Security assessments for software that matters.

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

# Pentest-Report Forward Email Code, Nodemailer, Infra 08.2026

Cure53, Dr.-Ing. M. Heiderich, MSc. H. Mösl-Canaval, Dipl.-Ing. D. Gstir, MSc. R. Peraglie

## Index

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

## Introduction

*“Forward Email is a free and open-source email forwarding service focused on a user's right to privacy. What began as a simple email forwarding solution in 2017 has evolved into a comprehensive email platform offering unlimited custom domain names, unlimited email addresses and aliases, unlimited disposable email addresses, spam and phishing protection, encrypted mailbox storage, and numerous advanced features.”*

From https://forwardemail.net/en/about

This report describes the results of a penetration test and source code audit conducted against the core codebase and underlying infrastructure of Forward Email, as well as the Nodemailer codebase.

To give some context regarding the assignment’s origination and composition, Forward Email LLC contacted Cure53 in June 2026. The test execution was scheduled for August 2026, namely from CW32 - CW34. A total of twenty-five days were invested to reach the coverage expected for this project, and a team of four senior testers was assigned to its preparation, execution, and finalization.

The methodology conformed to a white-box strategy, whereby assistive materials such as source code 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 Forward Email codebase
- **WP2**: White-box pen.-tests & code audits against Forward Email infra & IaC
- **WP3**: White-box pen.-tests & code audits against Nodemailer codebase

This was not the first time that the Forward Email complex had been inspected by Cure53. Some parts of this scope were also the focus of a previous audit held in May 2026 (see *FWD-01*).

All preparations for *FWD-02* were completed in late July 2026, specifically during CW31, 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 Forward Email 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, shared its findings, and offered live reporting through the aforementioned Slack channel.

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

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

Overall, it is advised that the number of findings made in this report can be seen as a high amount, and that this on its own should already be seen as a concerning sign with regard to the security of the inspected Forward Email components. Further to this, it is concerning that Cure53 unveiled several *High* and *Critical* severity vulnerabilities. This indicates that more effort should be invested in properly strengthening the Forward Email codebase and infrastructure against severe threats. This can likely be attributed to the fact that compared to the previous audit, Cure53 was able to perform a much deeper analysis of the system due to deeper knowledge from the first audit and a larger budget. Furthermore, this test also widened the scope significantly to include core third-party dependencies.

Once all of the listed findings have been addressed, it is strongly recommended to reinspect both the codebase and the infrastructure. This will enable the correctness of the applied fixes to be verified, and will help to ensure that no further vulnerabilities are yet to be discovered.

The report will now shed more light on the scope and testing setup, and will provide a comprehensive breakdown of the available materials. 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, and 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 core codebase and underlying infrastructure of Forward Email, as well as the Nodemailer codebase.

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

## Scope

- **Pen.-tests & audits against Forward Email sources, Nodemailer & infrastructure**
  - **WP1:** White-box pen.-tests & code audits against Forward Email codebase
    - **Source code repository:**
      - URL:
        - https://github.com/forwardemail/forwardemail.net
      - Commit:
        - ba132b590796f5620efb2339b5fe0a9476c8ea08
  - **WP2:** White-box pen.-tests & code audits against Forward Email Infra & IaC
    - **Source code repository:**
      - URL:
        - https://github.com/forwardemail/forwardemail.net
      - Commit:
        - ba132b590796f5620efb2339b5fe0a9476c8ea08
  - **WP3:** White-box pen.-tests & code audits against Nodemailer codebase
    - **Source code repository:**
      - URL:
        - https://github.com/nodemailer/nodemailer
      - Commit:
        - a208a0bc86a315d037d5be8849fab5862c488baa (v7.0.12)
  - **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., FWD-02-001) to facilitate any future follow-up correspondence.

### FWD-02-001 WP3: Prototype Pollution in *mailauth* lib. allows RCE on MX (*Critical*)

***Fix note**: This issue was fixed by Forward Email through a custom patch to mailauth to filter dangerous property keys. Cure53 verified the fix, confirming that the problem no longer exists.*

It was found that the Forward Email MX server suffers from a JavaScript Prototype Pollution vulnerability within the *mailauth* library, that can be leveraged into full Remote Code Execution (RCE) through a gadget in the *nodemailer* component. This can be done from an unauthenticated remote perspective, by feeding a malicious email with a specially-crafted DKIM header into the MX component. For this reason, this vulnerability is rated as being of *Critical* severity.

Below is shown a source code excerpt of the latest version of the *mailauth* library (version *v4.13.3* at the time of writing), which is responsible for parsing the value parts within the DKIM-related email headers received by the Forward Email MX server from a mail client. After parsing the header value into individual key and value parts, the logic traverses these key-value pairs, and intends to aggregate them into a single object, referenced by the *result* variable, by effectively treating the key as a JSON path to set the value on the aggregated result object.

This is done by iterating each key, splitting it into segments on dot characters (.), and then walking the result object down. Each segment of the split becomes the property value in a member access read performed on the currently walked JavaScript object referenced by the *curRes* variable. However, it is not validated that the property value is not an internal JavaScript member (such as *__proto__*). This can result in the JavaScript prototype object being walked by the logic before the attacker-chosen properties are set there, resulting in a Prototype Pollution vulnerability.

**Affected file - Prototype Pollution in *mailauth*:**
*https://github.com/postalsys/mailauth/blob/3b7515df768ce1d2e4e02858fdfca8efca6243fb/lib/parse-dkim-headers.js#L99-L109*

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

**Affected code - Prototype Pollution in *mailauth*:**

```
const valueParser = str => {
  [...]
  const parse = () => {
    [...]
    parts.slice(1).forEach(part => {
      if (part.key || part.value) {
          let path = part.key.split('.');
          let curRes = result;
          let final = path.pop();
          for (let p of path) {
              if (typeof curRes[p] !== 'object' || !curRes[p]) {
                  curRes[p] = {};
              }
              curRes = curRes[p];
          }
          curRes[final] = part.value;
      }
```

The following snippet shows the DKIM header lines of subsequent MIME emails that are sent to the MX server in a single SMTP session by an attacker and then parsed by the vulnerable function provided by *mailauth*. It can be observed that the header value contains highlighted key-value pairs. All key-value pairs include a key that has *__proto__* as the first dot-separated section. This will cause the next section to be assigned as a property name to the general object prototype. In this case, the *sendmail*, *path*, and *resolvedValue* properties are set, causing all objects to inherit these properties through the prototype chain.

**PoC - DKIM headers of MIME emails exploiting Prototype Pollution:**

```
ARC-Authentication-Results: i=1; mx.example.net; dkim=pass
__proto__.sendmail=yes __proto__.path=/usr/bin/tee
__proto__._resolvedValue="id > /tmp/PWNED.txt ; uname -a >> /tmp/PWNED.txt
#"

ARC-Authentication-Results: i=1; mx.example.net; dkim=pass
__proto__.sendmail=yes __proto__.path=/bin/bash
```

The properties are later inherited by an options object that configures the *nodemailer* library to deliver the inbound email via Sendmail, while executing an attacker-chosen binary (through the *path* property), instead of the Sendmail binary, which is fed attacker-controlled data through the *_resolvedValue* property. This allows the attacker to execute the Linux utility tool *tee*, to drop a file with attacker-controlled input in the current working directory. This is then executed via Bash upon the arrival of a second email.

**Affected file - *nodemailer* RCE gadget:**
*https://github.com/nodemailer/nodemailer/blob/cb4f904a53d2c2feeaf327203c92378d46304398/lib/sendmail-transport/index.js*

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

**Affected code - *nodemailer* RCE gadget:**

```
const { spawn } = require('child_process');
[...]
class SendmailTransport {
  constructor(options) {
    options = options || {};
    this._spawn = spawn;
    [...]
      if (options.path) {
          this.path = options.path;
      }
    [...]
  }
  send(mail, done) {
    [...]
    const args = this.args
        ? ['-i'].concat(this.args).concat(envelope.to)
        : ['-i'].concat(envelope.from ? ['-f', envelope.from] :
[]).concat(envelope.to);
    [...]
    sendmail = this._spawn(this.path, args);
```

It is advised that ideally, all JavaScript code - including the *mailauth* library - should abandon dynamically-computed JavaScript member access, and should instead use alternatives such as *Map* instances with the *set* and *get* methods. By doing this, Prototype Pollution - which relies on the internal *__proto__* JavaScript property - could only occur through static member accesses. These are easy to spot, and much less likely to be introduced.

Further, it is advised that an additional hardening mechanism can hinder some Prototype Pollution vulnerabilities through the *__proto__* property, by setting the *--disable-proto* option to *delete* when starting all relevant NodeJS processes[1].

## FWD-02-002 WP1: Open redirect via *return_to* parameter *(Low)*

***Fix note**: This issue was fixed by Forward Email by tightening the URL parsing logic. Cure53 verified the fix, confirming that the problem no longer exists.*

It was found that the application suffers from an open redirect vulnerability through the *return_to* parameter. This could be used by attackers to lure a victim onto a URL with a trust-inducing host part, which will then intransparently redirect away from the page to another user interface.

**PoC:**
*https://forwardemail.net/en/login?return_to=%2F%5Ccure53.de*

[1] https://developer.mozilla.org/en-US/docs/Web/Security/Attacks/Prototype_pollution

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

It is recommended to always validate the URL, and to ensure that it is an element of a predefined static list of trusted paths, before using it in a redirect. Doing this will mean that attackers can only choose to redirect to a path from the trusted list, instead of supplying URLs with malicious schemes.

### FWD-02-003 WP1: OTP bypass - second factor deleted with password only *(Medium)*

***Fix note**: This issue was fixed by Forward Email by requiring a 2FA token or recovery key for disabling 2FA. Cure53 verified the fix, confirming that the problem no longer exists.*

It was found that the Forward Email web application allows the second OTP factor to be avoided and deleted, simply by sending the valid password to */otp/disable*. This effectively removes the second factor in the event that the password of a user is compromised.

**Steps to reproduce:**

1. An attacker authenticates on the Forward Email instance, using the email and password credentials for a victim user.
2. The victim has OTP enabled, and the attacker is therefore forwarded to the OTP user interface.
3. The attacker sends the following HTTP request, authenticated with the session cookies of the browser that contains the password of the victim, in the highlighted JSON object:

   **HTTP request:**

   ```
   POST /en/otp/disable HTTP/2
   Host: forwardemail.net
   Cookie: locale=en; locale.sig=7KSTPLTadnyU856kb97usovZo5M;
   lad.sid=2959113b6ed49b74eb5652f28e01bbd9;
   lad.sid.sig=wAYiPJpn5bHEaptPIBOCwhlPWUo
   User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:153.0)
   Gecko/20100101 Firefox/153.0
   Accept: application/json
   Content-Type: application/json
   Content-Length: 33

   {"password":"Qbjo[...]TKC"}
   ```

4. The server will respond with a HTTP *200* status code. The OTP is now disabled, and the attacker can refresh the page without being interrupted for an OTP token.

It is recommended to ensure that privileged endpoints - such as those served under the */otp/** route - all require that the session is authenticated with a second factor. This could be achieved by always invoking the *ensureOtp* policy function for both the web and API endpoints, with the exception of individually-marked endpoints.

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

This will mean that accidentally omitting the *ensureOtp* policy function on a dedicated endpoint will be much more difficult, while it will still be possible to carve out exceptions for selected endpoints, if needed.

## FWD-02-004 WP2: Systemd timers pass passwords in command line *(Medium)*

***Fix note**: This issue was fixed by Forward Email by passing the secrets via a file instead. Cure53 verified the fix, confirming that the problem no longer exists.*

Multiple occurrences were found where passwords are passed as command line arguments, which are readable by other unprivileged users of the server operating system at run time. This could be abused in order to access the underlying services (such as the internal Redis server), or to decrypt the backup data, effectively allowing lateral movement from a compromised unprivileged Linux user.

**Affected file:**
*ansible/playbooks/redis.yml*

**Affected code:**

```
- name: Create Redis backup script
  copy:
    dest: /usr/local/bin/backup-redis.sh
    mode: "0755"
    content: |
      #!/bin/bash
      [...]
      if [ -n "${REDIS_PASSWORD}" ]; then
        VALKEY_CMD="${VALKEY_CMD} -a ${REDIS_PASSWORD}"
      [...]
      # Run encryption and upload with error capture
      if ! gpg --symmetric --cipher-algo AES256 --batch --yes --passphrase
"$BACKUP_SECRET" < "${RDB_FILE}"
```

**Also affected:**

- *scripts/delete-f-keys.sh* (though this appears to be unused)
- *ansible/playbooks/mongo.yml*
- *ansible/playbooks/logs.yml*

It is recommended never to pass secrets or passwords through the command line, but rather through environment variables, read-protected files, or the standard input stream. Reading these areas requires at least the same privileges as the executing user, thereby mitigating this vulnerability.

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

### FWD-02-005 WP2: Potentially-weak API secrets auth guard on self-hosted *(Medium)*

***Fix note**: This issue was fixed by Forward Email by unifying how API secrets are processed. Cure53 verified the fix, confirming that the problem no longer exists.*

It was found that the application contains some auth guards that rely on the *API_SECRETS* variable being an array, while other parts of the codebase allow for this to be a string. In the latter case, the auth guards were found to be ineffective - allowing access to restricted areas. At the same time, it was discovered that the default value for the *API_SECRETS* variable does not appear to be overwritten by the deployment scripts for self-hosted environments, which leaves the application with weak default credentials.

Because the production environment has *API_SECRETS* set in such a way that it is fully unaffected, this vulnerability was only rated as being of *Medium* severity.

**Affected file - default value of *API_SECRETS*:**
*.env.defaults*

**Affected code - default value of *API_SECRETS*:**
```
API_SECRETS=secret,
```

The following source code shows the authentication guard that is in place for the API. It can be seen that the guard uses a *for of* loop to iterate over the *API_SECRETS* property value of the *env* constant, and compares each iterated value with the supplied user credentials of the HTTP request. This only works as long as the *API_SECRETS* property value is an array where each value holds a secret string value. However, when the comma is omitted from the respective entry of the loaded *.env* file, the value stored under *env.API_SECRETS* will be a string. As a result, the *for of* loop iterates over each character of the defined string secret, allowing an attacker to authenticate with a single character of the string only.

**Affected file - API auth guard:**
*app/controllers/api/v1/restricted.js*

**Affected code - API auth guard:**
```
async function restricted(ctx, next) {
  const credentials = auth(ctx.req);
  [...]
  let isValid = false;
  const providedBuf = Buffer.from(credentials.name);

  for (const secret of env.API_SECRETS) {
    const secretBuf = Buffer.from(secret);
    if (crypto.timingSafeEqual(providedBuf, secretBuf)) {
      isValid = true;
```

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

```
    }
  }
```

The second affected location (shown below) is contained within the authentication guard of the SQLite server. Although the underlying root cause is the same - a type confusion on *env.API_SECRETS* - the code pattern looks different this time, making use of the *includes* function on the confused receiver object.

Remarkably, this method is defined for both strings and arrays in modern JavaScript. The operations differ; for arrays the method will return true if the argument is found among its elements, but for strings, it will return true if the argument is a substring of its contents. Therefore, the latter case will allow attackers to authenticate with a single character, in a similar manner to the case shown above.

It should be noted that the decryption does not pose an obstacle here, due to the presence of the Padding Oracle described in FWD-02-012.

**Affected file - SQLite server auth guard:**
*sqlite-server.js*

**Affected code - SQLite server auth guard:**

```
function authenticate(request, socket, head, fn) {
  if (!env.API_SECRETS.includes(decrypt(credentials.name)))
  return fn(Boom.unauthorized( [...] ));
```

It is recommended to integrate a type check for both authentication guards, and to potentially wrap the single string secret in an array before validating it. By doing so, the authentication guards remain effective if only a single value has been defined for the *API_SECRETS* environment file loaded from the *.env* file.

At the same time, for self-hosted environments, it is advised that the *API_SECRETS* environment variable should be overwritten to a value derived from a cryptographically secure random number generator (CSRNG) with sufficient entropy to prevent the default value from being used in production.

### FWD-02-006 WP1: Unauthed *otp_remember_me* cookie allows 2FA bypass *(Medium)*

***Fix note**: This issue was fixed by Forward Email by encrypting and authenticating the otp_remember_me cookie. Cure53 verified the fix, confirming that the problem no longer exists.*

It was confirmed that the presence of the *otp_remember_me* cookie set to the currently authenticated user ID causes a pre-route hook to upgrade the session to a fully authenticated second factor session.

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

Since the user ID is served using inline client-side JavaScript, and the cookie is not required to be authenticated with a signature, an attacker can simply add this cookie to bypass the second factor.

In the following source code, it can be seen that the *get* method on the *ctx.cookies* receiver is invoked for the *otp_remember_me* string literal. If the received cookie value matches the currently authenticated user ID, then the *otp* property of the session object is set to a truthy value that represents a second factor authenticated session.

Although the cookie is paired with an extra signature cookie when it is emitted, the signature must not match, as no options object with a *signed* property is passed to the invocation when retrieving the cookie.

**Affected file - *otp_remember_me* cookie allows 2FA bypass:**
*config/web.js*

**Affected code - *otp_remember_me cookie* allows 2FA bypass:**

```
if (ctx.state.user[config.passport.fields.otpEnabled]) {
  if (ctx.session.otp) {
    [...]
  } else if (ctx.cookies.get('otp_remember_me') === ctx.state.user.id) {
    ctx.session.otp = 'remember_me';
  }
}
```

It is recommended to cryptographically bind the *remember_me* cookie (that is, its name, value, and expiry) to a secret that prevents attackers from forging any of those properties. An alternative approach could be to integrate the OTP remember me state into the already-existing session system.

### FWD-02-008 WP1: 2FA bypass via WebAuthn provider on *callbackRedirect* (Medium)

***Fix note**: This issue was fixed by Forward Email by removing the exception and adding a check for the WebAuthn provider. Cure53 verified the fix, confirming that the problem no longer exists.*

It was found that when the *callbackRedirect* handler runs for the WebAuthn provider, it simply elevates the session to a second factor session, without actually authenticating the factor by populating the *otp* property of the session object to a truthy value - similar to FWD-02-006. It was confirmed that this works as a 2FA OTP bypass in the production environment.

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

The following snippet shows the source code of the *callbackRedirect* handler that is gated by the *callbackCheck* middleware guard which usually authenticates the factor. However, it can be seen that the *callbackCheck* middleware has an exception and becomes a no-op if the provider is *webauthn* and the request method is *GET*. A similar exception exists at the top of the *callbackRedirect* method, which causes the session to be elevated.

**Affected file:**
*forwardemail.net/routes/web/auth.js*

**Affected code:**

```
function callbackCheck(ctx, next) {
  return (ctx.method === 'POST' ||
    (ctx.method === 'GET' && ctx.params.provider !== 'webauthn')) &&
    ctx.passport &&
    ctx.passport.authenticate
    ? ctx.passport.authenticate(ctx.params.provider, {[...]})(ctx, next)
    : next();
}
async function callbackRedirect(ctx, next) {
  [...]
  if (
    ctx.params.provider === 'webauthn' &&
    ctx.isAuthenticated() &&
    ctx.state.user[config.passport.fields.otpEnabled]
  ) {
    ctx.session.otp = 'passkey';
    await ctx.saveSession();
  }
```

**Steps to reproduce:**

1. Log in with a stolen or phished password. The user is stopped at the OTP prompt, but *ctx.login()* has already run.
2. Send a HTTP request to the endpoint *GET /auth/webauthn/ok*. Here, *callbackCheck* skips auth and *callbackRedirect* sets *session.otp*.
3. *ensureOtp* now passes, and the session is elevated to a second factor.

It is recommended to remove these exceptions and only to touch the *otp* session property directly after properly authenticating the web authentication credentials for the currently authenticated user. Doing so ensures that attackers will no longer be able to simply invoke the endpoint in order to bypass the second factor authentication.

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

### FWD-02-009 WP1: SQL injection into local SQLite db via *json-sql-enhanced* (*High*)

***Fix note**: This issue was fixed by Forward Email by adding an allow list for sortable keys. Cure53 verified the fix, confirming that the problem no longer exists.*

An SQL injection (SQLi) vulnerability was identified that can be tied to the *json-sql-enhanced* library used to translate MongoDB-like API calls into SQL queries. It was confirmed that this allows direct access to the underlying database of the affected alias. This could be abused to run complex query conditions that aim for a denial of the SQLite server that binds individual components.

The following HTTP request was sent to the locally-deployed API instance - authenticated with an alias that has created a calendar via the */v1/calendars* endpoints. The SQL payload is located within the *sort* parameter:

**HTTP request - sort SQLi:**

```
GET /v1/calendar-events?sort=
%28SELECT+subject+FROM+Messages+LIMIT+1+XASDSADAD%29 HTTP/1.1
Host: forwardemail.local:4000
Authorization: Basic
YWJhYmFAdmljdGltLmNvbTo2OWY4YTk1ZTU4ZTZiNGNiYTVjNGY1ZjA=
User-Agent: curl/8.5.0
Accept: */*
Connection: keep-alive
```

**Local error log:**

```
\u2716  error     An internal server error has occurred, please try again
later. {
  err: {
    name: 'SqliteError',
    message: 'An internal server error has occurred, please try again
later.',
    stack: 'SqliteError: near "XASDSADAD": syntax error\n' +
      '    at Database.prepare ([...]/node_modules/.pnpm/better-sqlite3-
multiple-ciphers@11.5.0/node_modules/better-sqlite3-multiple-ciphers/lib/
methods/wrappers.js:5:21)\n' +
      '    at SQLite.parsePayload
(/home/vmuser/forwardemail.net/helpers/parse-payload.js:2342:25)\n' +
```

The root cause of this SQLi vulnerability is that the internally used *wrapIdentifier* sanitizer function of the *json-sql-enhanced* library insufficiently sanitizes the identifier if the input argument contains parentheses or certain SQL keywords. At the same time, the Forward Email API does not validate the *sort* parameter before passing it to the library. As a result, the parameter directly allows injection into it.

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

**Affected file - *wrapIdentifier*:**
*node_modules/.pnpm/json-sql-enhanced@3.0.0/node_modules/json-sql-enhanced/lib/dialects/base/index.js*

**Affected code - *wrapIdentifier*:**
```
Dialect.prototype.wrapIdentifier = function (name) {
  [...]
  // Don't wrap SQL functions (anything with parentheses)
  if (name.includes('(') && name.includes(')')) {
    return name;
  }

  // Don't wrap SQL expressions (CASE, SELECT, etc.)
  const upperName = name.toUpperCase().trim();
  if (
    upperName.startsWith('CASE ') || [...] || upperName.includes(' END')
  ) {
    return name;
  }

  return '"' + name + '"';
};
```

Another occurrence (which is not reachable) could occur through the *$elemMatch* condition - which directly uses the keys of the underlying object and embeds them unsanitized into the underlying SQL query:

**Affected file - *$elemMatch* condition:**
*node_modules/.pnpm/json-sql-enhanced@3.0.0/node_modules/json-sql-enhanced/lib/dialects/base/operators/mongodb.js*

**Affected code - *$elemMatch* condition:**
```
function buildElementMatchCondition(dialect, field, conditions) {
  const fieldName = dialect.wrapIdentifier(field);

  if (typeof conditions === 'object' && conditions !== null) {
    const subConditions = [];
    for (const key in conditions) {
      [...]
      const value = conditions[key];
      if (typeof value === 'object' && value !== null) {
        [...]
      } else {
        const value_ = dialect.builder._pushValue(value);
        subConditions.push(`json_extract(value, '$.${key}') = ${value_}`);
      }
```

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

It is recommended to validate that the value passed as the *sort* parameter is actually a valid, sortable column name for the table. By doing so, the first injection can be prevented. The second injection is more subtle to fix - although it is currently not reachable. It is advised that remediation of this issue will require the consistent and deep validation of inputs and object keys, before using them to construct filter queries passed to the library.

## FWD-02-010 WP1: Pre-auth persistent XSS via DMARC report email *(**Critical**)*

***Fix note**: This issue was fixed by Forward Email by preventing the XSS during rendering and adding additional pattern matching to reject malicious DMARC reports. Cure53 verified the fix, confirming that the problem no longer exists.*

It was confirmed that the web user interface contains a Cross-Site Scripting (XSS) vulnerability when rendering the organization name of a received but unauthenticated DMARC report email into the HTML markup. The deployed Content Security Policy (CSP) does not stop this, as the *script-src* directive is deployed with the *strict-dynamic* keyword and the payload is added dynamically from trusted JavaScript invocation via jQuery's *html(...)* method.

This attack does not require any authentication, and only requires a passive interaction from the authenticated admin user (a click on the DMARC reports button in the UI). For this reason, this finding is rated as being of *Critical* severity.

**Affected file:**
*assets/js/dmarc-reports.js*

**Affected code:**

```
function renderReportsTable(reports) {
  const $table = $('#reports-table');
  [...]
  html += `
     <tr>
       <td class="text-nowrap">${receivedDate}</td>
       <td>${report.domain_name}</td>
       <td>${report.org_name}</td>[...]`;
  [...]
  $table.html(html);
```

**Steps to reproduce:**

1. The attacker retrieves the DMARC aggregate report email address from the DNS TXT records of the paid domain, which is of the form *dmarc-<domain-ID>@forwardemail.net*.
2. The attacker sends an email to that address containing the adapted XML seen below that contains the highlighted XML-encoded XSS payload.

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

3. The payload is decoded by the XML parser, inserted in the database, and added dynamically to the DOM by the client-side JavaScript once the domain admin clicks on DMARC reports for this domain.

**Malicious DMARC report XML attached to email:**

```
<?xml version="1.0" encoding="UTF-8"?>
<feedback><report_metadata>
<org_name>google.com&lt;script&gt;window.__XSS_POC=1;document.title=&quot;XSS-POC-OK&quot;&lt;/script&gt;</org_name>
<email>noreply-dmarc-support@google.com</
email><extra_contact_info>https://support.google.com/a/answer/2466580</
extra_contact_info><report_id>poc-1786473090</
report_id><date_range><begin>1786386690</begin><end>1786473090</end></
date_range></
report_metadata><policy_published><domain>cure53lazerpenguin.com</
domain><adkim>r</adkim><aspf>r</aspf><p>reject</p><sp>reject</
sp><pct>100</pct></policy_published><record><row><source_ip>203.0.113.10</
source_ip><count>42</count><policy_evaluated><disposition>none</
disposition><dkim>pass</dkim><spf>pass</spf></policy_evaluated></
row><identifiers><header_from>cure53lazerpenguin.com</header_from></
identifiers><auth_results><dkim><domain>cure53lazerpenguin.com</
domain><result>pass</result><selector>default</selector></
dkim><spf><domain>cure53lazerpenguin.com</domain><result>pass</result></
spf></auth_results></record></feedback>
```

It is recommended that this XSS vulnerability should be fixed by utilizing a template rendering mechanism that applies proper input escaping. This would mean that attackers would no longer be able to perform this type of attack.

## FWD-02-013 WP1: HTML injection through unsubscribe token *(Low)*

***Fix note**: This issue was fixed by Forward Email by HTML-encoding the user-controlled values before rendering. Cure53 verified the fix, confirming that the problem no longer exists.*

It was found that the application suffers from an HTML injection vulnerability involving a specially-crafted unsubscribe token that can be crafted by leveraging the Padding Oracle attack described in FWD-02-012. The application receives an encrypted token from the request path and uses the decrypt function to obtain a plaintext JSON object. The template attribute of this JSON object is then embedded unsanitized into the HTML markup in a Pug template. Fortunately, it was not possible to leverage this HTML injection into an XSS vulnerability, due to the presence of the CSP in production environments.

**Affected file - vulnerable Pug template:**
*app/views/unsubscribe.pug*

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

**Affected code - vulnerable Pug template:**

```
p.text-muted.mb-4!= t('You have been unsubscribed from <strong
class="notranslate">%s</strong> emails.',
humanize(titleize(unsubscribeTemplate)))
```

It is recommended to remove the HTML from the translated value and to re-enable encoding of HTML relevant characters in Pug, by not using the unsafe *!=* operator to render the value. Doing this will ensure that attackers can no longer inject HTML markup, because this will be encoded by the Pug template engine.

## FWD-02-014 WP1: Unauthenticated DoS through forward regex *(High)*

***Fix note**: This issue was fixed by Forward Email by limiting the recursion depth and generated destination address length. Cure53 verified the fix, confirming that the problem no longer exists.*

Analysis of the forwarding logic revealed an unauthenticated Denial-of-Service (DoS) vector by abusing the regular expression substitution feature. A single email sent to the MX service will then suffice to crash the service. This step does not require any authentication, and will therefore enable trivially-repeatable DoS against the service, resulting in a service disruption for all users.

**Steps to reproduce:**

1. Register a free account and set up a custom domain such as *evil.com*
2. Add the malicious DNS TXT record for forwarding forward-email=*/^(.*)$/: $1$1@evil.com*
3. Connect to the MX service (*mx1.forwardemail.net* and *mx2.forwardemail.net* on the production system) and deliver an email to *a@evil.com*

The above steps will trigger an infinite recursion through the substitution pattern in the forwarding expression, which will cause the logic to build an infinite list of forwarding addresses, resulting in a crash of the NodeJS process.

The code snippet below shows an excerpt of the *getForwardingAddresses()* function. The flaw is based on the fact that the user-supplied forwarding expression from the DNS TXT record is used to construct a list of *forwardingAddresses* once the regular expression matches a received email. Since */^(.*)$/* matches any user part of an address, this is executed for all inbound addresses of that domain.

The substitution pattern of *$1$1@evil.com* results in the matched user being duplicated. For example, *a@evil.com* results in *aa@evil.com*. Due to the unbounded recursive call of *getForwardingAddresses*, this substitution is continued endlessly, with each recursion re-executing the substitution pattern and resulting in a new forwarding address. This is also not limited by the call to *isEmail()* before calling *getForwardingAddresses()* recursively, as this function does not enforce an upper length limit on the email addresses.

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

**Affected file:**
*forwardemail.net/helpers/get-forwarding-addresses.js*

**Affected code:**

```
async function getForwardingAddresses(
  address,
  recursive = [],
  ignoreBilling = false,
  session
) {
  [...]
    //
    // regular expression support
    //
<https://github.com/forwardemail/free-email-forwarding/pull/245/commits/
       e04ea02d700b51771bf61ed512d1763bbf80784b>
    // (with added support for regex gi flags)
    //
    if (startsWithSlash && hasTwoSlashes) {
      [...]
      if (username && regex && regex.test(username.toLowerCase()) {
        [...]
        const hasDollarInterpolation =
             REGEX_INTERPOLATED_DOLLAR.test(target);

        const substitutedAlias = hasDollarInterpolation
          ? username.toLowerCase().replace(regex, target)
          : target;
        [...]
        if (isURL(substitutedAlias, config.isURLOptions))
          forwardingAddresses.push(substitutedAlias);
        else forwardingAddresses.push(substitutedAlias.toLowerCase());
      }
    } else if (
      [...]
  //
  // ensure forwarding addresses are unique to prevent additional hops
  // TODO: we could also do indexOf or includes check above before pushing
  //
  forwardingAddresses = _.uniq(forwardingAddresses);

  // TODO: isn't actually utilized since `recursive` arg not used
  //       (e.g. we don't do MX lookup on the TXT we're forwarding to)
  // allow one recursive lookup on forwarding addresses
  const recursivelyForwardedAddresses = [];

  const { length } = forwardingAddresses;
```

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

```
  for (let x = 0; x < length; x++) {
    const forwardingAddress = forwardingAddresses[x];
    try {
      // TODO: is the culprit
      if (recursive.includes(forwardingAddress)) continue;
      if (isURL(forwardingAddress, config.isURLOptions)) continue;

      const newRecursive = [...forwardingAddresses, ...recursive];

      // prevent a double-lookup if user is using + symbols
      if (isEmail(address) && forwardingAddress.includes('+'))
        newRecursive.push(
          `${parseUsername(address)}@$
             {parseHostFromDomainOrAddress(address)}`
        );

      // support recursive IMAP lookup

      const data = await getForwardingAddresses.call(
        this,
        forwardingAddress,
        newRecursive,
        ignoreBilling,
        session
      );
[...]
```

To mitigate this issue, Cure53 recommends adding a bound to the recursion, and limiting it to a single recursive call. Furthermore, it is advised that a length limit should be added for the substitution, so that it can never grow infinitely.

## FWD-02-017 WP1: SSRF to private IPs via IPv4-mapped IPv6 literals *(Medium)*

***Fix note**: This issue was fixed by Forward Email by utilizing ipaddr.js to parse IP addresses and reject private address ranges. Cure53 verified the fix, confirming that the problem no longer exists.*

Analysis of the webhook logic revealed that the logic to detect and prevent connections to private, internal IP addresses is flawed. A specially-crafted IPv6 address can bypass the string-matching logic in *forwardemail.net/helpers/regex-localhost.js* and enable attackers to connect to arbitrary IPv4 addresses responding to HTTP requests. This includes endpoints such as the metadata service common to standard cloud environments.

While standard IP addresses such as *127.0.0.1*, *::1* and IPv4-mapped IPv6 literals such as *::ffff:169.254.169.254* are properly rejected, it was found that various other notations - such as IPv4-mapped IPv6 literals in hexadecimal form - are not. Specifically, the following patterns appear to not be filtered:

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

- IPv4-mapped IPv6 literals in hexadecimal form such as *::ffff:7f00:1* for *127.0.0.1* or *::ffff:c0a8:101* for *192.168.1.1*
- Unspecified IPv6 address *::*
- IPv6 addresses using leading zeros - including IPv4-mapped literals such as *0:0:0:0:0:ffff:127.0.0.1*

As this filter logic is used throughout the codebase for various outbound connections of different services (webhooks, push notifications, domain categorization, S3 access checks, etc.), this gives a wide range of potential attack vectors. While testing primarily focused on the webhook logic, all of these locations can be used to initiate malicious connections against internal services.

Using webhooks, attackers can inspect parts of the response through the logs viewable in the web frontend, and will also be shown parts of the response when initiating a mail delivery via the SMTP protocol against the MX service if a mail alias triggers a webhook.

**Affected file:**
*forwardemail.net/helpers/is-private-host.js*

**Affected code:**

```
const REGEX_LOCALHOST = require('#helpers/regex-localhost');
[...]
function isPrivateHost(hostname) {
  if (!hostname) return true;

  // Strip brackets from IPv6
  const host = hostname.replace(/^\[|]$/g, '');

  // Check shared REGEX_LOCALHOST (all private/reserved IP ranges)
  if (REGEX_LOCALHOST.test(host)) return true;

  // Check if TLD (or bare hostname) is a reserved/test domain
  // Covers: localhost, local, internal, test, metadata, instance-data,
etc.
  const parts = host.toLowerCase().split('.');
  const tld = parts[parts.length - 1];
  if (config.testDomains.includes(tld)) return true;

  // Check full hostname against testDomains (e.g. "metadata", "instance-
data")
  if (config.testDomains.includes(parts[0])) return true;

  return false;
}
```

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

**Affected file:**
*forwardemail.net/helpers/regex-localhost.js*

**Affected code:**

```
const IP_RANGES = [
  // 0.0.0.0/8 - "This host on this network" (RFC 1122)
  /^(:{2}f{4}: )?0(?:\.\d{1,3}){3}$/,
  // 10.0.0.0 - 10.255.255.255
  /^(:{2}f{4}:)?10(?:\.\d{1,3}){3}$/,
  // 100.64.0.0/10 - CGNAT (RFC 6598)
  /^(:{2}f{4}:)?100\.(6[4-9]|[7-9]\d|1[01]\d|12[0-7])(?:\.\d{1,3}){2}$/,
  // 127.0.0.0 - 127.255.255.255
  /^(:{2}f{4}:)?127(?:\.\d{1,3}){3}$/,
  // 169.254.0.0/16 - Link-Local (RFC 3927) - full range
  /^(:{2}f{4}:)?169\.254(?:\.\d{1,3}){2}$/,
  // 172.16.0.0 - 172.31.255.255
  /^(:{2}f{4}:)?(172\.1[6-9]|172\.2\d|172\.3[01])(?:\.\d{1,3}){2}$/,
  // 192.168.0.0 - 192.168.255.255
  /^(:{2}f{4}:)?192\.168(?:\.\d{1,3}){2}$/,
  // 198.18.0.0/15 - Benchmarking (RFC 2544)
  /^(:{2}f{4}:)?198\.1[89](?:\.\d{1,3}){2}$/,
  // ::1 - IPv6 loopback
  /^:{2}1$/,
  // fc00::/7
  /^f[cd][\da-f]{2}(::1$|:[\da-f]{1,4}){1,7}$/,
  // fe80::/10
  /^fe[89ab][\da-f](::1$|:[\da-f]{1,4}){1,7}$/,
  // 192.0.0.0/24 - IETF Protocol Assignments (RFC 6890)
  /^(:{2}f{4}:)?192\.0\.0\.\d{1,3}$/,
  // 192.0.2.0/24 - Documentation TEST-NET-1 (RFC 5737)
  /^(:{2}f{4}:)?192\.0\.2\.\d{1,3}$/,
  // 198.51.100.0/24 - Documentation TEST-NET-2 (RFC 5737)
  /^(:{2}f{4}:)?198\.51\.100\.\d{1,3}$/,
  // 203.0.113.0/24 - Documentation TEST-NET-3 (RFC 5737)
  /^(:{2}f{4}:)?203\.0\.113\.\d{1,3}$/,
  // 224.0.0.0/4 - Multicast (RFC 5771)
  /^(:{2}f{4}:)?2(2[4-9]|3\d)(?:\.\d{1,3}){3}$/,
  // 240.0.0.0/4 - Reserved for future use (RFC 1112)
  /^(:{2}f{4}:)?2(4\d|5[0-5])(?:\.\d{1,3}){3}$/,
  // 255.255.255.255/32 - Limited Broadcast
  /^(:{2}f{4}:)?255\.255\.255\.255$/
];
```

Another noteworthy issue is the fact that the pattern for *0.0.0.0/8* contains a stray space character, and will therefore never get matched. Furthermore, the subnet *192.88.99.0/24*, previously used for 6to4 relay anycasts - which is a reserved IP range - is allowed and not prohibited.

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

To mitigate this vulnerability, Cure53 recommends parsing IP addresses into canonicalized form (e.g. using the package *ipaddr.js*[2]), and to filter on the parsed form using the *range()* function.

## FWD-02-018 WP1: DMARC bypass via duplicate *From* header *(High)*

***Fix note**: This issue was fixed by Forward Email by rejecting emails with multiple From headers. Cure53 verified the fix, confirming that the problem no longer exists.*

Analysis of the DMARC policy processing showed that it can be bypassed by a malicious email containing more than one *From* header. This results in the malicious email being delivered to the recipient's inbox, even in cases where the DMARC policy is configured to reject emails that fail SPF and DKIM verification. This could be abused by spammers to fake the sender of an email, to deliver spam emails to a user without that email being marked as spam or rejected.

The flaw is rooted in the way the *From* header is processed by the MX service; the *mailauth* package (version 4.12.0) which performs the SPF, DKIM, and DMARC policy processing will extract all *From* headers (in *DkimVerifier*), and, if there are more, the *verifyDmarc* function will return *false*.

This is shown in the code snippet below:

**Affected file:**
*https://github.com/postalsys/mailauth/blob/1a6514aa3d2545e2f64c5d6874e7439fe731084e/lib/dmarc/verify.js#L14-L21*

**Affected code:**
```
const verifyDmarc = async opts => {
    let { headerFrom, spfDomains, dkimDomains, resolver, arcResult } =
opts;

    resolver = resolver || dns.resolve;

    if (Array.isArray(headerFrom)) {
        if (headerFrom.length === 1) {
            headerFrom = headerFrom[0];
        } else {
            // invalid number of FROM addresses found
            return false;
        }
    }
```

[2] https://www.npmjs.com/package/ipaddr.js

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

As a result, the MX service's *session.dmarc* will be set to *false*. It is important to note that this is a different result than a missing DMARC policy - which would return an object with *session.dmarc.status.result* set to '*none*'. Since the case of *session.dmarc == false* is never handled in the MX service, the service defaults to accepting these emails and delivering them to the user's mailbox without any spam markers or similar limits.

**Affected file:**
*forwardemail.net/helpers/is-authenticated-message.js*

**Affected code:**

```
async function isAuthenticatedMessage(headers, body, session, resolver) {
  [...]
  const [results, spfFromHeader] = await Promise.all([
    authenticate(Buffer.concat([headers.build(), body]), {
      ...options,
      sender: session.envelope.mailFrom.address,
      seal: config.signatureData
    }),
    spf({
      ...options,
      sender: session.originalFromAddress
    })
  ]);
  [...]
  session.arc = results.arc;
  session.dmarc = results.dmarc;
  session.bimi = results.bimi;
  session.receivedChain = results.receivedChain;
  [...]
  if (
    session.dmarc &&
    session.dmarc.status &&
    session.dmarc.status.result === 'fail' &&
    session.dmarc.policy === 'reject' &&
    !isLegitDSN &&
    !isTruthSource
  ) {
    throw new SMTPError(
      "The email sent has failed DMARC validation and is rejected due to
the domain's DMARC policy",
      {
        // if spf status was temperror then retry
        responseCode: session.spf.status.result === 'temperror' ? 421 : 550
      }
    );
  }
```

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

```
  // DMARC p=quarantine rejection (non-allowlisted senders only)
  if (
    session.dmarc &&
    session.dmarc.status &&
    session.dmarc.status.result === 'fail' &&
    session.dmarc.policy === 'quarantine' &&
    !session.isAllowlisted &&
    !session.hadAlignedAndPassingDKIM &&
    session.spf.status.result !== 'pass' &&
    !isLegitDSN &&
    !isTruthSource
  ) {
    [...]
    throw new SMTPError(
      "The email sent has failed DMARC validation and is rejected due to
the domain's DMARC policy",
      {
        responseCode: session.spf.status.result === 'temperror' ? 421 : 550
      }
    );
  }

  // if no DMARC and SPF had hardfail and no aligned DKIM then reject
  // NOTE: it'd be nice if we alerted admins of SPF permerror due to SPF
misconfiguration
  if (
    session.spf.status.result === 'fail' &&
    session.dmarc &&
    session.dmarc.status &&
    session.dmarc.status.result === 'none' &&
    [...]
  )
    throw new SMTPError(
      "The email sent has failed SPF validation and is rejected due to the
domain's SPF hard fail policy (ensure that your messages have a DKIM
aligned signature with your From header)"
    );
}
```

Any further anti-spoofing and protection features in *forwardemail.net* are based on the first *From* header. *isArbitrary* (*forwardemail.net/helpers/is-arbitrary.js*) will use the *getHeaders* helper, which will yield only the first header and ignore any subsequent headers. Due to this, the administration panel will not show the fact that an email contained multiple *From* headers, thereby hiding any attacks of this nature.

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

Further, the header keys constructed by *getHeaders* are constructed from the non-normalized form. Therefore, *From* is a different key than *from* or *FROM*. While this does not have a direct impact here, as the *specificKey* match is performed on the lower-case variant, it can become a problem in situations where *getHeaders* is used with *specificKey* set to *null*.

**Affected file:**
*forwardemail.net/helpers/get-headers.js*

**Affected code:**
```
const SINGLE_KEYS = new Set([
  [...]
  'from',
  [...]
]);
function getHeaders(headers, specificKey = null) {
  const _headers = {};
  [...]
  for (const { line } of lines) {
    const index = line.indexOf(':');
    const key = line.slice(0, index);
    const lc = key.toLowerCase();

    // only keep the first value for certain keys
    //
<https://github.com/nodemailer/mailparser/blob/ac11f78429cf13da42162e996a05
b875030ae1c1/lib/mail-parser.js#L395-L413>
    if (_headers[key] && SINGLE_KEYS.has(lc)) continue;

    _headers[key] = line
      .slice(index + 1)
      .trim()
      .replace(/\s*\r?\n\s*/g, ' ');
```

A specific spam attempt would then include the spoofed sender in the first *From* header and another one in the second *From* header, as shown below. As common mail clients usually only display the first *From* header, this will result in the forged header (such as *security@paypal.com*) being shown.

**PoC email with multiple *From* headers:**
```
Subject: Important Account Update
From: security@paypal.com
From: admin@asfasdfasdfasdf.com
To: test@victim.com

This is a PoC showing how to spoof the sender on forwardemail.net despite
proper DMARC, DKIM and SPF configuration.
```

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

To mitigate this vulnerability, Cure53 recommends directly rejecting emails with multiple *From* headers (including case-insensitive matching), instead of silently ignoring additional headers. Additionally, it is advised that a *false* DMARC verification result should be treated as a failure condition, and should not result in the email being delivered to the user's inbox.

## FWD-02-020 WP1: Attachment injection through weak hash function *(Medium)*

***Fix note**: This issue was fixed by Forward Email by utilizing SHA-256 and falling back to full comparison in case of an existing hash value. Cure53 verified the fix, confirming that the problem no longer exists.*

The *AttachmentStorage* type manages email attachments and deduplicates attachments based on a hash over the Base64-encoded MIME part for the attachment. This logic uses the *rev-hash* library, which internally uses the first ten hex characters (40 bits) of the content's MD5 hash. A malicious actor could abuse this to inject different attachments into a legitimate email holding an attachment before the legitimate email is delivered to the user's mailbox.

A realistic, but not always trivial attack vector for this vulnerability involves newsletters, which are commonly delivered in batches. An attacker can therefore receive the legitimate attachment before the user receives it and has often enough time (minutes to hours) to forge a malicious attachment with different contents, but identical 40-bit truncated MD5 hash sum. The attacker can then send this via an email to the user. If the malicious attachment arrives before the legitimate one, then only the malicious attachment will be stored, and the legitimate email will return the malicious attachment once the user opens its attachment. On recent hardware, finding such an MD5 collision takes between a few hours or seconds, which makes it likely to find a collision before the legitimate email arrives in the user's mailbox.

**Affected file:**
*forwardemail.net/helpers/attachment-storage.js*

**Affected code:**
```
async function createAttachment(instance, session, node) {
  // const hex = await this.calculateHashPromise(node.body);
  // node.hash = revHash(Buffer.from(hex, 'hex'));
  node.hash = revHash(node.body);
  node.counter = 1;
  node.counterUpdated = new Date();
  node.size = node.body.length;
  if (Number.isNaN(node.magic) || typeof node.magic !== 'number') {
    const err = new TypeError('Invalid magic');
    err.node = node;
    throw err;
```

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

```
}

const sql = builder.build({
  type: 'update',
  table: 'Attachments',
  condition: {
    hash: node.hash
  },
  modifier: {
    $inc: {
      counter: 1,
      magic: node.magic
    },
    $set: {
      counterUpdated: new Date().toISOString()
    }
  },
  returning: ['*']
});
```

To mitigate this vulnerability, Cure53 recommends using a secure hash function such as SHA-256, and in the rare case of a collision, also to compare the full body before treating the new attachment as a duplicate of the existing one.

## FWD-02-021 WP1: Sieve security validation bypass through UI update *(Low)*

***Fix note**: This issue was fixed by Forward Email by checking the security validator result. Cure53 verified the fix, confirming that the problem no longer exists.*

It was verified that the Sieve update logic does not run the security validation on the updated script, and therefore enables the bypass of any security validations in place when the script is created via the UI or the API.

The core of the flaw resides in the fact that the UI sieve update logic in the controller does not check the *securityResult.valid* property, and the Mongoose *save* hook will not be triggered due to the use of the *findByIdAndUpdate* function, as shown below:

**Affected file:**
*app/controllers/web/my-account/sieve.js*

**Affected code:**

```
async function updateSieveScript(ctx) {
[...]
    // Security analysis
    const securityResult = securityValidator.validate(content);

    updates.content = content;
```

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

```
    updates.required_capabilities = securityResult.requiredExtensions ||
[];
    updates.security_warnings = securityResult.warnings || [];
  }

  // Handle activation
  if (
    typeof is_active === 'boolean' ||
    is_active === 'true' ||
    is_active === 'false'
  ) {
    const shouldActivate = is_active === true || is_active === 'true';
    if (shouldActivate) {
      // Deactivate other scripts
      await SieveScripts.updateMany(
        {
          alias: ctx.state.alias._id,
          _id: { $ne: ctx.state.sieveScript._id },
          is_active: true
        },
        { $set: { is_active: false } }
      );
    }

    updates.is_active = shouldActivate;
  }

  const script = await SieveScripts.findByIdAndUpdate(
    ctx.state.sieveScript._id,
    { $set: updates },
    { new: true }
  )
    .lean()
    .exec();
```

The impact of this vulnerability is similar to the finding *FWD-01-004* from the previous report (see *FWD-01*), where an authenticated mailbox owner can inject headers and modify protected headers to hide spam, signature, and policy results downstream. Further, other protections such as the maximum redirect count and the alias *vacation_responder* conflict check can be bypassed through this flaw.

To mitigate this issue, Cure53 recommends also running the *SieveSecurityValidator* on the update path for the Web UI. This could be done by replacing the *findByIdAndUpdate* Mongoose call with respective calls to load the document and the call *save*, which will trigger the *save* hook to call the security validator. Alternatively, the validation call could be performed in the controller logic itself.

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

## FWD-02-022 WP1: SSRF via non-default SMTP forward port *(Medium)*

***Fix note**: This issue was fixed by Forward Email by rejecting connections to private IP ranges. Cure53 verified the fix, confirming that the problem no longer exists.*

Testing of the custom SMTP forward port feature via DNS TXT record *forward-email-port=* revealed that this enables connections to internal hosts via the SMTP protocol. This enables the testing of internal IP addresses and ports if they respond to SMTP. As this feature is also accessible for free accounts by configuring the two respective DNS TXT records (e.g. *forward-email-port=2525* and *forward-email=example.com*), this attack only requires a valid account with a custom domain. By configuring the DNS for *example.com* - in this case to an internal IP and delivering a mail to the open MX service - the SMTP connection to the internal host will be initiated. As the MX service will return a different response depending on the success and failure of the connection, this indicates to the attacker if the probed port and IP were reachable.

It should be noted that for standard SMTP configurations with Port 25, this is no problem, as in this case, the *mx-connect* package will actually perform the necessary IP check and disallow connections to internal IP addresses. Non-standard port configurations, however, do not call into *mx-connect*, and therefore enable unrestricted IP ranges.

**Affected file:**
*forwardemail.net/helpers/get-transporter.js*

**Affected code:**

```
async function getTransporter(options = {}, err) {
  const {
    ignoreMXHosts,
    mxLastError,
    target,
    port,
    resolver,
    logger,
    cache,
    // client
    envelope
  } = options;

  // safeguard to ensure port is always a number
  if (typeof port === 'string') throw new TypeError('Port must be a
number');

  let mx = {
    host: target,
    port
```

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

```
  };

  //
  // TODO: rewrite to use SMTPConnection object instead of nodemailer
  //       <https://github.com/nodemailer/nodemailer/issues/1575>
  //

  // this is required since custom port forwarding would be recursive
otherwise
  if (env.NODE_ENV === 'test' || port === 25) {
    // <https://github.com/zone-eu/mx-connect#configuration-options>
    mx = await asyncMxConnect({
      ignoreMXHosts,
```

To mitigate this vulnerability, Cure53 recommends a similar logic as for FWD-02-017, by utilizing the *ipaddr.js* package to determine whether an IP address falls within a restricted IP range or not. It is advised that this should be implemented for all connections in a production environment - including non-standard SMTP ports.

**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.

### FWD-02-007 WP1: Default and weak cookie signature secret *(Low)*

***Fix note**: This issue was fixed by Forward Email by enforcing a secure SESSION_KEYS value. Cure53 verified the fix, confirming that the problem no longer exists.*

It was found that the web application does not override the default known and weak cookie signature secret applied by the *ladjs* library. It was confirmed that this is true for the current production environment. As a result, an attacker could modify the cookie values and forge a valid signature for them. However, since the application currently does not rely on the cookie signatures - which might change during a fix of FWD-02-006 - this issue was only rated with a severity of *Low*.

**Affected file - *ladjs* default cookie secret:**
*node_modules/.pnpm/@ladjs+web@21.1.3_@babel+core@7.28.0_axe@13.0.0_cabin@14.0.0_axe@13.0.0__handlebars@4.7.7_mus_2rckptr4wwrs2d4pqh47hxemwu/node_modules/@ladjs/web/index.js*

**Affected code - *ladjs* default cookie secret:**

```
     sessionKeys: process.env.SESSION_KEYS
        ? process.env.SESSION_KEYS.split(',')
        : ['lad'],
```

The following excerpt from the shell can be used to forge a valid signature for an arbitrary cookie. The script uses an HMAC with the default weak secret. It was confirmed that this secret generates the matching signatures for the production environment:

**Excerpt from shell - forging a signature with the default secret:**

```
const crypto = require('node:crypto');
const sign = (data, key = 'lad') =>
    crypto.createHmac('sha1',
key).update(data).digest('base64').replace(/[/+=]/g, c => ({ '/': '_', '+':
'-', '=': '' }[c]));

sign('otp_remember_me=69f9a3ad17ddb7f441f6bcf5');
// => 'DhIPb4I-juAp5Oz8F3fRZkUZ_D0'
```

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

It is recommended to explicitly overwrite the *SESSION_KEYS* environment variable with a fresh key generated from a CSRNG. As seen from the code above, this will be used to seed the session signature with a strong secret, preventing attackers from using the known and weak secret.

### FWD-02-011 WP1: Outdated and vulnerable dependencies *(Low)*

***Fix note**: This issue has been partially fixed by the development team. The Forward Email team successfully patched all direct production dependencies within their scope, removing 50 advisories and updating 15 core packages. Further dependency updates are currently blocked by a required Node v18 lock due to yet unresolved performance regressions in newer versions[3] and deep transitive parent chains that cannot be safely overridden without breaking application behavior or UI styling.*

During the security assessment, the observation was made that several software packages leveraged outdated versions that are vulnerable to a host of security risks. Notably, the version information provided is based on data collected at the time of testing. Whether these vulnerabilities are exploitable entirely depends on how the relevant functionality is used in the targeted application at present.

**Command:**

```
% pnpm audit --prod
```

The command lists **12 Critical** and **162 High** severity vulnerabilities in total. The critical severity issues are listed below:

| Package | CVSS | CVE | GHSA |
|---|---|---|---|
| *xmldom* | 9.8 | CVE-2022-39353 | GHSA-crh6-fp67-6883 |
| *jsrsasign* | 9.1 | CVE-2021-30246 | GHSA-27fj-mc8w-j9wg |
| *plist* | 9.8 | CVE-2022-22912 | GHSA-4cpg-3vgw-4877 |
| *protobufjs* | 9.8 | CVE-2023-36665 | GHSA-h755-8qp9-cq85 |
| *form-data* | - | CVE-2025-7783 | GHSA-fjxv-7rqg-78g4 |
| *fast-xml-parser (covers 2 vulns)* | 9.3 | CVE-2026-25896 | GHSA-m7jm-9gc2-mpf2 |
| *handlebars* | 9.8 | CVE-2026-33937 | GHSA-2w6w-674q-4c4q |
| *protobufjs* | 9.8 | CVE-2026-41242 | GHSA-xq3m-2v4x-88gg |
| *shell-quote* | 8.1 | CVE-2026-9277 | GHSA-w7jw-789q-3m8p |

[3] https://github.com/nodejs/node/issues/60719

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

| | | | |
|---|---|---|---|
| *tar* | 7.5 | CVE-2026-59873 | GHSA-23hp-3jrh-7fpw |
| *jsrsasign* | 9.1 | CVE-2026-4599 | GHSA-5jx8-q4cp-rhh6 |

Notably, the testing team was unable to comprehensively prove any potential impact during the limited time frame granted for this review. As such, the wider implications remain unknown at this point, and it is advised that they should be subjected to internal research at the earliest possible convenience for the in-house team.

To mitigate the existing issues as effectively as possible, Cure53 recommends upgrading all affected libraries and establishing a policy to ensure that libraries remain up-to-date, moving forward. This will ensure that the premise can benefit from patches rolled-out for previously-detected weaknesses across a variety of different solutions.

## FWD-02-012 WP1: Padding Oracle against legacy CBC decryption *(Medium)*

***Fix note**: This issue was fixed by Forward Email by adding a configuration switch to disable the legacy encryption feature. Cure53 verified the fix, confirming that the problem no longer exists.*

It was discovered that the decrypt function used throughout the application still supports the legacy AES-CBC decryption without any integrity protection. This artifact was previously exploited in *FWD-01*, and still allows attackers to essentially generate a valid ciphertext for a chosen plaintext by frequently querying exposed endpoints that make use of this function and expose valid versus incorrect padding. This becomes a problem when the application trusts that these plaintexts originate from the application - as observed in FWD-02-013.

**Affected file:**
*helpers/encrypt-decrypt.js*

**Affected code:**
```
function decrypt(
  text,
  encryptionKey = env.HELPER_ENCRYPTION_KEY,
  algorithm = 'aes-256-cbc', // Kept for backwards compat
  triedLegacyKey = false // Internal flag to prevent recursion
) {
  if (!text) throw new TypeError('Text value missing');
  [...]
  if (looksLikeLegacy) {
    try {
      return decryptLegacyAES(text, encryptionKey, algorithm);
```

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

It is recommended to remove support for legacy unauthenticated decryption algorithms, through a configuration flag that is set for new installations. By doing so, both production and new self-hosted instances will be safe from this attack, because all remaining decryption algorithms will ship with integrity protection.

## FWD-02-015 WP1: API token leak via WebSocket query string fallback *(Info)*

***Fix note**: This issue was fixed by Forward Email by removing the URL query fallback logic. Cure53 verified the fix, confirming that the problem no longer exists.*

During a static analysis of the API surface, the WebSocket endpoint at */v1/ws* was identified as accepting authentication credentials through URL query parameters as a fallback mechanism for browser-based WebSocket clients that lack the ability to set custom *Authorization* headers on the HTTP upgrade request. This pattern is a known anti-pattern for WebSocket authentication, and introduces multiple information disclosure risks.

Query parameters are inherently unsuitable for secrets. They are captured by virtually every layer of the HTTP stack - they appear in server access logs, CDN logs, load balancer logs, and reverse proxy logs; they are saved in browser history and browser autocomplete data; they are transmitted in the *Referer* header to third-party resources; and they can be leaked via *document.referrer* in JavaScript[4]. An attacker who gains access to any of these locations can recover the API token and associated *alias_id*, allowing them to authenticate to the WebSocket endpoint and receive real-time email, calendar, and contact event notifications for the compromised alias.

**Affected file:**
*forwardemail.net/helpers/api-websocket-handler.js*

**Affected code:**

```
async _authenticate(request) {
// Prefer the standard Authorization header, but fall back to query
// parameters for browser WebSocket clients that cannot set custom
// headers on the upgrade request.
let creds = basicAuth(request);

if (!creds || !creds.name) {
  const { query } = url.parse(request.url, true);
  if (query.username) {
    creds = { name: query.username, pass: query.password || '' };
  } else if (query.token) {
    // API token via query param (password-less; alias_id required)
    creds = { name: query.token, pass: '' };
  }
}
[...]
}
```

[4] https://owasp.org/www-community/vulnerabilities/Information_exposure_through_query_strings_in_url

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

To remediate this issue, Cure53 recommends that the query-string fallback mechanism should be removed entirely. In its place, it is advised that a dedicated authentication handshake should be performed after the WebSocket connection has been established, thereby keeping credentials out of the HTTP upgrade request and its associated metadata.

### FWD-02-016 WP2: Secrets in world-readable systemd unit files *(Low)*

***Fix note**: This issue was fixed by Forward Email through placing the secrets in an environment file. Cure53 verified the fix, confirming that the problem no longer exists.*

During static analysis of the Ansible playbooks, it was noted that systemd service units for MongoDB and Redis backups are created with all required secrets, including the backup encryption secret and object storage credentials (access key and secret key), stored directly as inline environment variables within the service files themselves. systemd unit files are, by default, created with world-readable permissions (*0644*), meaning that any local user or process on the system can read these files and recover the credentials contained within. This would allow users to access and decrypt all backup files from Cloudflare R2.

**Affected files:**

- *forwardemail.net/ansible/playbooks/redis.yml*
- *forwardemail.net/ansible/playbooks/mongo.yml*

**Affected code (*redis.yml*):**

```
- name: Create Redis backup systemd service
   copy:
     dest: /etc/systemd/system/redis-backup.service
     content: |
       [Unit]
       Description=Redis Backup to Cloudflare R2
       OnFailure=failure-notification@%n.service

       [Service]
       Type=oneshot
       RemainAfterExit=no
       TimeoutStartSec=600
       TimeoutStopSec=30
       Environment="AWS_ACCESS_KEY_ID={{ lookup('env',
    'AWS_ACCESS_KEY_ID') }}"
       Environment="AWS_SECRET_ACCESS_KEY={{ lookup('env',
    'AWS_SECRET_ACCESS_KEY') }}"
       Environment="BACKUP_SECRET={{ lookup('env', 'BACKUP_SECRET') }}"
       Environment="REDIS_PASSWORD={{ lookup('env',
'REDIS_PASSWORD') }}"
```

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

Cure53 recommends that secrets should never be placed directly inline in systemd unit files. It is advised that multiple secure alternatives exist:

- Using systemd's *LoadCredential* directive, which loads secrets from files with restricted permissions through a secure file descriptor mechanism.
- Using an environment file (via *EnvironmentFile=*) placed outside the service unit directory with owner-read-only permissions (e.g., *0600*).
- Leveraging Ansible Vault to encrypt the secrets at rest within the playbook repository itself. Additionally, it is advised that the backup encryption key and the AWS credentials should be separate secrets with independent rotation policies, so that a compromise of one does not automatically grant access to the other.

## FWD-02-019 WP1: Unescaped regex widens anti-spoofing rules *(Info)*

***Fix note**: This issue was fixed by Forward Email by escaping the configuration value before usage in a regex. Cure53 verified the fix, confirming that the problem no longer exists.*

Analysis of the *isArbitrary* anti-spoof logic found that it constructs a regular expression from the *WEB_HOST* environment variable. As this variable contains the domain, this results in an unescaped value being used as regular expression. Specifically, the dot in the domain will then match arbitrary characters and not specifically a dot value. As a result, the domain matching logic is too loose, and allows more values than the configured one. For example, if *WEB_HOST* is configured to *forwardemail.net*, then the value *forwardemailXnet* will also produce a match.

**Affected file:**
*forwardemail.net/helpers/is-arbitrary.js*

**Affected code:**

```
const REGEX_DOMAIN = new RE2(new RegExp(env.WEB_HOST, 'im'));
[...]
function isArbitrary(session, headers) {
  [...]
  if (!config.isSelfHosted) {
    // extract domain from From header
    const fromDomain = parseHostFromDomainOrAddress(from);

    // check if domain contains our brand name or is a homograph/TLD attack
    let isDomainImpersonation = false;

    if (fromDomain) {
      // get root domain of the From address
      const fromRootDomain = parseRootDomain(fromDomain);

      // check for exact domain match (e.g., forwardemail.net)
      if (REGEX_DOMAIN.test(fromDomain)) {
```

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

```
    isDomainImpersonation = true;
}
```

To mitigate this issue, Cure53 recommends escaping any value before using it as a regular expression. While the impact in this case is limited, the issue could become more problematic with future changes, and might lead to bypass possibilities.

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

## Conclusions

As noted in the *Introduction*, this August 2026 penetration test and source code audit was conducted by Cure53 against the core codebase and underlying infrastructure of Forward Email, as well as the Nodemailer codebase.

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

This engagement represented the second penetration testing assignment conducted by Cure53 for Forward Email. Testing targeted the core repository - including the infrastructure and application code consisting of Ansible playbooks to set up the production environment, setup scripts to configure self-hosted environments, and several vanilla JavaScript entry points that spin up the Forward Email web, API, and mail stacks.

Overall, the number of vulnerabilities identified increased when compared to the previous audit. This can likely be attributed to the fact that Cure53 has utilized and deepened its knowledge from the previous test and spent more than double the amount of days in order to extend the coverage deeper into the Forward Email architecture and dependencies. The latter becomes especially visible when observing the most critical finding made in this report - a Prototype Pollution vulnerability within the *mailauth* library that effectively allows RCE on the production mail server component (FWD-02-001).

Although the scope of this audit was originally intended to focus on the mail stack, this repeatedly shifted and steered away toward other areas that were identified during the analysis and code searches. This also led to the discovery of three 2FA / OTP bypasses (FWD-02-003, FWD-02-006, and FWD-02-008).

On other occasions, the team observed that some of the countermeasures and hardening measures recommended in the previous audit are already effective - such as the deployment of a CSP that mitigated the HTML injection discussed in FWD-02-013. This is generally a good sign, although it was also shown that the CSP is an obstacle that can potentially be bypassed, as was observed in FWD-02-010. It is advised that this could potentially be improved by avoiding the use of the *strict-dynamic* directive in the CSP.

Both the *nodemailer* and *forwardemail.net* repositories were analyzed for outdated and vulnerable dependencies. It was found that the dependency posture of the nodemailer repository was sound, with no notable outdated or vulnerable components identified.

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

In contrast, the *forwardemail.net* repository was found to contain a large number of third-party dependencies with severe vulnerabilities (FWD-02-011). Due to time constraints a full impact analysis was not performed by Cure53 testers. Therefore, it was not possible to determine if these vulnerabilities are reachable in a production scenario and this finding was documented merely as a defense-in-depth measure. Nevertheless, Forward Email developers swiftly analyzed and updated or patched a significant portion of the affected dependencies. Further updates are currently blocked by a Node v18 lock and deep transitive dependency chains.

Additionally, the Infrastructure as Code (IaC) configuration was examined, by thoroughly reviewing the Ansible playbooks. It was determined that various secrets were supplied to services through environment variables defined directly within systemd service files (FWD-02-016). While environment variables are a common pattern for configuration, it is advised that their use in systemd unit files without adequate access controls or encryption leaves the secrets potentially exposed to any process or user capable of reading the service file or inspecting the process environment. As such a compromise would require an attacker to be able read this file either through local access, a code execution flaw or process logic abuse, this issue was documented as another in-depth hardening recommendation only.

The API backend was also analyzed for insecure defaults and fallback mechanisms. During this analysis, it was observed that sensitive credentials, such as the API token, could be transmitted via query parameters (FWD-02-015). It is advised that this behavior is insecure by default, as query parameters are frequently logged by web servers, proxies, and browser histories, thereby increasing the risk of credential exposure. However, with the exception of this finding, no further issues were identified in this area.

A very close analysis of the MX service and its associated features (such as forwarding configuration via DNS TXT records) revealed multiple processing flaws. Firstly, the regex substitution support of forwarding patterns enables trivial DoS attacks that can crash the MX service within seconds (FWD-02-014). Secondly, a DMARC policy bypass can be achieved due to inconsistent *From* header processing when multiple *From* headers are inserted into an email (FWD-02-018). These flaws underline that subtle issues, such as slightly different processing of the *From* header, can lead to significant vulnerabilities in the application's overall security posture. Going forward, it is recommended to enhance the input validation further, and to limit features such as regex patterns to an absolute minimum for the functionality required.

The deduplication logic used for attachments was found to use a weak hash function. This enables the fast discovery of duplicate hash values that would enable attackers to replace the attachment contents of unrelated emails when the attachment produces the same hash value (FWD-02-020). During attachment insertion, a duplicate hash is immediately treated as an identical attachment, instead of first comparing the attachment body. It is advised that this should be remedied, and that a stronger, cryptographically secure hash function should be used, to avoid the simple production of hash collisions by malicious actors.

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

Cure53's previous review of the *forwardemail.net* codebase revealed a Server-Side Request Forgery (SSRF) vulnerability that enabled attackers to connect to internal IP addresses. While this specific flaw was found to have been remedied, the filtering logic was still found to be flawed, and bypasses were possible by using IPv4-mapped IPv6 literals (FWD-02-017).

In order to properly fix this vulnerability, it is advised that string-based pattern matching should be replaced, by first canonicalizing IP addresses, and then using libraries such as *ipaddr.js* to parse and categorize the target IP address.

The management of the Sieve logic through the Web UI and API were also closely reviewed, as Sieve scripts are a powerful tool which can enable the manipulation of emails during processing. As the previous assessment already found flaws with the Sieve processing, the security validator was inspected more closely, and was found to be bypassable through the Web UI, by creating a valid, allowed script that is subsequently modified to contain malicious code. As the update logic does not properly check the security validator output, such modifications are allowed to pass through (FWD-02-021). It is advised that this underlines how important proper validation is in all endpoints. Further, it shows that central, early validation logic is required in order to catch all paths that enable scripts to be modified.

Overall, it was found that *forwardemail.net* already has multiple protections in place that prevent abuse. Nevertheless, the large number of features available provides many possibilities for bugs to occur. As this report clearly shows, these bugs can lead to severe vulnerabilities that harm the overall security posture of the system, and enable compromise.

Going forward, once all of the identified findings have been addressed, it is strongly recommended to conduct further penetration testing with the same scope, until the number and severity of vulnerabilities naturally decreases. This is expected, given the strong effort and reactiveness shown by the developers in both preparing this testing engagement, and in fixing vulnerabilities.

The emergence of LLM-aided attackers presents a growing challenge for open source projects such as forwardemail.net, as these technologies significantly accelerate both the identification of security flaws and the development of working exploits. The decision to undertake such an in-depth assessment demonstrates the developers' dedication to maintaining a secure system and proactively staying ahead of these evolving threats.

Cure53 would like to thank Nick Baugh from the Forward Email LLC team for his excellent project coordination, support and assistance, both before and during this assignment. His rapid responses to and mitigation of vulnerabilities prior to the conclusion of this test demonstrate a professional and serious approach to security.