ASSURED

SECURITY CONSULTANTS

Report

Mullvad API pentest

Emilie Barse, Benjamin Svensson

Project	Version	Date
MUL007	v1.3	2022-12-20

ASSURED SECURITY CONSULTANTS

REPORT

Project	Version	Date
MUL007	v1.3	2022-12-20

Executive summary

Between 2022-11-07 and 2022-11-29 Assured Security Consultants performed an API penetration test on behalf of Mullvad VPN.

The in-scope items are the API published on `api.mullvad.net`, its related services and configuration. A testing environment was installed on `[REDACTED HOSTNAME]` to perform the test against. The test was performed with access to the API source code and with shell access to the installed test environment.

No critical, high or medium rated issues were identified during the penetration test and the overall security of the API is deemed good. We identified one issue related to how secrets are managed in the docker environment, which could expose unnecessary secrets if an attacker identifies a way to read environment variables. All other issues are related to the API implementation. There are a few issues where an attacker could send the API input which is passed directly to a function without being filtered for any unexpected characters. Even though we weren't successful in exploiting these issues, it's good practice to verify user input before passing it to functions.

This report is listing the security issues found, along with recommendations for fixing or mitigating them. In our conclusions we discuss the issues and address apparent patterns in areas where security is lacking.

Issues were found with the following risk severity assessments (number of issues):

Critical 0 High 0 Medium 0 Low 7 Note 5

We recommend to

- Implement Hashicorp Vault to manage secrets in Docker
- Remove unnecessary read-write permissions on bind-mounts in docker
- Implement encryption in transit for the Redis serivce
- Verify user input and only accept expected data. If any special characters is required these should be handled with extra care
- Remove account numbers from GET request URLs
- Handle the unexpected error in `/app/v1/replace–wireguard–key`
- Use the same type of throttling with IP-blocking for both authentication endpoints. Or use the same authentication endpoint for both use cases

- Enforce increased complexity and length of the admin passwords, implement blacklist.

Assured would like to thank Simon, Alexander, Grégoire, Michal and Richard for their support during this penetration test. We are happy to answer any questions and provide further advice.

Contents

1 Introduction

1.1 Background

Assured AB (Assured) was contracted by Mullvad VPN to perform a security assessment on their API and related source code.

1.2 Constraints and disclaimer

This report contains a summary of the findings found during the project period. This report should not be considered as a complete list of all vulnerabilities, security flaws and/or misconfigurations.

1.3 Project period and staffing

Assured started the project on 2022-11-07 and finished on 2022-11-29.

This report was last reviewed on 2022-12-20.

Involved in the penetration testing were Assured consultants Emilie Barse and Benjamin Svensson.

1.4 Risk rating

In this report we have assessed the severity of issues and identified vulnerabilities. The levels of severity are rated according to the OWASP Risk Rating Methodology [1].

Table 1: OWASP Risk Rating overall severity model

Overall risk severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

As Table 1 visualizes, the overall risk assessment is determined from a combined likelihood and impact of an identified vulnerability or security issue. A value from 0 to 9 is assessed for each variable, where 0-2 is determined LOW, 3-5 is MEDIUM and 6-9 is HIGH.

Likelihood is dependant on attributes related to threat actors and the identified vulnerability, with factors such as: the skill level and motivations of the threat agents; how easily the vulnerability can be found and exploited, and; how likely an exploit may be detected.

Impact depends on technical and business factors, such as: level of loss of confidentiality, integrity, availability and accountability; potential financial damage; potential brand damage, and; potential violations of privacy.

Please note that the severity assessment is made by Assured consultants and ratings may differ from the resource owners' ratings.

2 Scope and methodology

2.1 Scope

The API (api.mullvad.net) and all its related functions are in-scope for this assessment. A test environment was installed and configured to be used as a target for the assessment. Part of the test was also the message-queue, dns-janitor functions and the docker configuration.

The test was done as a white-box test with access to code, credentials for the different business roles, and ssh access into the test system host. The environment was setup with access to third party test environment and related API keys, for services such as Stripe and Paypal.

2.2 Methodology

The test was performed using primarily dynamic testing and source code review. The source code was mainly used as a support in the dynamic testing and a full code review has not been performed.

Network traffic and logs were examined during the test to understand the application flow and to check the management of events in the backend as well as to identify any personal identifiable information (PII).

Mullvad API The API was tested using static source code analysis for Python called bandit to find common security issues in Python code. The source code was also used to aid in the dynamic testing to setup and execute functional API requests and understanding the business flow of the API.

All endpoints in the API was tested dynamically using Burp Suite Pro.

We followed the OWASP API Security project top 10 list [2] to ensure good coverage during the test, the list includes the following items:

- API1 Broken Object Level Authorization
- API2 Broken User Authentication
- API3 Excessive Data Exposure
- API4 Lack of Resources & Rate Limiting
- API5 Broken Function Level Authorization
- API6 Mass Assignment

- API7 Security Misconfiguration
- API8 Injection
- API9 Improper Assets Management
- API10 Insufficient Logging & Monitoring

In addition to this list we examined other areas of importance described in the OWASP testing guide [3] and not covered by the API top 10 list:

- Business logic
- Error handling
- Weak cryptography

The API is implemented on the Django framework. We investigated the code and configuration for common security issues related to Django and lack of security best practices.

Docker The docker configuration was analyzed in runtime using trivy, and through source code review, identifying configuration issues in the docker environment.

message-queue The message-queue has one external endpoint which implements a websocket managing different channels. In this case, only the ”wireguard” channel exists which is used by relay servers to subscribe to messages about changes in peer devices. This endpoint was dynamically tested for authentication/authorization and injection. The code was reviewed and the network traffic examined to verify how it handles messages.

dns-janitor We performed static code analysis on the source code using gosec to identify possible issues in the code and analyzed the result.

2.2.1 Tools used

- gosec
- bandit
- Burp Suite Pro
- trivy

2.3 Limitations

Since throttling is implemented in the API we decided to patch the settings for throttling allowing us to be more efficient in our dynamic testing.

3 Observations - Docker

3.1 Low Unencrypted network traffic to Redis

Likelihood: LOW (2), Impact: MEDIUM (3)

The network traffic to the Redis backend is sent in plaintext from an application layer point of view. However, the network traffic between hosts are sent in VPN tunnels.

An attacker who is able to intercept the network traffic in the VPN protected network can get hold of the Redis password and all traffic sent to Redis. The attacker can use the password to edit any information in Redis which could cause Denial-of-Service. Plaintext application layer traffic also makes it easier to perform man-in-the-middle attacks.

Several docker instances are communicating with the Redis backend and some interesting information is sent over the network.

Figure [REDACTED SCREENSHOT] shows a network packet where the redis password is sent from the API host ([REDACTED IP]) to the redis_exporter instance ([REDACTED HOSTNAME]).

[REDACTED SCREENSHOT]

Figure [REDACTED SCREENSHOT] shows the packet sent from the mullvad-api ([REDACTED IP]) to redis ([REDACTED IP]) after a wireguard pubkey is added with the request POST /www/wg–pubkeys/add/.

[REDACTED SCREENSHOT]

Figure [REDACTED SCREENSHOT] shows the network packet where the paypal authentication token is updated.

[REDACTED SCREENSHOT]

Security best practice is to encrypt traffic to backend databases and other services and encrypt traffic where credentials are sent, to implement a defence-in-depth solution.

Other services communicating between docker instances are also using plaintext protcols. However, as long as the docker instances are located in the same host and only communicating on host-local interfaces, this is not an issue.

We recommend using end-to-end TLS encryption on the traffic to redis. If the infrastructure is changed in the future so that docker instances may end up on different hosts, be careful to check that any communication between services is encrypted.

3.2 Note Unnecessary read-write permissions on bind-mounts

Multiple docker containers bind-mount filesystems from the host. The docker containers have excessive read-write permissions to some of these mount points.

The following examples show multiple docker containers where we find the mount permissions to be excessive. If an attacker identifies a way to breach the docker container they could write to these directories or files.

Example 1: celery_worker_api-standalone-2

```
[
  {
    "Type": "bind",
    "Source": "/home/mad/api_data",
    "Destination": "/var/lib/api/data",
    "Mode": "rw",
    "RW": true,
    "Propagation": "rprivate"
  }
]
```

Example 2: monitor_api-standalone-2

```
[
  {
    "Type": "bind",
    "Source": "/home/mad/api_cluster/monitor/telegraf.conf",
    "Destination": "/etc/telegraf/telegraf.conf",
    "Mode": "rw",
    "RW": true,
    "Propagation": "rprivate"
  }
]
```

Example 3: api_api-standalone-2

```
[
  {
    "Type": "bind",
    "Source": "/home/mad/api_data",
    "Destination": "/var/lib/api/data",
    "Mode": "rw",
    "RW": true,
    "Propagation": "rprivate"
  }
]
```

Example 4: dns-janitor_api-standalone-2

```
[
  {
    "Type": "bind",
    "Source": "/home/mad/api_cluster/dns-janitor",
    "Destination": "/etc/dns-janitor",
    "Mode": "rw",
    "RW": true,
    "Propagation": "rprivate"
  }
]
```

Our recommendations is to add `:ro` to all above bind mounts if write permission is not necessary for the function of the docker container.

NOTE: This issue was reported as fixed by Mullvad prior to the release of this report.

3.3 Note Secrets in docker-compose.yml and environment variables

We identified that the Mullvad API cluster is storing secrets using environment variables and as clear-text in the docker-compose file. The secrets are placed in clear-text in a file which is loaded to environment variables when the docker containers are started.

The following example shows a container with a clear-text password in `docker-compose.yml`.

Example 5: Clear-text password in docker-compose.yml

```
  redis-exporter:
    container_name: 'redis_exporter_api-standalone-2'
    image: bitnami/redis-exporter:1.45.0@sha256:25
        eb538edfab6a96cdc4011acfd9cc316ad227aa907f4151efe0de28d2ae8dab
    command:
      - --redis.addr=redis://[REDACTED HOSTNAME]
      - --redis.user=[REDACTED USERNAME]
      - --redis.password=[REDACTED PASSWORD]
    ports:
      - '[REDACTED IP]:9121:9121'
    networks:
      nginx:
```

Docker swarm provides a secret management layer called "docker secrets", however docker swarm orchestration is not used in Mullvads docker setup.

Docker-compose has a simpler implementation of secrets, where the container mounts a file from the host to use as a secret. Example:

```
services:
  frontend:
    image: awesome/webapp
    secrets:
      — server—certificate
secrets:
  server—certificate:
    file: ./server.cert
```

Still, this implementation would require the secrets to be stored in files on the host.

After discussions with the Mullvad team we understood that Mullvad is implementing a Hashicorp Vault instance in their environment to manage secrets for their services which would be a much better way to include secrets in their service since Vault will provide encryption at rest, in transit and audit trails for access to the secrets.

We recommend Mullvad to configure and implement a solution to use Hashicorp Vault to be leveraged for secrets for their API service.

4 Observations - API

4.1 Low HTML injection in email

Likelihood: MEDIUM (4), Impact: LOW (2)

As described in section 4.2, users should not be able to input arbitrary data to functions in the API. We discovered that an unauthenticated user can POST to the endpoint `[REDACTED URL]` which allows for any user input. This input constructs an email, as seen in example [REDACTED CODE EXAMPLE], and sends it to `support@mullvad.net`.

[REDACTED CODE EXAMPLE]

Figure [REDACTED SCREENSHOT] shows an example request and response which resulted in a email being sent to `support@mullvad.net`.

[REDACTED URL]

Djangos `EmailMessage` parameter `message` defaults to content-type `text/plain`. Therefore the only way this issue could be exploited requires the receiving e-mail client to not respect the content-type specified by the sender which is deemed unlikely since the Mullvad team have internal policies addressing secure e-mail settings.

We recommend to make sure to filter the user input and verify it only allows for the expected format for the different email fields. For the `message` field at least HTML tag characters should be encoded.

NOTE: This issue was reported as fixed by Mullvad prior to the release of this report.

4.2 Low Unverified user input

Likelihood: MEDIUM (3), Impact: LOW (1)

All input coming from untrusted sources (such as VPN users) should be used with extra care in any system. Using unfiltered data from untrusted sources creates unnecessary attack surfaces. We identified that the endpoint `[REDACTED URL]` picks the platform and version as strings from the URL and looks up the values in the backend database. However, before the values are looked up they are sent as part of a string to `statsd` and they are also used for a lookup in the cache implemented in Redis.

The following code is handling this request:

[REDACTED CODE EXAMPLE]

We have not seen that the releases parameters can be exploited, and the network traffic shows that dangerous characters in the parameters are exchanged for ”_” before they are forwarded to the telegraf docker instance.

```
0000  02 42 ac 13 00 02 02 42 ac 13 00 04 08 00 45 00 .B.....B......E.
0010  00 7d 2f bd 40 00 40 11 b2 86 ac 13 00 04 ac 13 .}/.@.@.........
0020  00 02 81 a0 1f bd 00 69 58 a7 61 70 70 2e 61 70 .......iX.app.ap
0030  70 5f 76 65 72 73 69 6f 6e 5f 63 68 65 63 6b 3a p_version_check:
0040  31 7c 63 7c 23 70 6c 61 74 66 6f 72 6d 3a 32 32 1|c|#platform:22
0050  32 32 2c 76 65 72 73 69 6f 6e 3a 73 65 6c 66 2e 22,version:self.
0060  5f 5f 69 6e 69 74 5f 5f 2e 5f 5f 67 6c 6f 62 61 __init__.__globa
0070  6c 73 5f 5f 2c 70 6c 61 74 66 6f 72 6d 5f 76 65 ls__,platform_ve
0080  72 73 69 6f 6e 3a 4e 6f 6e 65 0a                rsion:None.
```

We recommend to make sure to filter the user input as soon as possible in the code and verify it only allows for the expected format used in the URL.

NOTE: This issue was reported as fixed by Mullvad prior to the release of this report.

4.3 Low IP blocking can be circumvented

Likelihood: MEDIUM (3), Impact: LOW (2)

The endpoints [REDACTED URL] and [REDACTED URL] produce the same type of authorization token which can be used interchangeably for access to the /app/ and /accounts/ endpoints. The difference is that [REDACTED URL] will block the IP after some failed attempts and [REDACTED URL] only implements rate-limiting.

An attacker who discovers that both of these endpoints exist, can easily circumvent the IP blocking. Figure [REDACTED SCREENSHOT] and [REDACTED SCREENSHOT] show the two usage of the two auth endpoints.

[REDACTED SCREENSHOT]

[REDACTED SCREENSHOT]

The Mullvad team have made an active choice to implement IP-blocking for their web authentication endpoint. An mobile app endpoint used for the same purpose should be considered to be implemented with the same precaution since it exposes the same functionality.

Though this is not a critical functionality for the security of the API, it is bad security practice to have to have different functions with different policy to generate the same authorization token.

We recommend to either remove one of the endpoints or differentiate the functionality so that the tokens cannot be used interchangeably.

4.4 Low Sensitive information in URL

Likelihood: LOW (2), Impact: MEDIUM (3)

The account number is included in some [REDACTED URL] GET requests. The account number is considered sensitive because it is the value used for authentication and can be compared to a password.

Placing sensitive information in the URL makes it easier for an attacker to get access to it, than if it is sent in a POST request. The URL with the account number is stored in the web server logs, it may be stored in the user's browser or sent to other sites in the Referer header. In this case, the account number will at least end up in the nginx access.log. This log is however deleted with a short interval.

The following GET requests with account number in the URL have been found:

https://[REDACTEDURL]/admin/accounts/account/2245202913091650/change/

https://[REDACTEDURL]/admin/accounts/partneraccount/de6ccbeccfe74a6ea7aadcd0827e7b52/change/

https://[REDACTEDURL]/admin/accounts/wireguardpeer/?device__account__token__exact=2245202913091650

[REDACTED URL]

We recommend to use POST request for all requests handling account numbers, and to place the account number in the body of the request instead of in the URL.

4.5 Low Admin password change does not enforce policy

Likelihood: LOW (2), Impact: MEDIUM (3)

There are password policy rules presented on the Password change page in the admin application which is deemed insufficient for administration services. Figure [REDACTED FIGURE] shows the rules. It is possible to change to passwords such as Sommar2022! or the username repeated (<username><username>).

Sommar2022! is a commonly used password, and the username repeated should be considered to similar to personal information.

[REDACTED SCREENSHOT]

The admin functionality is restricted to the Mullvad internal network, which reduces the impact.

Our recommendation is to implement a blacklist approach when setting administrators passwords. Using for example https://github.com/danielmiessler/SecLists/tree/master/Passwords to check for commonly used passwords, a more extensive list than the one included in Django. The password blacklist used should also include Swedish words normally used in passwords such as Sommar, Vinter, Mullvad, etc. The path to the password blacklist can be defined with the parameter `password_list_path` for the validator `CommonPasswordValidator`. Since the password can be set to the username repeated twice, it would be a good idea to increase the `max_similarity` for the validator `UserAttributeSimilarityValidator` to force the users to create password similar to their user attributes (username, email, etc).

An example implementation can be seen in the example below.

Example 6: Recommended settings for Password Validators in Django

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
        'OPTIONS': {
            'max_similarity': 0.1,
        }
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
        'OPTIONS': {
            'password_list_path': '/path/',
        }
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

NOTE: This issue was reported as fixed by Mullvad prior to the release of this report.

4.6 Low Potential Slowloris attack (DoS)

Likelihood: LOW (2), Impact: MEDIUM (3)

Three issues were found using the SAST tool gosec which noted that no timeout was set when setting up the http server in both dns-janitor and message-queue. Without a timeout this could lead to a DoS.

Example 7: gosec result

```
message—queue—main/main.go:154—157 — G112 (CWE—400): Potential Slowloris Attack because
    ReadHeaderTimeout is not configured in the http.Server (Confidence: LOW, Severity: MEDIUM)
    153:
  > 154:  server := &http.Server{
  > 155:    Addr:    *listen,
  > 156:    Handler: api.Router(),
  > 157:  }
    158:
```

Example 8: dns-janitor: gosec result

```
src/dns—janitor—main/main.go:56] — G114 (CWE—676): Use of net/http serve function that has no
    support for setting timeouts (Confidence: HIGH, Severity: MEDIUM)
    55:  http.Handle(”/metrics”, promhttp.Handler())
  > 56:  go func() { log.Fatal(http.ListenAndServe(conf.Metrics, nil)) }()
    57:
```

Example 9: message-queue:gosec result

```
src/message—queue—main/main.go:148 — G114 (CWE—676): Use of net/http serve function that has
    no support for setting timeouts (Confidence: HIGH, Severity: MEDIUM)
    147:    server.Handle(”/metrics”, promhttp.Handler())
  > 148:    log.Fatal(http.ListenAndServe(”:9999”, server))
    149:  }()
```

We recommend to add the parameter `ReadHeaderTimeout` to your http.server declaration, see Example 4.6.

```
server := &http.Server{
  Addr:    *listen,
  Handler: api.Router(),
  ReadHeaderTimeout: 3 * time.Second,
}
```

NOTE: This issue was reported as fixed by Mullvad prior to the release of this report.

4.7 Note Unexpected behaviour of refund with amount zero

The endpoint `[REDACTED URL]` handles the amount of 0 as a refund of the whole payment amount. There is no information about this in the user interface and no comment in the code. This may be intended behaviour.

Figure [REDACTED SCREENSHOT], [REDACTED SCREENSHOT] and [REDACTED SCREENSHOT] shows the procedure of adding the 0 refund and the resulting refund page.

[REDACTED SCREENSHOT]

[REDACTED SCREENSHOT]

[REDACTED SCREENSHOT]

Figure [REDACTED SCREENSHOT] shows the refund in the database.

[REDACTED SCREENSHOT]

Example 10: payments/models.py, class Refund

```
    @classmethod
    def create_object(cls, payment, amount=None, support_user=None, comment=None):
        if payment.is_refunded():
            raise exceptions.PaymentAlreadyRefunded
        if payment.amount < 0 or (amount is not None and amount < 0):
            raise exceptions.RefundNegativeAmountError
        if amount and amount > payment.amount:
            raise exceptions.RefundGreaterError

        try:
            amount = amount if amount else payment.amount
            extra_kwargs = cls.before_create_object(payment, amount)

            return cls.objects.create(
                payment=payment,
                amount=amount,
                support_user=support_user if support_user else '',
                comment=comment,
                **extra_kwargs,
            )
```

We recommend to add a comment to the web page and in the code that amount 0 can be used to refund the whole payment. If it is not intended, the code should be corrected to give an error for 0 refunds.

NOTE: This issue was reported as fixed by Mullvad prior to the release of this report.

4.8 Note Recent actions shows wrong amount on refunds

The "Recent actions" list on the main admin page does not show the correct amount for 0 refunds, and shows refunds which were not carried through because of errors.

A refund with amount 0.00000000, where actually the whole payment is refunded, is shown as 0E-8. A very big refund amount or a negative refund amount, which were not refunded, are shown like they were refunded.

[REDACTED SCREENSHOT]

As discussed with the developers, the "Recent actions" amounts are not the ones used for accounting, and this may not have much impact.

We recommend correcting the messages to show the actual refund amount, to avoid confusion for the administrators.

NOTE: This issue was reported as fixed by Mullvad prior to the release of this report.

4.9 Note Unhandled error

Likelihood: LOW (2), Impact: LOW (2)

The endpoint `/app/v1/replace-wireguard-key` triggers an internal server error if the following series of requests are performed:

1. Create a device on an account (with no previous devices) using the `/app/v1/replace-wireguard-key` endpoint.

2. Remove the device with the request `DELETE /accounts/v1/devices/<device ID>`

3. Repeat the same `/app/v1/replace-wireguard-key` request again.

The endpoint response does not give any detailed information more than "Internal server error" to the end user, but the log says:

```
{"levelname": "ERROR", "asctime": "2022-11-18 14:58:20,049", "name": "core.api.exceptions", "
    funcName": "exception_handler", "lineno": 50, "message": "", "exc_info": "Traceback (most
    recent call last):\n File \"/usr/local/lib/python3.8/site-packages/rest_framework/views.py
    \", line 506, in dispatch\n response = handler(request, *args, **kwargs)\n File \"/mullvad-
    api/app/views.py\", line 311, in post\n new_device = register_device(\n File \"/usr/local/
    lib/python3.8/contextlib.py\", line 75, in inner\n return funcargs,
```

```
**kwds)\n File \"/mullvad-api/accounts/services.py\", line 37, in register_device\n
    _check_pubkey(pubkey)\n File \"/mullvad-api/accounts/services.py\", line 116, in
    _check_pubkey\n raise PubkeyAlreadyInUse()\naccounts.exceptions.PubkeyAlreadyInUse"}
```

This is not a security issue in itself, but proper error handling makes the code more robust and can prevent security problems caused by future changes of the code.

We recommend adding error handling in the function register_device(), to handle this case.

NOTE: This issue was reported as fixed by Mullvad prior to the release of this report.

5 Conclusions and recommendations

Assured have performed an API penetration test against the test environment [REDACTED HOSTNAME] which implements the same functionality as api.mullvad.net. The test was performed using white-box methodology together with source code for the API and related services and with shell access to the environment.

We followed the OWASP API Security project top 10 list to ensure good coverage during the test, the list includes the following items:

- API1 Broken Object Level Authorization
- API2 Broken User Authentication
- API3 Excessive Data Exposure
- API4 Lack of Resources & Rate Limiting
- API5 Broken Function Level Authorization
- API6 Mass Assignment
- API7 Security Misconfiguration
- API8 Injection
- API9 Improper Assets Management
- API10 Insufficient Logging & Monitoring

We tested the message-queue functionality and we can connect to the websocket and receive messages using the proper authentication. However, the websocket does not accept any incoming messages, which was verified during code review.

We reviewed the dns-janitor application using a SAST tool (gosec) and brief manual review. The dns-janitor component does not parse any external user input and is not possible to interact with from an attackers point. Unless an attacker compromises the API for remote code execution we couldn't identify any possible attack vector introduced by this component.

All objects or UUID we discovered have been tightly related to the users token and/or groups assigned to the user object.

API1: Objects in the API are referred to using GUIDs, the account number or the WireGuard public key. These IDs cannot be guessed. In addition, only objects belonging to the authenticated user can be accessed, even if the ID is known. In this case, the objects are mainly the WireGuard public keys and devices.

API2: User authentication is based only on the account number. This is considered sufficient, since no sensitive information is stored about the user. A low risk vulnerability found is that two different endpoints exist for generating authorization tokens for users, where one will block the IP if too many account number guesses are done and the other will only throttle the requests. However, it is still unlikely that a user will guess an existing account number even without IP blocking, and the reward is low. Django’s user authentication functionality is used, while the authorization token handling is a custom implementation.

API3: The data exposed by the API endpoints is minimal and no personal information is stored in the backend. No vulnerabilities have been found in this area.

API4: Rate limiting and throttling are implemented for endpoints where it’s reasonable. We couldn’t identify any way to bypass throttling in an exploitable way for any endpoint.

API5: Authorization of all endpoints in the API has been tested, and no vulnerabilities have been found. Some endpoints can be accessed without authorization, but these are intended to be publicly accessible. Different roles (user, partner, admin) have access to different endpoints, and the access control for the roles is implemented correctly.

API6: No objects other than the ones expected by the endpoints were processed by the API. No vulnerabilities have been found in this area.

API7: Django configuration of the API was analyzed and checked against common security issues. No vulnerabilities have been found in this area.

API8: We aimed to cover all endpoints which take some kind of user input, starting with the most exposed ones. Some issues were identified in this area. User input is in some cases not filtered or verified to be as expected by the API and an attacker could include arbitrary data in the fields. We didn’t manage to exploit any of these issues.

API9: OpenAPI documentation exists for all public endpoints. However the internal endpoints lacks documentation and we recommend Mullvad to implement documentation for those as well.

API10: The logging is minimal in the API and other services. This is by design to avoid storing any personal information. There is some error logging of events such as erroneous payment requests. The logging is considered adequate.

Summary of recommendations:

- Implement Hashicorp Vault to manage secrets in Docker
- Remove unnecessary read-write permissions on bind-mounts in docker
- Implement encryption in transit for the Redis serivce
- Verify user input and only accept expected data. If any special characters is required these should be handled with extra care
- Remove account numbers from GET request URLs
- Handle the unexpected error in `/app/v1/replace-wireguard-key`
- Use the same type of throttling with IP-blocking for both authentication endpoints. Or use the same authentication endpoint for both use cases
- Enforce increased complexity and length of the admin passwords, implement blacklist.

References

[1] OWASP, “OWASP Risk Rating Methodology.” https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology, 2019.

[2] OWASP, “OWASP API Security Project.” https://owasp.org/www-project-api-security/, 2022.

[3] OWASP, “OWASP Testing Guide v4.” https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents, 2016.