Fine penetration tests for fine websites

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

Pentest-Report Project 11 Web App UI, API & Infra 06.2025

Cure53, Dr.-Ing. M. Heiderich, T. Orlita

Index

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

Introduction

This report, identifiable as P11-02, presents the outcomes of a penetration test and source code audit against the Project 11 web application, UI, and REST API, as performed by Cure53 in early June 2025.

For background information, representatives from Project 11 Limited contacted Cure53 in May 2025 to request the assessment and specify the overall aims. The initiatives were completed over a one-week period (CW23) by a two person review team. Five days were allocated for the analysis, which was deemed an ample time frame to achieve the expected coverage and yield of results.

Three individual Work Packages (WPs) were created for the examinations, denoting the key areas of interest. These read as follows:

- **WP1**: White-box pen.-tests & code audits against Project 11 web UI
- **WP2**: White-box pen.-tests & code audits against Project 11 REST API
- **WP3**: White-box pen.-tests & reviews against website infrastructure & config

To facilitate the white-box initiatives, the Project 11 maintainers provided a suite of materials, including URLs, sources, documentation, and other assorted assets. All preparations were completed in late May 2025 (CW22) to ensure a seamless start.

Communication throughout the assignment occurred via a dedicated Slack channel, which included all relevant personnel from both Project 11 and Cure53. The cross-team discourse was generally seamless, with minimal need for clarification as the scope was clearly defined and well-prepared. No significant obstacles arose during the testing period.

Cure53 provided regular status updates on the progress and identified findings. Live reporting was also offered and deemed beneficial for this exercise, conducted via the designated Slack channel.

Following satisfactory depth and breadth of coverage over the scope elements, Cure53 detected and documented a total of five findings in ticket format. Four were categorized as security vulnerabilities, while the remaining ticket was filed as a miscellaneous weakness.

In sum, the Project 11 application exhibits a robust security foundation under the current construct. The deployed components indicate the dev team’s thoughtful security design, presenting minimal frontend attack surface and clean, readable code that inherently reduces risk.

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

In addition, the infrastructure is commendably architected and affected by only minor workflow shortcomings. Despite diligent efforts, Cure53 could not locate any significant findings related to its core setup or deployment.

Nevertheless, Cure53 noted some areas that would benefit from improvement. All of the corresponding tickets were assigned a severity rating of *Medium* or lower. Moreover, the Project 11 developers acted immediately and swiftly resolved the detected rate limiting vulnerability upon identification, highlighting their commitment to security proficiency.

The report will now provide insights into the *Scope* and testing setup, as well as display a comprehensive breakdown of all available materials in bullet point form. Subsequently, the report will list all findings identified in chronological order, starting with the *Identified Vulnerabilities* and followed by the *Miscellaneous Issues*. Each finding will be accompanied by a technical description and Proof of Concepts (PoCs) where applicable, plus any relevant mitigatory or preventative advice to action.

In summation, the report will finalize with a conclusion in which the Cure53 team will appraise the general security posture of the elements in focus, offering high-level hardening advice and next steps for the internal team.

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

Scope

- **Pen.-tests & code audits against Project 11 web application UI, API & infra**
  - **WP1:** White-box pen.-tests & code audits against Project 11 web UI
    - **Source code:**
      - **URL:**
        - https://github.com/p-11/yellowpages-client
      - **Branch:**
        - *development*
      - **Commit:**
        - *d0191650d61778119cc018c6b554e9dffd3adce9*
    - **Production environment:**
      - https://www.yellowpages.xyz/
  - **WP2:** White-box pen.-tests & code audits against Project 11 REST API
    - **Data API:**
      - **Source code:**
        - **URL:**
          - https://github.com/p-11/yellowpages-data-layer-service
        - **Branch:**
          - *development*
        - **Commit:**
          - *b1e0742f56c238e71fc24552423f8207f3236de5*
    - **Proof API:**
      - **Source code:**
        - **URL:**
          - https://github.com/p-11/yellowpages-proof-service
        - **Branch:**
          - *development*
        - **Commit:**
          - *81fdb0ac6ceac0b213856516ea1a8bcba6cb866f*
      - **Production environment:**
        - https://yellowpages-proof-service.app-1312b66384d.enclave.evervault.com
    - **Verification API:**
      - **Source code:**
        - **URL:**
          - https://github.com/p-11/yellowpages-verification-service
        - **Branch:**
          - *development*
        - **Commit:**
          - *79302565a9f95f5b28a08f68047a6ca5d4645e37*
      - **Production environment:**
        - https://verification-api.yellowpages.xyz/

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

- **WP3:** White-box pen.-tests & reviews against website infrastructure & config
  - **Primary focus areas:**
    - **Frontend:**
      - Deployed using Vercel
    - **Proof service:**
      - Evervault Enclaves (uses AWS Nitro Enclave)
    - **Data & verification services:**
      - Deployed to AWS using Typescript AWS CDK infra scripts
      - Elastic Container Repository
      - Elastic Container Service cluster
      - Application Load Balancer
      - CI/CD using GitHub Actions
    - **Mongo DB:**
      - Mongo Atlas
      - IP whitelisting, accessible only by the data service
- **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

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., *P11-02-001*) to facilitate any future follow-up correspondence.

P11-02-001 WP3: Command injection in GitHub Actions *(Low)*

Fix note: *This issue was fixed during the testing phase. Cure53 verified the fix, confirming that the problem no longer exists.*

While reviewing the GitHub Actions workflows, Cure53 discovered that the *restrict_pr_branches* workflow, present in the service and client repositories, is vulnerable to script injection attacks. Specifically, the workflow directly interpolates the *github.head_ref* context variable into shell commands without sufficient escaping. This variable is utilized for the branch name of the originating pull request, which can contain special characters. As such, substituting it directly into the script permits injecting arbitrary shell commands that will be executed by the GitHub Actions runner.

Affected file:
.github/workflows/restrict_pr_branches.yml

Affected code:

```
steps:
  - name: Check source branch
    run: |
      echo "Base branch: ${{ github.base_ref }}"
      echo "Head branch: ${{ github.head_ref }}"
      if [ "${{ github.head_ref }}" != "development" ]; then
        echo "   Pull requests into 'main' must come from 'development'."
        exit 1
      fi
```

This activity could be exploited by a user that is permitted to submit pull requests to the GitHub repository. For example, naming the source branch *;{cat,/etc/passwd};* will cause the string to break out of the *echo* statement and be interpreted as code.

The impact of command injection in this scenario is limited, as the attacker would need access to the private repository. However, it should still be resolved to prevent privilege escalation and shield against future alterations to repository permissions or unforeseen attack vectors.

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

To mitigate this vulnerability, Cure53 advises leveraging intermediate environment variables[1] to pass values into inline shell scripts for all variables with potentially untrusted input.

P11-02-003 WP1: Lack of general HTTP security headers *(Low)*

Fix note: *This issue was fixed during the testing phase. Cure53 verified the fix, confirming that the problem no longer exists.*

Testing confirmed that the *yellowpages-client* website lacks certain HTTP security headers in HTTP responses. This does not directly evoke security risk, but could aid attackers in their efforts to exploit other areas of weakness. The following list enumerates the headers that require reviewing and implementing in order to prevent associated flaws.

- **X-Frame-Options**: This header specifies whether the web page is allowed to be framed. Although this header is known to prevent clickjacking attacks, a plethora of alternative breach strategies are achievable when a web page is framable[2]. Cure53 recommends configuring the value to either *SAMEORIGIN* or *DENY*.
- **Content-Security-Policy**: This header is used to control which resources the web page is allowed to load. By restricting the execution of code and the resources that can be loaded on the page, CSP provides an additional layer of security against attacks such as Cross-Site Scripting and CSS injection.
- Notably, the CSP framework offers similar protection to X-Frame-Options via methods that overcome some shortcomings of the aforementioned header. To optimally protect users of older browsers and modern browsers simultaneously, Cure53 recommends deploying the *Content-Security-Policy: frame-ancestors 'none';* header in addition.

All in all, the neglect to incorporate beneficial security headers is suboptimal and should be avoided.

To mitigate this issue, Cure53 advises inserting the aforementioned headers into all server responses, including error messages such as *4xx* items.

[1] https://docs.github.com/en/actions/security-for-github-actions/security-guides/security-[...]-variable
[2] https://cure53.de/xfo-clickjacking.pdf

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

P11-02-004 WP1: Lack of cross-origin-related HTTP security headers *(Low)*

Fix note: *This issue was fixed during the testing phase. Cure53 verified the fix, confirming that the problem no longer exists.*

Cure53 discovered that the *yellowpages-client* website lacks cross-origin-infoleak-related HTTP security headers[3] in its responses. Similarly to the previous ticket, this circumstance does not evoke direct security implications. However, attackers may be encouraged to exploit other areas of weakness due to the suboptimal protection, such as issues relating to Spectre attacks[4]. The following headers were verified to be absent from the construct and should be implemented to prevent associated vulnerabilities.

- **Cross-Origin Resource Policy (CORP)** and *Fetch Metadata Request* headers allow developers to control which sites can embed their resources, such as images or scripts. They prevent data from being delivered to an attacker-controlled browser-renderer process, as seen in *resourcepolicy.fyi* and *web.dev/fetch-metadata*.
- **Cross-Origin Opener Policy (COOP)** grants developers the ability to ensure that their application window will not receive unexpected interactions from other websites, allowing the browser to isolate it in its own process. This incorporates important process-level protection, particularly in browsers that do not enable full Site Isolation; see *web.dev/coop-coep*.
- **Cross-Origin Embedder Policy (COEP)** ensures that any authenticated resources requested by the application have explicitly opted-in to passing into a load state. In the current climate, to guarantee process-level isolation for highly sensitive applications in Chrome or Firefox, applications must enable both COEP and COOP; see *web.dev/coop-coep*.

Generally speaking, the absence of cross-origin security headers should be considered a negative practice that could be avoided in times when attacks such as Spectre are known to be easily practicable and exploit code is publicly available.

To mitigate this issue, Cure53 recommends inserting the aforementioned headers into every relevant server response. Resources with detailed information regarding headers of this nature are available online, explaining both header setup best practices[5] and the potential consequences of neglecting to install them entirely.[6]

[3] https://security.googleblog.com/2020/07/towards-native-security-defenses-for.html
[4] https://meltdownattack.com/
[5] https://scotthelme.co.uk/coop-and-coep/
[6] https://web.dev/coop-coep/

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

P11-02-005 WP2: DoS via subpar IP rate limit configuration *(Medium)*

***Fix note**: This issue was fixed during the testing phase. Cure53 verified the fix, confirming that the problem no longer exists.*

While evaluating the *yellowpages-proof-service* rate limiting configuration, Cure53 determined that the imposed IP-based rate limit protection fails to correctly retrieve the client's IP address. Specifically, as this service is deployed behind a reverse proxy using Evervault, the IP address employed for rate limiting corresponds to the proxy's IP address, rather than the client's actual IP address. Therefore, the IP-based rate limit safeguarding will effectively function as a global rate limiter and could unintentionally block legitimate requests.

The current setup leverages a global and IP-based rate limiter. The global rate limiter (*general_rate_limiter*) is configured to allow 1000 requests per minute in total. The IP-based rate limiter (*ip_rate_limiter*) allows 10 requests every two seconds per IP address. The IP-based rate limiter is implemented as a middleware using the *tower_governor* Rust crate. The default *Governor* config[7] is utilized, which applies the *PeerIpKeyExtractor*[8] by default in order to determine the IP address to rate limit.

Affected file:
yellowpages-proof-service/src/main.rs

Affected code:

```
let governor_conf = Arc::new(
    GovernorConfigBuilder::default()
        .per_second(2)
        .burst_size(10)
        .finish()
        .unwrap(),
);
```

As noted above, the peer IP will be the same for all requests, meaning that legitimate requests could be unintentionally rate limited during high traffic periods. Alternatively, this could potentially allow a single attacker to DoS the service for other users. This was verified by repeatedly sending *GET /prove* requests from a device to trigger the IP-based rate limiter, then attempting to send the same request from a second device with a different IP address, issuing a *429 Too Many Requests* response.

While the IP-based rate limiter is approximately three times stricter than the global rate limiter on average, increasing the global rate limiter to allow increased traffic offers negligible benefits and will likely lead to security implications in the future if unaddressed.

[7] https://docs.rs/tower_governor/latest/tower_governor/governor/struct.GovernorConfigBuilder.html#[...]
[8] https://docs.rs/tower_governor/latest/tower_governor/key_extractor/struct.PeerIpKeyExtractor.html

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

To mitigate this vulnerability, Cure53 recommends adopting the actual IP address of the client from the appropriate server headers. This can be accomplished by implementing a custom *KeyExtractor* to retrieve the appropriate header, such as *X-Forwarded-For*, containing the client's IP address. Notably, sending a request with this header will not overwrite the header configured by the proxy.

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 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.

P11-02-002 WP3: Lack of commit pinning in GitHub Actions *(Info)*

Fix note: *This issue was fixed during the testing phase. Cure53 verified the fix, confirming that the problem no longer exists.*

While exploring the GitHub Actions workflows across all repositories, Cure53 acknowledged that third-party actions are pinned to a version tag, rather than a specific commit. While version tag pinning is a common practice, the full commit SHA should be utilized for supplemental protection against supply chain attacks, as it directly references a unique point in the Git history.[9]

Affected file:
yellowpages-data-layer-service/.github/workflows/deploy_prod.yml

Affected code:

```
- name: Set up Docker Buildx
  uses: docker/setup-buildx-action@v3

- name: Configure AWS credentials
  uses: aws-actions/configure-aws-credentials@v4
```

To mitigate this vulnerability, Cure53 recommends introducing commit pinning as an additional defense-in-depth measure against potential supply chain attacks, even though the third-party actions present in the analyzed workflows are maintained by established actors.

[9] https://docs.github.com/en/actions/security-for-github-actions/security-guides/security-[...]-actions

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

Conclusions

The *Conclusions* chapter compiles all observations yielded during the pentesting period, summarizing the overall verdict and offering actionable next steps. In summary, Cure53 is pleased to report that the scrutinized Project 11 web app and associated features exhibit performant safeguarding at present, although certain aspects can be augmented for encompassing defense.

Regarding the coverage for WP1, the Cure53 consultants conducted a thorough analysis of the web application frontend, which is constructed using React and Next.js. These are established frameworks that provide secure default paradigms and effective foundations for client-side development.

The web application is deployed with Vercel, which automatically configures HTTP Strict Transport Security (HSTS), ensuring protection against protocol downgrade attacks. However, other HTTP security response headers have not been configured, such as CSP and XFO (P11-02-003). Additionally, newer security headers related to browser-level isolation should be incorporated, such as CORP and COOP (P11-02-004). Incorporating these headers is strongly recommended to provide auxiliary shielding against client-side attacks.

Elsewhere, Cure53 noted limited rendering of user-provided data and avoidance of potentially dangerous JavaScript sinks, which provides steadfast protection against client-side injection pathways. Web Workers are utilized for cryptographic operations, providing performance advantages and additional separation from the main thread. Moreover, the NPM dependencies are up-to-date, eliminating any threats that could arise due to outdated libraries. To finalize, the clean and minimal code, combined with comprehensive end-to-end tests, provides a solid foundation. Implementing the aforementioned security headers will help to strengthen the overall security posture.

Regarding the coverage for WP2, Cure53 systematically explored the three backend services in scope, all of which offer a healthy security posture. The utilization of Rust provides type safety and prevents memory flaws, significantly negating the vast array of plausible attacks in this area. Access to the data layer service is restricted to specific internal services and enforced via API keys. Each endpoint called by a different service requires an alternate key, maintaining the principle of least privilege.

Nonetheless, supplementary security hardening can be achieved by implementing dynamic authentication methods, such as short-lived tokens or public-key cryptography, rather than static key-based authentication. Additionally, the server provides verbose error messages while parsing the request body prior to checking the API key. To prevent disclosing the expected structure of the request body, verbose error messages should be disabled or the key initially validated.

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

The proof service leveraged for websocket frontend connection enforces Cloudflare Turnstile verification, which mitigates automated bot abuse. A *Medium* severity vulnerability was discovered pertaining to the rate limit configuration, which could lead to a DoS condition if unresolved (P11-02-005). On a positive note, this pitfall was promptly nullified by the in-house team during the testing period and fix verified by Cure53.

Cross-Origin Resource Sharing (CORS) and verification of the websocket request origin header are enforced, ensuring that only permitted websites can connect to this service. Comprehensive end-to-end tests were implemented, guaranteeing that the service's functionality operates as intended and invalid input is appropriately handled.

Similar to the other variants, the verification service presents a robust security posture, as CORS restricts the allowed origins and extensive unit/integration tests verify the expected code behavior. Rate limiting is enforced for all services, guarding against brute-force attacks and other automated abuse vectors.

The current logging implementation is useful for debugging but could introduce a potential security risk by recording request parameter values. This setup should be revised to ensure that sensitive values are not logged. Across all services, request parameters are stringently validated before they are processed, limiting the attack surface for vulnerabilities caused by malformed data.

In summary, despite the sole finding and recommendations for security modifications, the backend services offer a solid foundation for the application's ongoing development, with key security principles established.

Regarding the coverage for WP3, the infrastructure of the website and backend services was vetted to pinpoint any prevalent security defects or misconfigurations. The data and verification services leverage infrastructure-as-code scripts, which are stored within their respective repositories. This approach is highly advantageous, as it provides a clear and auditable definition of the infrastructure, as well as monitors all amendments via version control. Furthermore, the project adopts a robust CI/CD pipeline using GitHub Actions to automate deployments to AWS and Enclave. This automation is fundamental for security purposes, as it asserts that deployments are consistent and repeatable.

A minor limitation was detected in a workflow present across repositories, caused by direct interpolation of context variables (P11-02-001). While the impact is constrained, it is important to reinforce the use of intermediate environment variables, as is correctly performed in other workflows. Additionally, third-party actions should be pinned to a specific commit rather than a version tag for defense-in-depth protection against supply chain attacks (P11-02-002).

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

AWS Secrets Manager and GitHub secrets are employed to create a resilient and maintainable system, allowing for secure deployments across various environments. All repositories and deployed services are strictly divided into distinct development and production environments. This is a critical control that isolates development and testing activities from the live system, preventing untested code from impacting users and protecting the integrity of production data.

In summary, Cure53's examination of the application components in scope for this review confirms the Project 11 team's proactive approach to security. The frontend codebase exposes minimal attack surface, although the installation of beneficial security headers will offer supporting defense. The codebase, written in TypeScript and Rust, is readable, organized, and written to a first-rate standard. The infrastructure is effectively architectured, with only two minor shortcomings identified related to workflows. The infrastructure setup and deployment process are not affected by any major security drawbacks.

The swift remediation of the rate limiting vulnerability in the proof service is commendable. Additional security hardening improvements were also introduced by the development team during the engagement. While several security weaknesses were located, none exceeded an impact score of *Medium*, reflecting a steadfast security foundation. Considering the established security controls and security-conscious approach to development, the inspected scope garnered a favorable verdict on the whole.

Cure53 would like to thank Conor Deegan and David Nugent from the Project 11 Limited team for their excellent project coordination, support, and assistance, both before and during this assignment.