CORS configuration on *https://id-staging.raspberrypi.com*, it was noticed that an exception is made for the */me.json* endpoint, which reflects any subdomain of the *raspberrypi.com* main domain within the *Access-Control-Allow-Origin* response header. This was tracked down to the following location within the underlying source base:

Affected file:
identity-main/config/initializers/cors.rb

Affected code:

```
Rails.application.config.middleware.insert_before 0, Rack::Cors do
  allow do
    origins %r{\Ahttps://(?:.+?)\.raspberrypi\.(?:com|lan)\z}
    resource "/me.json", headers: :any, credentials: true, methods: %i[get
options head]
  end
```

Essentially, this means that any subdomain can read the contents of *https://id-staging.raspberrypi.com/me.json* by executing a JavaScript GET request. The risk here is mainly attributed to the fact that a single XSS vulnerability on a random *raspberrypi.com* subdomain will be able to fetch */me.json* and thus collect victim email addresses.

Given the number of subdomains that are currently registered on the public internet, the chance of unintentionally exposing access to */me.json* seems rather high. It is advised that this issue can be mitigated by having an allow-list of trusted subdomains that rely on *https://id-staging.raspberrypi.com* as identity provider, instead of broadly accepting arbitrary ones.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

RSP-01-002 WP2: CSP can be bypassed due to wide *script-src* directive *(Low)*

Fix-Note: *This issue was fixed during the testing phase and the fix was verified by Cure53.*

While checking the deployed CSP on the *id-staging.raspberrypi.com* domain, it was noticed that an explicit *script-src* exception is made for *https://*.hcaptcha.com*. This was observed via the following CSP response header that is served on every route:

CSP:

```
Content-Security-Policy: default-src https://id-assets-
staging.raspberrypi.com; font-src https://id-assets-staging.raspberrypi.com
https://fonts.gstatic.com; img-src https://id-assets-
staging.raspberrypi.com data: https://www.gravatar.com https://*.wp.com;
object-src 'none'; script-src https://id-assets-staging.raspberrypi.com
https://hcaptcha.com https://*.hcaptcha.com 'nonce-
ca68c6ca8f9fd5bdde75af4f20bc4b0e'; frame-src https://id-assets-
staging.raspberrypi.com https://hcaptcha.com https://*.hcaptcha.com;
connect-src 'self' https://hcaptcha.com https://*.hcaptcha.com; style-src
https://id-assets-staging.raspberrypi.com https://fonts.googleapis.com
https://hcaptcha.com https://*.hcaptcha.com 'sha256-
WAyOw4V+FqDc35lQPyRADLBWbuNK8ahvYEaQIYF1+Ps=' 'nonce-
ca68c6ca8f9fd5bdde75af4f20bc4b0e'
```

Because *https://js.hcaptcha.com* allows the onloading of arbitrary JavaScript methods, this CSP XSS protection can be bypassed by including the following *<script src>* directive on the web page. In presence of an actual XSS vulnerability, the following injected payload would then trigger an alert-box:

Bypass:

```
<script                                src="https://js.hcaptcha.com/1/api.js?
onload=alert&render=explicit"></script>
```

It is recommended to fully rely on nonces only, without host-based allow-lists. To enforce this, it is recommended to additionally include the *strict-dynamic*[4] keyword.

[4] https://content-security-policy.com/strict-dynamic/

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

RSP-01-003 WP2: HTML injection in email address suggestion feature *(Info)*

Fix-Note: *This issue was fixed during the testing phase and the fix was verified by Cure53.*

During the review of the client-side JavaScript code, an insecure use of the *innerHTML* property was identified. Specifically, the email suggestion feature renders the email address entered into the signup form as HTML code. However, due to the deployed CSP and the requirement for manual editing of the email address to trigger the vulnerable code, this issue is considered to have limited exploitability. Consequently, it has been classified as informational only.

Steps to reproduce:

1. Visit the following URL, which auto-fills the malicious email address into the signup form:

   PoC URL:
   https://id-staging.raspberrypi.com/oauth/authorize?prompt=create&login_hint=test%3cmeta+http-equiv="refresh"+content="0;+url=https://example.com%22%3e@gmail.co

2. Edit the auto-filled email address by appending an additional character, and click on the window.
3. Observe that the HTML code is rendered into the DOM and that a redirect to *https://example.com* is performed.

The code excerpt below shows that the email address suggestion is rendered into the DOM using the *innerHTML* property:

Affected file:
identity-main/app/javascript/controllers/email_controller.js

Affected code:

```
check() {
    const [_localPart, domain] = this.emailTarget.value.split("@", 2);
[...]
    const suggestion = this.emailTarget.value.replace(
       `@${domain}`,
       `@${commonDomain}`,
    );
    this.feedbackTarget.innerHTML = `Did you mean ${suggestion}?`;
```

To mitigate this issue, it is recommended to use the *innerText* property instead of *innerHTML*. This ensures that the entered email address is treated as plaintext, rather than being interpreted as HTML code.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

RSP-01-005 WP1: Devices can initiate WebRTC connections *(Info)*

Fix-Note: *This issue was fixed during the testing phase and the fix was verified by Cure53.*

Following the discovery of RSP-01-004, it was determined that if two devices are logged in with the same serial number, specifically an attacker-controlled device and a victim device, then it is possible for the attacker to establish an SSH session with the victim's device.

This issue arises from the implementation of the exchange of SDP offers and answers. Typically, only the JavaScript client is intended to generate SDP offers, while devices are expected to continuously poll for offers and submit only SDP answers. However, due to the *KvPairsController* allowing devices to submit SDP offers, an attacker can exploit this behavior. Once a malicious SDP offer is sent by the attacker, the victim's device will retrieve the offer, generate an answer, and await the establishment of the connection.

It is important to note that this attack is not feasible when two devices with different serial numbers are connected to the same account.

The following request and corresponding response show that a malicious device can create SDP offers:

PoC request:

```
POST /keys/com.raspberrypi.connectd/connections/offers/eviloffer HTTP/2
Host: api.connect-staging.raspberrypi.com
Authorization: Bearer: rpdev_EVX8GyRhvP3CoivVpA9vDmML
[...]
{"type":"offer","sdp":"[...]"}
```

Response:

```
HTTP/2 204 No Content
Cache-Control: no-cache
[...]
```

It is recommended to restrict authenticated devices to creating only SDP answers, in order to prevent WebRTC connection establishment by malicious devices. This restriction helps to maintain the intended flow of communication and enhances the security of the device isolation.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

RSP-01-006 WP1: *connect* user gains root privileges through *docker* group *(Info)*

Cure53 leveraged SSH access to the staging server to identify potential infrastructure misconfigurations that could facilitate the escalation of vulnerabilities within the target web application. Operating as the user (*connect*) configured to run the application, a potential escalation path to root privileges was identified. The *connect* user was found to be a member of the *docker* group, granting it root-level privileges. As documented by Docker[5], membership in the *docker* group provides access to the Docker socket, requiring root permissions. This allowed the user to spin-up a container, mount the root filesystem, and subsequently write to the host machine's root filesystem to gain passwordless *sudo* privileges, thereby achieving root access on the staging server.

Subsequent to this finding, all Docker containers were reviewed to identify any potential container escapes that could be exploited in conjunction with this privilege escalation. This aimed to determine whether code execution vulnerabilities within the target web application could be leveraged to achieve full root access on the server.

Fortunately, no container escape vulnerabilities were identified, as all containers were operating with default security settings enabled, and were configured to run with restricted user privileges. Consequently, this finding was marked as informative.

PoC:

```
$ ssh connect@connect-staging-1.rpi.computer
connect@rptlsconnect:~$ docker -H unix:///var/run/docker.sock run -v
/:/host -it ubuntu chroot /host /bin/bash
root@0c03911c833e:/# echo "connect ALL=(ALL) NOPASSWD:ALL" >>
/etc/sudoers.d/connect
root@0c03911c833e:/# exit
connect@rptlsconnect:~$ sudo bash
root@rptlsconnect:/home/connect#
```

To mitigate the security risk noted here, it is advised that Docker in Rootless mode[6] could be utilised. This eliminates the need for root privileges, and would therefore not grant the *connect* user root privileges.

[5] https://docs.docker.com/engine/security/#docker-daemon-attack-surface
[6] https://docs.docker.com/engine/security/rootless/

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Conclusions

As noted in the *Introduction*, this January 2025 penetration test and source code audit was conducted by Cure53 against the Raspberry Pi Connect web application, the Raspberry Pi ID authentication provider, and the Connect Golang client that runs on Raspberry Pi devices.

From a contextual perspective, twenty working days were allocated to reach the coverage expected for this project. The methodology used conformed to a white-box strategy, and a team of four senior testers was assigned to the project's preparation, execution, and finalization.

Cure53's audit of the Raspberry Pi Connect web application revealed very few exploitable issues. Overall, this is indicative of an application complex that has been very well thought through, and a lean architecture that does not lend itself to errors. Combined with the solid codebase employed throughout the whole architecture - beginning with the Ruby on Rails code in the web frontend, right through to the Go clients on the Pi's themselves - the general consensus among the testing team was that security across the tested scope was solid.

Auditing began with testing of the connect application under *connect-staging.raspberrypi.com* (WP1) and the identity provider under *id-staging.raspberrypi.com* (WP2). This formed the testing team's main focus. Here, the absence of low-hanging vulnerabilities in the Ruby on Rails code was felt to demonstrate the team's effective usage of Brakeman. While not 100% comprehensive and accurate, Brakeman can uncover a range of classic Ruby on Rails security issues, as well as general oversights that can be made during the development process. While no issues were found to be present in the Ruby codebase, it is advised that this does highlight the importance of incorporating similar security tooling into the JavaScript development workflow. This would help to proactively identify and mitigate issues such as RSP-01-003.

Overall, the use of Ruby on Rails was felt to be a wise decision, given that it incorporates numerous robust security features, thereby making the implementation of security measures more straightforward. For example, enforcement of CSRF countermeasures was effectively achieved through the utilization of built-in Ruby on Rails functionalities. The same can be said of the relational database methods for all underlying objects within the codebase, which are automatically solid. This ensures that the writing of secure SQL queries is a standard practice. Although mistakes can be made when mixing certain methods with raw user input, no such issue was found within the tested codebase.

The client-side JavaScript code developed using the Stimulus framework underwent a thorough review for potential client-side vulnerabilities, including XSS and open redirects. Although the code generally made a robust impression on the team, a vulnerability involving HTML injection was identified within the signup form (RSP-01-003). However, the issue

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

could not be exploited to achieve XSS, due to the implementation of a CSP. This underlines the effectiveness of a multi-layered defense-in-depth strategy. Although the CSP has a bypass (as explained in ticket RSP-01-002), this cannot be used for the HTML injection issue, thus meaning that the impact of RSP-01-003 is informational in nature only.

After having covered potential web security weaknesses throughout the platform, special attention was then directed towards the implementation of WebRTC communication, particularly the exchange of SDP offers and answers, which is a critical system component and therefore represented a core part of the scope for this assignment. The exchange was thoroughly reviewed for potential vulnerabilities that could permit malicious users unauthorized access to other users or organizational devices. However, the results of both the source code review and dynamic testing confirmed the robustness of the ACL checks in place. The additionally-included *Test Methodology* section of this report supports the conclusion that the developers have adhered to the security best practices prescribed by the underlying architecture here.

During additional deep-dives into more theoretical attacks, two minor issues were identified, which could be exploited through a CSRF attack if the serial number of a victim's device was known (RSP-01-004, RSP-01-005). The development team promptly provided a patch for both issues by implementing an additional confirmation prompt, the effectiveness of which was verified by Cure53. This swift response underlines the development team's commitment to addressing security concerns, and demonstrates that even minor issues are taken seriously and resolved with due diligence.

As a final scope item, Cure53 verified the security of the supply chain, as well as the employment of the GitHub package registry with read-only user access from the automation pipeline. This was found to significantly reduce available attack surface. Furthermore, the team found that the *@raspberrypi* organization had already been claimed on the public npm registry, thereby effectively mitigating Dependency Confusion attacks against the Raspberry Pi team’s JavaScript dependencies.

Some known vulnerabilities - specifically Common Vulnerabilities and Exposures (CVEs) - were identified in certain Ruby and JavaScript dependencies. However, none of these vulnerabilities were deemed to be actively exploitable. This was primarily due to the fact that the affected and vulnerable methods were either not utilized, or did not present a significant enough risk for active exploitation. Nonetheless, it is advisable to maintain up-to-date dependencies in order to minimize any unnecessary risks.

All-in-all, Cure53 has been very happy with this project, including the overall test preparation, all discussions that took place throughout the testing phase, as well as the admirable overall result. The low number of issues uncovered here speaks for itself, and the additionally included *Test Methodology* section of this report highlights more potential attack vectors that ultimately did not bear fruit for the testing team.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Raspberry Pi has developed a solid application that includes few to no pitfalls or exploitable attack vectors. Cure53 hopes that the robust foundation the team has built will continue to persist in the future, as the codebase is further developed.

Cure53 would like to thank Paul Mucur and Gordon Hollingworth from the Raspberry Pi Ltd team for their excellent project coordination, support and assistance, both before and during this assignment.