Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest-Report TunnelBear VPN & Software 10.2020

Cure53, Dr.-Ing. M. Heiderich, J. Larsson, M. Rupp, BSc. B. Walny, BSc. T.-C. “Filedescriptor” Hong, MSc. F. Fäßler, MSc. J. Hector, MSc. S. Moritz, MSc. N. Krein

Index

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Introduction

> *"TunnelBear respects your privacy. We will never monitor, log, or sell any of your browsing activity. As the only VPN in the industry to perform annual, independent security audits, you can trust us to keep your connection secure."*
>
> From https://www.tunnelbear.com/

This report documents the findings of an annual security assessment conducted by Cure53 against the TunnelBear VPN service compound in October 2020. This recurring examination concerns penetration testing, reviews of configurations and infrastructure, as well as dedicated auditing of the TunnelBear VPN scope.

Last thorough investigations that Cure53 performed against similarly selected items took place in November 2019. To give some context, the tests and audits then yielded a total of twelve findings characterized by various severity ratings. Importantly, some items received *Critical* scores in terms of risks they presented.

The reported project was requested by TunnelBear and promptly enacted by nine testers from the Cure53 team. They have been selected on the basis of their best-suited skills and worked on areas best corresponding to their individual expertise. The total budget stood at forty person-days, which were invested into preparing, executing and finalizing this project.

As usual, the scope was very broad and the budget for this test was sufficient for reaching very good coverage levels. In order to optimize structured division and progress of tasks, the work has been split into several work packages (WPs). In WP1, Cure53 focused on the TunnelBear client applications, which were subjected to code auditing and penetration testing. Same methods were deployed in WP2 against browser extensions used by TunnelBear. This was followed by WP3, with penetration tests and configuration reviews dedicated to TunnelBear VPN infrastructure. Tests and audits of TunnelBear FilterPods took center stage in WP4, while WP4 zoomed in - via a code audit - on the PolarBear backend. Continuing and complementing the above, WP6 moved to frontend and public sites exposed by TunnelBear and WP7 offered a configuration review and audit of the AWS infrastructure serving TunnelBear.

White-box methods were once again chosen and deployed in the frames of this cooperation. Cure53 was given access to all relevant materials and sources, with the aim of optimizing coverage. The project started on time and progressed efficiently. Communications were done using the usual, shared Slack channel, wherein members of both the TunnelBear and Cure53 teams were able to collaborate. Note that a second

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

shared channel was created on Slack during the audit to discuss a feature-specific concern raised by Cure53 with TunnelBear and McAfee staff, specifically relating to WP4: TunnelBear FilterPods. Communication was productive and helpful, the tests and audits managed to proceed without any hindrance and the coverage levels reached by the Cure53 team were very good.

As for the outcomes, Cure53 managed to find several fresh and relevant issues, with nineteen findings in total. Five discoveries were categorized as security vulnerabilities and fourteen represented general weaknesses of lower exploitation potential, resultingly placed in the *Miscellaneous* category of findings. On the one hand, it needs to be noted that the number of findings exceeds the total from 2019. On the other hand, the overall severity levels have gone down substantially, which is a positive sign. The testing team was unable to spot *Critical* issues during this 2020 exercise. Only one flaw was graded as a *High* risk, while the remaining problems were located in the realm of *Medium* and lower severity scores. This indicates progress and shows that the TunnelBear complex is on the right track from a security perspective.

The report will now shed some light on the scope and the test setup. Findings are then discussed within two groups of vulnerabilities and general weaknesses, with chronological order used for reporting within the two larger finding-types. Each finding will be accompanied by a technical description, a PoC where possible, as well as finders' perspectives on mitigation and fix advice. After that, the report will close with a conclusion, in which the Cure53 team will elaborate on both general and specific impressions gained over the course of this October 2020 test and audit. Tailored hardening advice is also incorporated into the final section.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Scope

- **Penetration Tests & Audits against TunnelBear VPN Software & Servers**
  - **WP1**: TunnelBear client apps (code audit & pentest)
    - **macOS**
      - Download link:
        - https://s3.amazonaws.com/tunnelbear/downloads/mac/TunnelBear.zip
      - Repositories:
        - *tunnelbear-apple*
        - *tunnelbear-apple-openvpn.git*
        - *TBMapKit*
    - **iOS**
      - Download link:
        - https://apps.apple.com/us/app/tunnelbear-secure-vpn-wifi/id564842283
      - Repositories:
        - *tunnelbear-apple*
        - *tunnelbear-apple-openvpn.git*
        - *TBMapKit*
    - **Android**
      - Download link:
        - https://play.google.com/store/apps/details?id=com.tunnelbear.android
      - Repositories:
        - *tbear-android*
        - *polarbear-android*
        - *Tb-vpn-android*
    - **Windows**
      - Download link:
        - https://tunnelbear.s3.amazonaws.com/downloads/pc/TunnelBear-Installer.exe
      - Repositories:
        - *Tunnelbear-windows*
        - *polarbear-windows*
  - **WP2**: TunnelBear browser extensions (code audit & pentest)
    - Download link:
      - https://chrome.google.com/webstore/detail/tunnelbear-vpn/omdakjcmkglenbhjadbccaookpfjihpa
    - Repositories:
      - *web-tb-browser*

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

- **WP3**: TunnelBear VPN infrastructure
  - Repositories:
    - *Opscode*
- **WP4**: TunnelBear FilterPods (pentest & code audit)
  - Repositories:
    - *filterpods-2019-audit*
- **WP5**: TunnelBear PolarBear backend (code audit)
  - Repositories:
    - *polarbackend*
    - *Backend*
    - *Axon*
- **WP6**: TunnelBear frontend & public sites (pentest & code audit)
  - https://www.tunnelbear.com
  - https://www.tunnelbear.com/teams
  - Repositories:
    - *web-tb-com*
    - *web-tb-landing*
- **WP7**: TunnelBear AWS infrastructure (configuration review & audit)
  - Polarbackend:
    - https://github.com/TunnelBear/polarbackend
  - Tundra:
    - https://github.com/TunnelBear/tundra
  - Assorted Terraform modules:
    - *tf-module-logdna-router*
    - *tf-module-read-secrets*
    - *tf-module-vmf-proxy*
    - *tf-module-app-server*
    - *tf-module-load-balancer*
- **Tests-accounts were created by Cure53**:
- **SSH server access was granted for Cure53**
- **Binaries were shared with Cure53**
- **Sources were shared with Cure53**
  - Note that some source code for selected Work Packages could only be accessed using a remote connection to a TunnelBear-maintained system
- **Test-supporting material was shared with Cure53**

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *TB-08-001*) for the purpose of facilitating any future follow-up correspondence.

TB-08-001 API: Rate-limiting results in user-lockout *(Medium)*

It was discovered that the implemented rate-limiting mechanism for the two *login* API endpoints can be used to lock users out of the platform. If more than ten requests are sent to the endpoint within a short amount of time, the server responds with the *429 Too Many Requests* status. At first glance, it appears to be a normal rate-limiting but, as it turns out, the limit is not bound to the IP of the client but rather to the email address sent in the request. This allows an attacker to lock out other users from the platform while sending more than ten requests with one email address to the affected endpoints (see below).

Once the 15-minute unlock time expires, the attacker can still send new requests to permanently prevent users from logging into the applications. Due to the fact that the email address of a user must be known for a successful attack, the issue was rated *Medium*.

The following Proof-of-Concept shows how the user with the email *seba+tb@cure53.de* can be prevented from logging in to the web application by sending the request more than 10 times in a short amount of time.

PoC Request #1. Locking users out of the web app:

```
POST /core/web/api/login HTTP/1.1
Host: api.tunnelbear.com
Content-Type: application/x-www-form-urlencoded
[...]

username=seba%2Btb%40cure53.de&password=167&withUserDetails=true&v=web-1.0
```

Response (after ~10 requests):

```
HTTP/1.1 429 Too Many Requests
[...]
Rate limiting
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

The following Proof of Concept shows how the user with the email *seba+tb@cure53.de* can be prevented from logging in to a TunnelBear client by sending the request more than 10 times in a short amount of time:

PoC Request #2. Locking users out of the client app:

```
POST /v2/token HTTP/1.1
Host: api.tunnelbear.com
Content-Type: application/json
[...]

{"username":"seba+tb@cure53.de","password":"81","device":"","grant_type":"passwo
rd"}
```

Response (after ~10 requests):

```
HTTP/1.1 429 Too Many Requests
[...]
{"error_code":10007,"error_message":"Please try again
later.","error_info":"AuthLimiter","error_id":6780943}
```

Please note that already authenticated users are not affected by this attack. However, it is recommended to not bind the rate limiting to the email received via the related requests. Instead, it is advised to bind the rate limiting to the client's IP address. Additionally, a captcha can be considered to add as well to make the attack less efficient.

TB-08-007 FilterPods: Use of *innerHTML* leads to XSS in *block* page *(Low)*

While auditing the HTML templates of the FilterPods, it was discovered that *innerHTML* is used in combination with untrusted user-input. This means a malicious user can execute arbitrary JavaScript. Since *filterpods* are not supposed to be part of the regular VPN (see TB-08-010), the impact here is not very clear. The endpoint vulnerable to this issue seems to give way to retrieving a *block* page which is displayed back to the user.

The execution domain will then depend on the domain that serves the HTML returned from the endpoint. In the worst-case scenario, sensitive information such as user-sessions or API tokens can be leaked. However, during the test it could not be determined how this endpoint is utilized. The vulnerable code excerpt with the relevant parts highlighted can be found below.

Affected File:
filterpods-2019-audit-master/mms-sb-blockpage-1.0.5/template/block_i18n.html

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Affected Code:

```
function updateLink() {
    domain = {{ .Domain }};
    url1 = document.getElementById("url1");
    if (url1) {
        url1.innerHTML = trunc(domain, 30);
    }
};
```

As shown above, the template uses a domain which is provided by the user through the request URL. The input is then truncated to 30 bytes and passed to *innerHTML,* allowing the execution of JavaScript. The following is a Proof-of-Concept (PoC) URL that triggers an alert box. However, note that arbitrary JavaScript execution can be achieved by fetching external JavaScript.

PoC URL:
http://172.17.2.7/block/%3Cimg%20src%20onerror=alert(1)%3Ecc/meow

Regardless of the overall impact, it is recommended to avoid the usage of *innerHTML* and use DOM manipulation instead. An alternative would be to properly sanitize the *domain* string. Note that *innerHTML* needs to be considered dangerous and should be avoided at all cost.

TB-08-010 FilterPods: Missing network restrictions allow access *(High)*

*W*hen auditing the FilterPods component, Cure53 discovered that the services specific to the FilterPods are accessible to regular VPN users. In order to access the services, it is enough to just connect to the VPN using the Windows client or similar.

After discussing the implications with the TunnelBear team, it was communicated that these FilterPods should not be accessible to regular VPN users. Although the FilterPods are running in a different IP range, this does not prevent unauthorized access sufficiently.

FilterPods accessible through relevant IPs:

- *frontend-api:* 172.17.2.3 ports: 8087, 9110
- *filterpod-client-api*: 172.17.2.5 ports: 8441
- *dns-proxy:* 172.17.2.4 ports: 53 (udp) 9110
- *mms-sb-blockpage:* 172.17.2.7 ports: 80, 443, 9110
- *mms-sb-redirector*: 172.17.2.6 ports: 80, 443, 9110

First, it is recommended to introduce a network firewall as a means to prevent direct access. If access is factually needed, a gateway service should be utilized in order to

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

access the Filter Pod components. Second, it should be considered to implement an authentication mechanism requiring basic authentication credentials for accessing the API endpoints.

TB-08-011 Web: Arbitrary redirect via *Core2* route *(Low)*

Testing items within the *tbearDashboard2* codebase led to the discovery of a slight input sanitization error in the handler for redirecting clients to *surveymonkey*. More specifically, the following affected code shows that the user-supplied URL parameter is not properly checked upon redirection.

Affected File:
backend-develop/tbearDashboard2/app/controllers/Application.scala

Affected Code:

```
val allowedRedirectUrl: String =
  Await.result(gs.get[String](gs.NpsSurveyUrl, "https://www.surveymonkey.com"),
10 seconds)

// Purpose to redirect users acting on NPS in app messages through TB domain to
survey
def redirectAction(url: String) =
  RateLimitedAction.async { implicit request =>
    if (url.startsWith(allowedRedirectUrl)) Future.successful(Redirect(url))
    else Future.successful(Forbidden)
  }
```

Consequently, the passed *url* parameter only has to *start* with *https://www.surveymonkey.com*, which is not enough to prevent arbitrary redirections to potentially malicious URLs. As the following link demonstrates, any visitor can be redirected onto a website with potentially malicious content.

Example URL:
https://www.tunnelbear.com/core2/redirect?url=https://www.surveymonkey.com.cure53.de

Arbitrary redirects are not seen as serious threats. However, they should still be prevented due to giving the attackers a capacity to hide malicious content (for example Phishing domains) behind the innocent-looking TunnelBear domain. In any case, this should be treated as an input validation issue and fixed accordingly. One approach would be to make sure that a trailing *slash* is included in the *surveymonkey* URL. Defining the URL as "*https://www.surveymonkey.com/*" should fix the issue.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

TB-08-019 Crypto: Known plain-text attack on *sendLogs* in AES *(Medium)*

The iOS application's feature for sending diagnostics uses crypto in an ineffective way. TunnelBear leverages the *HybridCrypto* class which encrypts the diagnostics data with a combination of AES and RSA. The idea is to encrypt the data with the AES block-cipher and then encrypt the AES key with public-key crypto using RSA. Then the AES-encrypted blob is sent to the server together with the RSA encrypted key. This scheme requires the AES keys to be securely generated for each transmission, however TunnelBear has a hardcoded key, which leads to an ineffective crypto layer.

Affected File:
./ios/TunnelBear/Code/Controllers/Onboarding/LandingPageViewController.swift

Affected Code:

```
private func sendLogs() {
    do {
      let aes = try AES(password: CryptoKeys.AESPasswordKeys.sendLogs.rawValue)
      let crypto = try HybridCrypto(derCertificateNamed: "send-logs", aes: aes)
      let zip = try LogArchiver.zipArchive()
      let zipData = try Data(contentsOf: zip)
      let ciphertext = try crypto.encrypt(zipData)
      let ciphertextKeyFile = URL(fileURLWithPath:
"\(NSTemporaryDirectory())/ciphertext.key")
      let ciphertextDataFile = URL(fileURLWithPath:
"\(NSTemporaryDirectory())/ciphertext.data")
      try ciphertext.key.write(to: ciphertextKeyFile)
      try ciphertext.data.write(to: ciphertextDataFile)
      // [...]
  }
```

As can be seen in the *sendLogs()* function, AES crypto is initialized with a hardcoded password in *CryptoKeys.AESPasswordKeys.sendLogs*.

Affected File:
./shared/core/Code/Strings/CryptoKeys.swift

Affected Code:

```
public enum AESPasswordKeys: String {
  case sendLogs = "TunnelBear.sendLogs"
}
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Once *HybridCrypto* is initialized with the hardcoded password *"TunnelBear.sendLogs"*, the *encrypt()* function is called and encrypts the log data with AES. Right after that, a new plain-text, specifically containing the AES key and the IV, is prepared for encryption with RSA.

Affected File:
./shared/core/Code/Utils/HybridCrypto.swift

Affected Code:

```
  public func encrypt(_ plaintext: Data) throws -> Ciphertext {
    let ciphertextData = try aes.encrypt(plaintext)
    let plaintextKey = Data("\(aes.hexKey)\n\(aes.hexIV)".utf8) as CFData
    var error: Unmanaged<CFError>?
    guard let ciphertextKey = SecKeyCreateEncryptedData(publicKey, algorithm,
plaintextKey, &error) as Data? else {
      throw error!.takeRetainedValue()
    }

    return Ciphertext(key: ciphertextKey, data: ciphertextData)
  }
```

The IV is a secure random 16-byte value required for proper decryption. At first sight, the crypto seems safe here because the random IV is never directly exposed and only transmitted within the safely encrypted RSA blob. However, the sequence means an attacker can perform a ‘known plain-text’ attack to recover the IV.

Steps to Reproduce:

1. Encrypt *"AAAAAAAAAAAAAAA"* using *HybridCrypto* with the default AES key *"TunnelBear.sendLogs"*
2. Take notes of the randomly generated IV and the resulting cipher-text
   Example IV: *55748a521ed5590e7d3e7275ea6a9d94*
   Cipher-text: *e6446ce701225b0e594b61c2ae8acb62*
3. Prepare to decrypt the cipher-text with a null-IV *00000000000000000000000000000000*. During decryption the IV is applied with XOR to recover the plain-text, so the result with the null-IV will be the raw output of AES without the IV.
   Decryption result: *1435cb135f94184f3c7f3334ab2bdc95*
4. Because the expected plain-text is known, *"AAAAAAAAAAAAAAA"* (with padding in hex *41414141414141414141414141414100*), it can be XORed with the *1435cb135f94184f3c7f3334ab2bdc95* to recover the IV.

The example above uses a clearly known plain-text of AAAAs to focus on the main attack. An actual attack requires one additional - albeit easy to accomplish - step.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

TunnelBear is encrypting a ZIP archive, thus the plain-text of the first AES block is either known or can easily be guessed. This results in the recovery of the IV and ultimately the full decryption. It is important that the key is randomly generated and no static key is used. The key is already part of the RSA encrypted part, indicating that the server can use private keys to recover the AES key.

Note: *The issue was discussed with the client and the outcome was no extra encryption is needed here, the data is sent by the user manually when diagnosing a particular issue or bug.*

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

TB-08-002 Web: HTML injection in notification email via *team name* *(Info)*

It was found that the emails sent by the platform are not properly shielded against HTML injections. This means that changing data on the website form, i.e. altering the team's *name* into strings that contain HTML characters, has an effect on user capacities. Specifically, the user cannot cause XSS on the website itself, but is able to influence the optics of the messages sent by the platform. This can be noticed when a user receives an email about changes in account-privileges and *Team Update* announcements. An example of the current behavior can be consulted below.

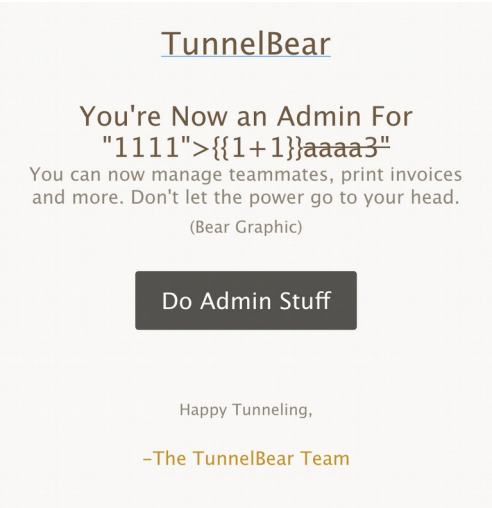

Fig.: HTML injected into own emails

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

The screenshot above shows an HTML injection from the team's *name*. In the resulting email, the *name* parameter is not being sanitized, leading to an HTML injection. The following query was used to change the *team name*.

Request:

```
POST /core/web/team/name HTTP/1.1
Host: api.tunnelbear.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:81.0)
Gecko/20100101 Firefox/81.0
Accept: application/json, text/plain, */*
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
TB-CSRF-Token: 596dae579be30c7db7c65b5d811890516d9f777e
X-XSRF-TOKEN: 596dae579be30c7db7c65b5d811890516d9f777e
Content-Length: 70
Origin: https://www.tunnelbear.com
Cookie: PLAY_SESSION=2655b3a7c85de4be65c4c9a125eaeac17b83e5b2-
___AT=596dae579be30c7db7c65b5d811890516d9f777e&tbcsrf=596dae579be30c7db7c65b5d81
1890516d9f777e&___TS=1603632240817&sessionid=%40CK-79ac65d4-c0b4-44cb-bd5c-
af93bd89e590;

name=1111%22%3E%7B%7B1%2B1%7D%7D%3Cs%3Eaaaa3&password=[...]
```

Just as with the web application more broadly, it is here recommended to make sure that all user-controlled data employed in email templates gets escaped and encoded. The right choice would be to use HTML encoding as a means to completely mitigate the attack.

TB-08-003 Android: *Secure* flag missing on views *(Info)*

During the assessment of the Android app, it was discovered that the *FLAG_SECURE* security flag is not used to protect views that display sensitive content. By setting the flag for Android views, the app's windows can no longer be manually "screenshotted". Additionally, the items will be excluded from automatic screenshots or screen-recordings, which ultimately prevents screen data from being leaked to other apps. Especially for the implemented views showing sensitive data, such as the *Login* view, it is advised to add the specified flag.

To reiterate, it is recommended to add the *FLAG_SECURE* within the *WindowManager* responsible for handling views like the *WebView*. The flag can be set via *WindowManager.LayoutParams,* i.e. as *FLAG_SECURE* within the function of

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

setFlags(). As for additional information on how to prevent this type of attacks, please refer to the *OWASP Mobile Security Testing Guide*[1].

TB-08-004 AWS: No KMS keys for SSE in SQS queues *(Info)*

The analysis of the configuration attached to *Simple-Queue-Service* (SQS) used by TunnelBear revealed that encryption is not enforced for any of the SQS queues used in the assessed AWS context. If an attacker manages to get access to an AWS object or role that has access to the SQS feature, the content of the queues would be readable and all data stored within the queue will be compromised.

Excerpt of SQS queues without encryption:

```
arn:aws:sqs:us-east-1:113810520231:openvpnmonitor_queue_*
arn:aws:sqs:us-east-1:113810520231:blocked_url_queue_*
arn:aws:sqs:us-east-1:113810520231:blocked_url_queue_test
arn:aws:sqs:us-east-1:113810520231:dedup-test.fifo
arn:aws:sqs:us-east-1:113810520231:dns_update_queue*
arn:aws:sqs:us-east-1:113810520231:kms_cloudtrail_log_notifier*
arn:aws:sqs:us-east-1:113810520231:lambda-job*
```

Even if those queues don't contain any PII2 data, in order to increase the overall security posture, it is recommended to enforce encryption for all queues used by TunnelBear. Furthermore, once encryption is enabled, it is important to use a KMS configuration that relies on customer-managed keys instead of AWS-managed keys. The latter would confirm encryption usage at rest.

TB-08-005 Web: Error messages reveal internal information *(Info)*

It was found that some API endpoints may reveal minor internal information via errors. This happens when unexpected input is provided for certain endpoints and entails appearance of surprising characters or types inside parameters. The actions cause errors in the server-side functionality that handles the submitted input. This does not directly lead to a security issue, yet it might aid a malicious user in acquiring more internal information.

The following example relates to the affected endpoint and demonstrates the present behavior.

[1] https://mobile-security.gitbook.io/mobile-security-testing-guide/android-testing-g...atic-analysis-8

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Request:

```
POST /core/v2/referral HTTP/1.1
Host: api.tunnelbear.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:81.0)
Gecko/20100101 Firefox/81.0
Accept: application/json, text/plain, */*
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
TB-CSRF-Token: ef86f6b052a0071e0edf1f4d486e18862d5722d6
Cookie: PLAY_SESSION=6d5abe45454b8d65d[...]

{aaaa<>aaaaaaa:}
```

Response Body:

```
Execution exception
JsonSyntaxException occurred : com.google.gson.stream.MalformedJsonException:
Expected value at line 1 column 16 path $.aaaa<>aaaaaaa
This exception has been logged with id 7hddi621e
```

As no critical-level information could be leaked through this finding, the risk has been evaluated as *Info*. While this may be brushed away as just a harmless information disclosure, it should be viewed in the context of generally helping adversaries in their efforts of obtaining different ways to conduct further attacks against the application or the hosts.

It is recommended to store stack traces on the server and only provide a correlation ID on HTTP responses when unexpected errors occur. This will facilitate debugging and investigation of the application issues without revealing information or introducing security vulnerabilities. Error messages must only be composed of static error descriptions that do not include information pertinent to the implementation of the service or the deployed software and/or version.

TB-08-006 Android: Unencrypted *shared preferences* and *database* (*Info*)

During the assessment of the Android app, it was discovered that the application does not always make consistent use of the encrypted *shared preference* feature provided by the Android SDK. This may lead to an information disclosure in case a local attacker is able to get *root* access to the phone. Sensitive information stored within the *shared_prefs* data folder in plain-text, such as the VPN token (see below), could be revealed.

In addition, implemented third-party libraries also have access to the protected *app data* folder and are, therefore, able to read such kind of data, too. Moreover, if the application

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

is vulnerable to a local arbitrary file-read attack, e.g., via an insecurely implemented content provider, it could also be used to read such files.

Affected File:
vpn-android/src/main/java/com/tunnelbear/vpn/models/VpnConfig.java

Affected Code:
```
@SuppressLint("ApplySharedPref")
private VpnConfig(Context context,
        ArrayList<VpnServerItem> vpnServers,
        String vpnToken,
        Bundle bundle) {
    mVpnToken = vpnToken;
    [...]
    Gson gson = new Gson();
    String serializedConfig = gson.toJson(this);

    context.getSharedPreferences(CONFIG_RAW, MODE_PRIVATE)
         .edit()
         .putString(CONFIG_RAW, serializedConfig)
         .commit();
}
```

Example *shared preferences* file (*vpnconfig.xml*):
```
<string name="vpnconfig">
[...]
,”mVpnToken;”TBR-547b2e2e-6fcd-4d86-be5c-b3f237fca423”}</string>
```

Besides the flaws described above, it was discovered that sensitive data, such as the VPN token, is also stored in plain-text within the *tunnelbear_database* used by the app.

Affected File:
tbear-android-develop/app/src/main/java/com/tunnelbear/android/persistence/KeyValuePairHelper.kt

Affected Code:
```
suspend fun set(key: Keys, value: Any?) {
    if (value == null) {
        removeKeyValuePair(key)
    } else {
        db.keyValuePairDao().insert(KeyValuePair(key, gson.toJson(value)))
    }
}
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Example entry for *tunnelbear_database*:

```
USER_INFO:
{"account_status":"NORMAL","data_limit_bytes":1572864000,"id":0,"is_data_unlimit
ed":false,"vpn_token":"TBR-547b2e2e-6fcd-4d86-be5c-b3f237fca423"}
```

It is advised to use the provided wrapper class called *EncryptedSharedPreferences* to encrypt sensitive data stored within the *shared_prefs* folder, so as to make the application more robust against the illustrated attacks. The wrapper class uses the Android Keystore for handling the master key and is used to encrypt/decrypt all other keysets. For more information, please refer to the official Android guide on storing data more securely[2]. It is also advisable to encrypt the contents in the database using a key from the Android Keystore.

TB-08-008 macOS: Hardening the *Privileged Helper* (*Info*)

The TunnelBear macOS daemon suffered from several privilege escalation issues in the past. Those problems were related to the difficulty of validating that the XPC connection comes from the actual TunnelBear client, as well as from an attacker having the capability to fool the daemon into executing a malicious binary. In the current version no privilege escalation issue could be identified, however TunnelBear could harden the daemon even further.

Currently, the defenses are located on two levels. Firstly, the XPC connection is validated through the *auditToken* and then the *runSignedExecutable* ensures that the binary to execute is properly signed. The *auditToken* is not available in all macOS versions, thus a fallback to the vulnerable process ID method happens on older versions.

Secondly, *runSignedExecutable* is not the only exposed method. The daemon also has other functions such as *killProcess* or *fetchLogs*, which offer no additional checks. These methods do not directly lead to a privilege escalation but could provide a powerful primitive to chain with other bugs. Due to the first layer of defense, a malicious program cannot execute those remote procedure calls. If TunnelBear is executed on an old macOS version, or if an attacker finds a code injection into the TunnelBear client, the calls could still be reachable.

While the implementation is generally strong, offering an additional layer of hardening could assist TunnelBear in expanding the scope of restrictions as far as the capabilities of the *DaemonService* methods are concerned. For example, hardcoding the allowed file paths for the *fetchLogs* function or hardcoding allowed processes being killed by *killProcess,* could serve as some additional options.

[2] https://developer.android.com/topic/security/data

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

TB-08-009 Web: No HTTPS for data export link in emails *(Low)*

It was found that the links embedded in the data export emails rely on an unencrypted HTTP channel. An attacker who has the ability to eavesdrop (i.e. a Man-in-the-Middle adversary) on the connection of a user can take advantage of techniques like *sslstrip* to proxy clear-text traffic to the user-victim.

Sample Email:

```
[...]
Click the button below to download your data in a zip file.
<a
href="http://email.bearpostoffice.com/c/eJxdjstugzAQRb_G7IL8wA5ZsDAB8mpR2ySq1J1t
7IQoAYpNm_D1ddJNVc0srmZGZ06VzKhAKArqBEMMEUQzyCJCUeiLFtM5T1meZVMeUQIiKLXou9a61pha
6VC1l-
CYKKMYi6QxMJaESjZFzEiplDQMK4VnwTk5OtdZQDjAhW9xbZvJYHVfDZfOhpaE4iLGthHf9o70F6Y-
60pb1deda3uAKU6dts6HCKqh15SElZ7s9mWZP6U5f5tUwok7LRzrDpCCv2-
5Utrajb6tKkAyvlnx9fx5v1t_LBcvdLMol68As_za1b32ZhlikLIYxQT58bY-
NML5P36xxQMZHwanT6LS2yMeRNN8xdlY3lB78iBKsqBP_lv_Ov8x_gHWAXIF">Download my data</
a>
```

It is recommended to embed the links with a consistent use of HTTPS to eliminate the possibility of eavesdropping for a MitM actor.

TB-08-012 AWS: Expired ACM certificates *(Info)*

Cure53 noticed that that several certificates, specifically attached to the ACM configuration and used by TunnelBear as active in the AWS context, have expired. This in itself should not be regarded as a severe security issue but instead points to a bad security practice and potentially a lack of sufficient renewal process in regard to certificate renewal.

Excerpt for the resources with expired certificates:

- `aws:acm:us-east-1:113810520231:certificate/cb8f0491-b28b-4e94-9e88-d01fdd1bd35f`
  - `www.bearsmyip.com` **`(expired for 9 days)`**

- `aws:iam::113810520231:server-certificate/cloudfront/bearsmyip/bearsmyip`
  - `bearsmyip/wwwbearsmyip` **`(expired for 1114 days)`**
  - `bearsmyip/wwwbearsmyip2` **`(expired for 1106 days)`**
  - `bearsmyip/wwwbearsmyip3` **`(expired for 1104 days)`**

- `aws:iam::113810520231:server-certificate/cloudfront/blockbear/sep_15_blockbear`
  - `Sep_15_blockbear` **`(expired for 772 days)`**

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

- `aws:acm:ca-central-1:113810520231:certificate/84463049-4dd2-4ba4-a463-b602bc3400ce`
  - `Test-aws.polargrizzly.com` **(expired for 103 days)**

- `aws:acm:ca-central-1:113810520231:certificate/260ebaaf-ef49-4d35-a5d7-88992e8e14fc`
  - `*.polargrizzly.com` **(expired for 162 days)**

It is recommended to ensure that the configurations attached to SSL certificates used by TunnelBear are up-to-date. It needs to be guaranteed that a process or policy is in place to notify system operations that a certificate is close to its expiry date. Automation of the renewal process could be considered in order to maintain only valid SSL certificates in the production environment.

TB-08-013 AWS: Insecure TLS Configuration Used *(Medium)*

While analyzing the configuration attached to the CloudFront and certificate concepts adopted by TunnelBear, it was found that the current configuration relies on insecure defaults. These are, in particular, vulnerable to numerous attacks, especially as TunnelBear has several distributions relying on the deemed insecure TLSv 1.0.

Excerpt from the CloudFront configuration:

Distributions using TLSv 1.0:

```
arn:aws:cloudfront::113810520231:distribution/E28VTN2Q8UHW4A
arn:aws:cloudfront::113810520231:distribution/E3QY63UHI7SHAH
arn:aws:cloudfront::113810520231:distribution/EKA4U3XSAMAHC
arn:aws:cloudfront::113810520231:distribution/E6W2H9L3N2O2H
arn:aws:cloudfront::113810520231:distribution/E14LNUC77M87XI
```

Distributions configured to use HTTP-only:

```
arn:aws:cloudfront::113810520231:distribution/E28VTN2Q8UHW4A
arn:aws:cloudfront::113810520231:distribution/E3QY63UHI7SHAH
arn:aws:cloudfront::113810520231:distribution/EKA4U3XSAMAHC
arn:aws:cloudfront::113810520231:distribution/E6W2H9L3N2O2H
arn:aws:cloudfront::113810520231:distribution/E14LNUC77M87XI
```

Distributions missing configuration for HTTPS:

```
arn:aws:cloudfront::113810520231:distribution/E3QY63UHI7SHAH
arn:aws:cloudfront::113810520231:distribution/E2R3JO3YVE0Q6J
arn:aws:cloudfront::113810520231:distribution/E3T45YE1X0GLV1
```

It is recommended to review the configuration attached to the specified distributions and ensure that the current configuration is not relying on insecure defaults.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

TB-08-014 AWS: DynamoDB encryption relies on AWS-owned keys *(Medium)*

While analyzing the configuration used for the DynamoDB, it was found that the current encryption schema used by TunnelBear for their DynamoDB tables relies on KMS. This is a sound security practice, except for the fact that the current configuration relies on KMS with AWS-owned keys. In order to ensure a robust encryption scheme TunnelBear should consider changing the configuration to CMKs instead. This will foster having full control over the encryption process for the data.

In order to ensure integrity for the server-side encryption, it is recommended to migrate from AWS-managed keys to customer master keys[3]. This will enhance the data security aspect of the stored information in the affected tables. Switching from AWS-owned CMK to customer-managed CMK can be done through AWS Key Management Service.

***Note**: A list of keys was provided in the original version of the report and later removed upon request by the client.*

TB-08-015 AWS: Mutable ECR repositories *(Info)*

While analyzing the configuration attached to the ECR used by TunnelBear, Cure53 found that mutability is permitted on numerous ECR repositories. When the "*MUTABLE*" flag is set, the *repositories* tag can be overwritten or modified. In turn, this h could introduce Time-Of-Check and Time-Of-Use issues that could be leveraged by an attacker to gain an initial foothold or establish a pivoting point for post-exploitation activities.

In order to increase the overall defense-in-depth concepts attached to ECR, it is recommended to further investigate the current configuration. Mitigating potential attack vectors that can be leveraged with mutable repositories is highly encouraged.

***Note**: A list of repositories was provided in the original version of the report and later removed upon request by the client.*

TB-08-016 AWS: Insecure configuration on *metadata* instance *(Medium)*

An analysis of the configuration attached to EC2 instances used by TunnelBear demonstrated that the current configuration has the instance *metadata* endpoint enabled. If an attacker was able to reach the *metadata* endpoint through a Server-Side-Request-Forgery or a similar attack, the *metadata* layer would provide them with privileged information that can be queried from this endpoint.

[3] https://docs.aws.amazon.com/dynamodb-encryption-client/latest/devguide/client-server-side.html

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Since this attack vector is commonly used, AWS has developed additional protection against approaches targeting the *metadata* service. The IMDSv2[4] server protects the instances from SSRF attacks by implementing a token that can only be obtained by making a specific request using the HTTP *PUT* requests.

Affected Instances:

```
arn:aws:ec2:us-east-1:113810520231:instance/i-68e968db
arn:aws:ec2:ca-central-1:113810520231:instance/i-0ae8b71ec2ec69c80
arn:aws:ec2:ca-central-1:113810520231:instance/i-0ea89d9692d5300d6
arn:aws:ec2:ca-central-1:113810520231:instance/i-02617375f6296cf48
arn:aws:ec2:ca-central-1:113810520231:instance/i-00a09b9e80b6a8a80
arn:aws:ec2:ca-central-1:113810520231:instance/i-0c6fd980b2ff35b5f
arn:aws:ec2:ca-central-1:113810520231:instance/i-0d0e5849c00c89827
arn:aws:ec2:ca-central-1:113810520231:instance/i-05cbabab75943b697
```

In order to improve the overall security posture and adhere to defense-in-depth concepts recommended for the AWS infrastructure, TunnelBear should enable and configure the new and improved *metadata* service instance.

TB-08-017 AWS: Key-rotation process missing in IAM *(Medium)*

The IAM configuration used by TunnelBear has neither active policy nor process in place for facilitating automated *access* key rotation. It was observed that several accounts with attached keys have not been used for an extended period of time and should, therefore, be marked for deletion. Furthermore, Cure53 observed several access keys that were more than 300 days-old. This behavior - in itself - should not be regarded as a security issue but rather as a lack of sanitation on the overall configuration.

In order to minimize the risk of unwanted access due to leaked access keys, a key rotation policy should be implemented. AWS recommends to rotate access keys after 180 days in order to decrease the likelihood of accidental exposure, as well as to protect one's AWS resources against unauthorized access.

***Note**: A list of accounts was provided in the original version of the report and later removed upon request by the client.*

[4] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/configuring-instance-metadata-service.html

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

TB-08-018 AWS: Stale and unused *objects/roles* in IAM *(Info)*

While analyzing the IAM configuration used by TunnelBear, it was found that numerous *objects* and *roles* have either never been used or have not been used for a long time. Legacy *roles* and *objects* should be deleted from the AWS organization in order to prevent potential misconfigurations that could result in unauthorized access or overly permissive reachability of production resources.

It is recommended to regularly perform IAM housekeeping in order to ensure that stale accounts and *roles* are removed in a timely manner from the AWS organization. This process should be adopted and performed with a designated schedule, so as to protect the production environment from accidental exposure, as well as to safeguard all AWS resources in the context of possible misconfigurations leading to unauthorized access.

***Note**: A list of objects/roles was provided in the original version of the report and later removed upon request by the client.*

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Conclusions

After forty days on the scope in October 2020, nine members of the testing team can confirm that the examined TunnelBear compound makes a fairly good impression with regard to security. The project targeted a vast scope, including the TunnelBear clients, applications, browser extensions, frontend part of the public sites, project infrastructure and connected or underlying internal services. Across these broad scope areas, only five exploitable vulnerabilities were detected by Cure53 members. Crucially, none of the issues posed *Critical* danger for TunnelBear, strongly pointing to the fact that the TunnelBear team aptly avoids and actively prevents the majority of key risk vectors. Since only one *High*-severity bug could be confirmed, Cure53 must underline the clearly stable state of the tested scope items.

As positive results should be embraced for further growth, it is also important to account for the remaining weaknesses. In this context, the array of nineteen findings shows that the project still has room for more targeting hardening in certain areas. This also relates to the lower-priority issues that were unveiled, mostly in connection to flaws that had no significant level of exploitability on their own, but could ultimately be leveraged in sophisticated attacks. Such items should not be disregarded, especially within the advised forward-thinking security models.

Through a temporal lens enabled by long-term cooperation between Cure53 and TunnelBear, the testing team can ascertain the undeniable growth of the complex. Specifically, this project marks the eighth instance of the TunnelBear being subjected to external scrutiny from the Cure53 team. It is quite clear that the higher number of findings can be directly linked with the sheer size and complexity of the TunnelBear compound. Given its breadth, the absence of highly-rated issues is a great achievement. As such, this project's results also underscore the benefits of continuous engagement with external security examinations as the working mechanism towards reducing flaws.

Next, the report will take on a more granular approach to subsequent WPs and their contents, as well as spotted patterns and problems. This is envisioned as more focused and specific advice for teams that engage with a given area on a daily basis. Starting with WP1, which gathered impressions pertinent to the TunnelBear client apps, the Android branch was evaluated as making a really good impression. No serious issues could be observed, even though the app was investigated in terms of fitting into the Android’s ecosystem and handling communication with the Android’s platform API.

The related attack surface is composed of two exported activities, one exported service (protected with permissions) and four exported broadcast receivers (one of them is protected with permissions). It was investigated if and how the application is receiving

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

data through registered custom schemes, data URLs, extra strings or parcelable objects. Serious issues are seemingly averted in this area, meaning that typical local attack scenarios, e.g. by malicious applications, do not pose risks for TunnelBear here.

In addition, the storage encryption was examined, which was implemented with the help of the Android Keystore feature. However, it was found that this approach is not consistently followed and data is stored in plain-text in some *shared preferences* files and a database (see TB-08-006). It would be helpful to harden the app in some attack scenarios where data can be obtained more easily, for instance by adding an extra layer of security to protect user-privacy. Moreover, Cure53 examined whether the application processes files outside the protected data folder or reads data from files with universal access, but no such locations were spotted outside of the designated folders.

Staying with WP1 targets, it was found that the app sends and receives data via Android's broadcast functionality for communication with the TunnelBear widget. First, it was checked whether these data can be intercepted by a sniffer app. Due to the fact that the broadcasts are sent in explicit form and remain in the context of the app, no sensitive data sent by the *sendBroadcast()* function could be received. Second, it was checked if data can be sent to the broadcast receivers, which could influence the limit or other states of the app.

Compared to the last Android audits, more checks have been added to the TunnelBear app. These checks both eliminate a lot of crashes and help mitigate the risk of putting the app into undefined or unwanted states. It is recommended to invest further into user-privacy through leveraging the *FLAG_SECURE* (see TB-08-003). It can be said that the Android app makes a very good impression and has solidly implemented security mechanisms. Adding encryption to all *shared preference* files and databases would increase the difficulty and, possibly, deter some attackers.

The examined Windows client also makes a solid impression regarding implemented defense mechanisms. One of the main investigations of the file and folder permissions set during installation confirmed that only administrative users are able to change files inside the installation directory from the TunnelBear client. This holds up even if the location from the installation is outside the *Program Files* folder, which reduces heavily privilege escalation attacks. Additionally, Cure53 checked whether other files outside the installation directory are used by the *TunnelBear.Maintenance.exe* service, which runs with *SYSTEM* user. It has been verified that only files from the installation directory and from the Windows folder are used, preventing access from normal users.

Cure53 looked into whether the app could be disclosing sensitive data to third-parties, e.g. via DNS queries within an established VPN connection or in writing data to publicly

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

readable log files. However, no such behavior could be detected. The connected API endpoints were examined and held well to scrutiny. The implemented rate-limiting mechanism uses the email address as an indicator that the endpoint receives from the affected request. This allows an attacker to lock out other users from the platform by preventing them from logging into the web app and into the clients (two different endpoints are affected, see TB-08-001). To succeed, an attacker would need a user's email address, so the severity level of this problem has been reduced.

Moving on to macOS, the most critical types of issues here would pertain to privilege escalations. The macOS client uses a helper daemon for implementing privileged actions to change firewall rules, read logs or launch the *openvpn* binary. In previous tests issues in the TunnelBear daemon allowed for privilege escalation, including bypasses to proposed fixes, yet this is no longer the case. Even though the *auditToken* is properly verified, the exposed XPC functions were analyzed for race-conditions and to see if the XPC would be resilient if the *auditToken* layer failed.

Cure53 only has some minor recommendations here, as documented in TB-08-008. OpenVPN is started by the daemon as *root* and exposes a management interface via a unix socket. Non-*root* users can use this socket to talk to OpenVPN and this counts as an additional attack surface. The protocol itself fends off easy attacks, but the necessity of maintaining this interface could be questioned. The iOS client also suffered from problems in the past and these were re-evaluated for regressions. The usage of crypto algorithms was reviewed and it was found that the app encrypts diagnostics upon a user request and sends them over to TunnelBear. These logs are encrypted using a combination of RSA public key cryptography and AES symmetric cipher, however this is done in an insecure way susceptible to known plain-text attacks highlighted in TB-08-019.

Regarding browser extensions examined in WP2, the *manifest.json* configuration was audited. Checked items included making sure content scripts were not exposed to non-TunnelBear domains, unnecessary web accessible resources were not accessible to normal web pages, CSP was not overly lax and unnecessary permissions were not requested for users. No flaws were identified here. Furthermore, the content scripts were examined to similarly good results. The URLs which could influence the state of the proxy were not found and, in connection to proxy settings, Cure53 confirmed that the PAC script is leakage-proof (e.g. in terms of IP leaks and DNS leaks).

Further among WP2 tasks was checking how the script reacts to other web extensions taking control over the proxy settings of browsers, as well as its function of disabling WebRTC tracking. Once again, those areas were free from findings. Rounding up the scope, a more general extension security check was performed to cover XSS, API

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

usage and token management. All in all, the TunnelBear browser extensions are stable and benefit from a good security premise.

The VPN infrastructure has been covered in WP3 with a buckshot approach that included host discovery, content discovery, *nmap* scanning, and so forth. Nothing of note emerged from any of the exposed domains. Reverse-proxy attacks with a main focus on request smuggling also did not yield any negative results. It is safe to say that the public configuration appears alright and quite well-tested at this point. Leakage and negligent misconfigurations have been eradicated.

The TunnelBear FilterPods examined in WP4 is an application built internally to block malicious domains. Since it's not available to TunnelBear users as of now, Cure53 had limited information around its functioning or documentation. All in all, the provided functionality is not expansive and fairly straightforward, while having the services implemented in Go provides good security out of the box. However, TB-08-010 shows that these services should not be reachable by regular VPN users, so there is a notable lack of network separation. Thus, any user within the VPN that knows the IP can access these services. Because the services provided by FilterPods require no authorization, anyone can benefit from complete access. Additionally, TB-08-007 shows an XSS vulnerability due to *innerHTML* being used insecurely. The overall impact here is unclear: the role FilterPods in the larger scheme is unknown, even if no further issues regarding untrusted user-input were spotted.

Regarding the FilterPods backend implementation, the code has a consistently high quality. Untrusted user-input was handled correctly and manual SQL query constructions were not only avoided when possible, but also properly handled when necessary. Interactions with the filesystem were handled correctly, thus mitigating any risks regarding file disclosure or arbitrary file-write. Given the context, it was checked if any functionality can be abused to carry out SSRF attacks, but all requests are constructed in a secure manner. Further, no header injections into crafted requests stemmed from this Cure53 examination. Authentication mechanism was also judged as safe and sound.

Looking at the TunnelBear backend in WP5, Cure53 checked if permissions of the S3 bucket responsible for storing users' GDPR data export were configured properly. It was also checked if the settings of Mailgun and relevant mail delivery functions are secure. As non-HTTPS links are used in the mail body, this could allow malicious MitM actors to steal the link and access user-exported data (TB-08-009). This work package also covered multiple backend applications with a focus laid on auditing the controller code connected to all reachable routes. There are multiple authentication mechanisms, so attention was given to deep dives and audits of *JTW*, OTP, cookies and token checks. Especially the */console* routes were studied in great depth to make sure all endpoints are

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

protected. The console endpoints were also audited to exclude previously reported blind XSS flaws. The generated templates look quite well, with the exception of TB-08-011 which has a very low severity and suggests a slight coding mistake. There are indications of regular testing done by Cure53 resulting in quite some hardening and new vulnerabilities being tough to find. This is of course a positive result and underlines a comprehensive mixture of external and internal work leading to reduced attack surface.

As for WP6, meaning impressions regarding the TunnelBear fronted and public sites, the focus was placed on possible ACL implementation flaws. Simultaneously, possible leaking of potentially sensitive information and parsing issues were also addressed extensively. Cure53 also investigated the functionality for the presence of XSS attacks and similar input-manipulation issues. One of the key aspects is that the testers did not reveal any issues linked to the ACL at the allocated time despite intensive and dedicated searches for compromise pathways. The Cure53 team noted that endpoints clearly determine what can be done by the user and verifies whether certain actions are available for the user prior to the final acceptance of input. Strengths also entail the lack of issues connected with various types of injection attacks, which could compromise the server-side parts of the platform. Besides, despite extensive searches and very good coverage from the Cure53 testers, no noteworthy findings to report in this area.

Finally, the TunnelBear AWS infrastructure review and audit contained in WP7 have also indicated some security-relevant progress. The configuration review phase of this audit resulted in the discovery of eight miscellaneous security flaws, yet the overall risk attached to these flaws should be considered as rather limited. The AWS configuration adopted by TunnelBear shows signs of rapid expansion, as especially evident from tickets attached to the current IAM configuration. Access controls and authorization schemas leveraged by the current configuration could benefit from some additional work in order to ensure that old and unused *objects* are removed. Furthermore, TunnelBear could capitalize on implementing automated procedures that notify system operations of potential unused and legacy configurations attached to the current IAM concepts. In order to further strengthen the overall security posture and privacy aspects of running a VPN service, it is recommended to ensure encryption at rest wherever possible. The current KMS implementation could be improved to solely rely on customer-managed keys, which would ensure that TunnelBear would be the single point of truth holding all the encryption constructs. Finally, Terraform modules and their attached repositories which were audited as a part of this engagement left a sound impression.

To conclude, it is quite clear that the *High*-scoring flaw should be fixed urgently. In addition, Cure53 would like to underline that fixing and resolving all tickets connected with general weaknesses should also be seen as crucial moving forward at this stage of the already praiseworthy state of security. Evading minor risks systematically translates

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

to accomplishing better and better security milestones. It should be noted that even issues that seem minor at first sight tend to accumulate, potentially leading to new attack chains being formed. In conclusion, the results of this autumn 2020 assessment of the infrastructure and perimeter of the TunnelBear project, with a strong focus on the client applications and browser extensions, are indicative of a reasonably stable security posture. The targets have mostly proven robust against the various attacks that Cure53 attempted to execute against them.

All in all, not many exploitable issues were spotted, and the absence of *Critical* findings points to a good result achieved by the TunnelBear complex in this October 2020 assignment overall. This is an excellent foundation on which future investments into the project can be built. Cure53 sees the TunnelBear applications as being on the right track towards the main goal of delivering a secure foundation within their operations and customer services.

Cure53 would like to thank Rodrigue Hajjar, Brànè Petrovic, Arun Tomar, Jared Krause and everyone else from the TunnelBear team for their amazing project coordination, support and assistance, both before and during this assignment.