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

Pentest-Report Tor Browser Apps & Tools 01.-02.2024

Cure53, Dr.-Ing. M. Heiderich, M. Wege, L. Herrera, B. Casaje, A. Belkahla, Y. Yuang

Index

Introduction

"We believe everyone should be able to explore the internet with privacy. We are the Tor Project, a 501(c)(3) US nonprofit. We advance human rights and defend your privacy online through free software and open networks."

From https://www.torproject.org/

This report describes the results of a security assessment of The Tor Project complex, with the focus on various Tor censorship circumvention tools, changes in the Tor browser for Desktop and Android, as well as the OnionShare component. The project, which included a penetration test and a dedicated source code audit, was carried out by Cure53 in January 2024.

Registered as *TTP-03*, the examination was requested by The Tor Project in November 2023 and then scheduled to start in early 2024 to allow both sides sufficient time to prepare. Note that this was not the first time that Cure53 conducted a security analysis for The Tor Project. Just last year, during *TTP-01*, Cure53 had a thorough look at various Tor censorship bypass tools such as *RDSys* and Conjure.

The current test iteration builds on the previous work and expands the scope to allow for more deep-dives during the analysis. In terms of the exact timeline and specific resources allocated to *TTP-03*, Cure53 completed the research in late January and early February of 2024, more precisely in CW04 and CW05. In order to achieve the expected coverage for this task, a total of thirty-four days were invested. In addition, it should be noted that a team of six senior testers was formed and assigned to the preparations, execution, documentation and delivery of this project.

For optimal structuring and tracking of tasks, the examination was split into four separate work packages (WPs):

- **WP1**: Crystal-box penetration tests & code audits against censorship circumvention tools & libs
- **WP2**: Crystal-box penetration tests & code audits of changes in Tor browser for desktop
- **WP3**: Crystal-box penetration tests & code audits of changes in Tor browser for Android
- **WP4**: Crystal-box penetration tests & code audits of changes in OnionShare for desktop

As the titles of the WPs indicate, crystal-box methodology was utilized. Cure53 was provided with the way of reaching the relevant GitHub repositories, binary builds for the relevant desktop applications, as well as all further means of access required to complete the tests.

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

The project could be completed without any major problems. To facilitate a smooth transition into the testing phase, all preparations were completed in January 2024, namely in CW03. Throughout the engagement, communications were conducted via a private, dedicated and shared Signal channel. Stakeholders - including the Cure53 testers and the internal staff from The Tor Project - could participate in discussions in this space.

Although it needs to be underscored that the quality of all project-related interactions was consistently excellent, several questions had to be posed by Cure53. These predominantly related to slight delays with the delivery of files that were necessary in the context of the agreed-upon scope. These minor issues could be resolved swiftly. Ongoing communications and exchanges on Signal contributed positively to the overall outcomes of this project. Cure53 offered frequent status updates about the test and the emerging findings. Live-reporting was done on request of the customer, with various tickets shared on demand.

The Cure53 team succeeded in achieving very good coverage of the WP1-WP4 targets. Of the twelve security-related discoveries, eight were classified as security vulnerabilities and four were categorized as general weaknesses with lower exploitation potential. It should be noted that most of the Tor components examined in the frames of *TTP-03* exposed a robust security posture.

Among the examined targets, the OnionShare desktop application stood out for having a slightly weaker security premise on the whole. To be more specific, it suffered from noteworthy vulnerabilities with a severity rating set to *High*. The inspected chat feature, which is a highly sensitive area that must be implemented securely, was prone to several serious bugs. This included spoofing attacks explained in tickets TTP-03-006 to TTP-03-009.

For the OnionShare component, it must be made clear that the risks were quite paramount, as user impersonation could be achieved. This was connected to *disconnect* messages being triggered incorrectly, with the flaw translating to major privacy impact. The rest of the codebase left Cure53 with a positive impression. Especially the Tor Browser apps seemed solid and very much security-aware, even with regard to the newly implemented features that have not been scrutinized by Cure53 before. The following sections first describe the scope and key test parameters, as well as how the WPs were structured and organized. Next, all findings are discussed in grouped vulnerability and miscellaneous categories. Flaws assigned to each group are then discussed chronologically. In addition to technical descriptions, PoC and mitigation advice will be provided where applicable.

The report closes with drawing broader conclusions relevant to this January-February 2024 project. Based on the test team's observations and collected evidence, Cure53 elaborates on the general impressions and reiterates the verdict. The final section also includes tailored hardening recommendations for the Tor complex, specifically referring to improvements that can be made in the various censorship circumvention tools, Tor Browser for Desktop and Android, as well as in the OnionShare component.

Scope

- **Penetration tests & code audits against the censorship circumvention tools, as well as UI changes, in Tor Browser**
    - **WP1**: Crystal-box penetration tests & code audits against censorship circumvention tools & libs
        - **Snowflake:**
            - https://gitlab.torproject.org/tpo/anti-censorship/pluggable-transports/snowflake
            - https://gitlab.torproject.org/tpo/anti-censorship/pluggable-transports/snowflake-webext
        - **Webtunnel:**
            - https://gitlab.torproject.org/tpo/anti-censorship/pluggable-transports/webtunnel
        - **RDSys:**
            - https://gitlab.torproject.org/tpo/anti-censorship/rdsys
        - **Lox:**
            - https://gitlab.torproject.org/tpo/anti-censorship/lox-rs
        - **Bridgstrap:**
            - https://gitlab.torproject.org/tpo/anti-censorship/bridgestrap
        - **OnionSprout:**
            - https://gitlab.torproject.org/tpo/anti-censorship/gettor-project/OnionSproutsBot
    - **WP2**: Crystal-box penetration tests & code audits of changes in Tor browser for desktop
        - **Tor browser for desktop:**
            - https://gitlab.torproject.org/tpo/applications/tor-browser
    - **WP3**: Crystal-box penetration tests & code audits of changes in Tor browser for Android
        - **Tor browser for Android:**
            - https://gitlab.torproject.org/tpo/applications/firefox-android
    - **WP4**: Crystal-box penetration tests & code audits of changes in OnionShare for desktop
        - **OnionShare for desktop:**
            - https://github.com/onionshare/onionshare
    - **Test-supporting material was shared with Cure53**
    - **All relevant sources were shared with Cure53**

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 during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, each ticket has been given a unique identifier (e.g., *TTP-03-001*) to facilitate any future follow-up correspondence.

TTP-03-001 WP1: Sybil attack on Snowflake broker *(Medium)*

While testing the Snowflake censorship circumvention broker, it was discovered that no rate-limit mechanism has been deployed in the client/proxy offer components. As a result, malicious actors could masquerade as Snowflake proxies to send large amounts of non-functional connection offers to the broker. Thus, the clients would be matched with non-functional proxies, which would artificially make it much more difficult to connect.

Malicious actors could also send large amounts of client requests, which could artificially saturate the real proxies on the network.

It is recommended to implement an IP-based rate-limit system, as well as possibly consider crafting a proof-of-work system to prevent these kinds of attacks.

TTP-03-002 WP1: *POST* requests on *rdsys moat* lack body size limits *(Medium)*

While testing the *rdsys moat* distributor, it was discovered that the */moat/circumvention/settings* endpoint does not properly limit the size of the request's body. A malicious actor could use this to cause a DoS condition on the server by sending a large JSON document. This would cause the server to crash due to an out-of-memory condition. From there, attackers could DoS the server with very few resources, impacting the availability of the bridge distributor.

Affected file:
pkg/presentation/distributors/moat/web.go

Affected code:
```
func circumventionSettingsHandler(w http.ResponseWriter, r *http.Request) {
	w.Header().Set("Content-Type", "application/json")
	enc := json.NewEncoder(w)

	var request circumventionSettingsRequest
	dec := json.NewDecoder(r.Body)
	err := dec.Decode(&request)
```

It is recommended to use a *io.LimitReader* with a reasonable maximum body size limit (e.g., 100KB) to prevent attacks of this nature.

TTP-03-003 WP1: *rdsys moat* unconditionally trusts *X-Forwarded-For* (Medium)

While testing the *rdsys moat* circumvention settings distributor, it was discovered that the server trusted the *X-Forwarded-For* header without offering an ability to configure a "*trust proxy*" setting. In the deployment of a *moat* distributor that is not behind a trusted reverse-proxy, this would allow clients to spoof their IP address, fostering Sybil attacks against the distributor.

Affected file:
pkg/presentation/distributors/moat/web.go

Affected function:
ipFromRequest

A configuration option for *trust proxies* should be implemented and be specific about how many proxies should be trusted by the server. This will help prevent abuse of the overly trusting configurations.

TTP-03-006 WP4: Spoofable *disconnection* in chat mode (*High*)

While testing the chat service, it was discovered that users could spoof the "*disconnect*" event without actually disconnecting their *socket.io* connection. This makes the server think that a user has been disconnected, removing them from the *connected_users* list. At the same time, the user retains the capacity to receive and send messages under the name that they have used to already disconnect themselves from the service.

To spoof the *disconnect* event, an attacker can either patch the *socket.io* client library, removing the portion that prevents clients from manually emitting the *disconnect* event, or they can use a custom *socket.io* client.

An attacker could use this exploit to impersonate users or to spy on them. For example, if Alice and Bob both have access to the chatroom, Bob can set his username to Alice while the real Alice is factually disconnected. Bob can then spoof the *disconnection* operation. When the real Alice joins the chatroom again, she can set her name to Alice without any hurdles. At this point, Bob will receive all messages sent to the room and can inject chat messages into the room in the name of "Alice".

In this "ghost" state it is possible for an attacker to change their username an unlimited number of times without any broadcast message. This can be done by using the *update_username* event, due to an exception that is thrown halfway through the event handler.

The exception causes the session username to be updated but fails to address this issue in the global member list. When combined with TTP-03-009 for an alternative username spoofing attack, the approach would not even require the attacker to know what a user will call themselves in terms of username-choice.

Affected file:
cli/onionshare_cli/web/chat_mode.py

Affected code for fake *disconnection* bug:
```
@self.web.socketio.on("disconnect", namespace="/chat")
def disconnect():
    """Sent by clients when they disconnect.
    A status message is broadcast to all people in the server."""
    if session.get("name") in self.connected_users:
        self.connected_users.remove(session.get("name"))
    emit(
        "status",
        {
            "msg": "{} has left the room.".format(session.get("name")),
            "connected_users": self.connected_users,
        },
        broadcast=True,
    )
```

Affected code for silent username change:
```
@self.web.socketio.on("update_username", namespace="/chat")
def update_username(message):
    """Sent by a client when the user updates their username.
    The message is sent to all people in the server."""
    current_name = session.get("name")
    new_name = message.get("username", "").strip()
    if self.validate_username(new_name):
        session["name"] = new_name
        self.connected_users[
            self.connected_users.index(current_name)
        ] = session.get("name")
```

PoC:
https://files.larry.science/f/rWLAXJqg.mjs

Steps to reproduce:

1. Start OnionShare with the *--local-only* flag. Enable chat mode.
2. Change the port in the PoC code to the one that OnionShare is listening on.
3. Run *node {exploit}.mjs*
4. Notice the following:
    1. The chat log says that the user has disconnected, and the user is accordingly no longer listed on the sidebar.
    2. The user can still read and send messages using the username that they have originally joined with.
    3. The user can change their username without any messages being broadcasted.
    4. Others can change their usernames to be the same as the user in question.

It is recommended to forcefully terminate connections using *flask_socketio.disconnect*[1] in the disconnect handler. This should be done in place of just assuming that the connection is being successfully closed.

TTP-03-008 WP4: Joining chat without broadcast message *(High)*

While testing the chat service, it was discovered that users can join the chatroom without sending a *join* message or being displayed on the list of the connected users. To that end, the covertly joining users could silently spy on other users.

By not sending a session cookie with the *socket.io* connection, an exception is thrown when attempting to validate the username. This skips the process of adding the user to the list of connected users, yet still allows users to receive all chat messages sent to the room.

The problem means that attackers can impersonate users, with the same attacks described also in TTP-03-006 and TTP-03-009.

Affected file:
cli/onionshare_cli/web/chat_mode.py

Affected code:
```
@self.web.socketio.on("connect", namespace="/chat")
def server_connect():
    """Sent by clients when they enter a room.
    A status message is broadcast to all people in the room."""
    if self.validate_username(session.get("name")):
        self.connected_users.append(session.get("name"))
```

[1] https://flask-socketio.readthedocs.io/en/latest/api.html#flask_socketio.disconnect

PoC:
https://files.larry.science/f/0TBwWwcI.mjs

Steps to reproduce this issue are the same as those for TTP-03-006. The PoC simply joins the chat silently and echoes all messages.

It is recommended to ensure that users have a verifiably valid session upon joining.

TTP-03-009 WP4: Chat users can spoof names via control characters *(Medium)*

While testing the chat service, it was discovered that users can use ASCII control characters in their usernames, even though these should get removed during HTML sanitization on the web client. Hence, attackers can spoof usernames of other users relying on the service.

When combined with TTP-03-008 or TTP-03-006, this attack can happen in a manner that is completely silent.

Affected file - lack of validation:
cli/onionshare_cli/web/chat_mode.py

Affected code - lack of validation:
```
def validate_username(self, username):
    username = username.strip()
    return (
        username
        and username.isascii()
        and username not in self.connected_users
        and len(username) < 128
    )
```

Affected file - buggy sanitization:
cli/onionshare_cli/resources/static/js/chat.js

Affected code - buggy sanitization:
```
var sanitizeHTML = function (str) {
  var temp = document.createElement('span');
  temp.textContent = str;
  return temp.innerHTML;
};
```

PoC:
https://files.larry.science/f/slTqDE1X.mjs

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

Steps to reproduce:

1. Start OnionShare with the *--local-only* flag. Enable chat mode.
2. Change the port in the PoC code to the one that OnionShare is listening on.
3. Join the chat and choose the name "Alice".
4. Run *node {exploit}.mjs.*
5. Notice that another "Alice" joins the room and begins to talk.

To mitigate this flaw, it is recommended to ensure that usernames can only contain printable ASCII characters.

TTP-03-010 WP3: Potential phishing via task-hijacking on Android *(Medium)*

Testing confirmed that the Android app does not offer sufficient protection against task hijacking attacks.

The *launchMode* for the *HomeActvity* activity is currently set to *singleTask* for Android API level 29 and lower, which mitigates task hijacking via *StrandHogg 2.0*[2] whilst rendering the app vulnerable via older techniques such as *StrandHogg*[3] and other techniques documented since 2015[4].

The described vulnerability was patched by Google in March 2019 for Android versions 28 and newer. Since the android app supports devices from Android 5 (API level 21), this renders all users running Android 5-8.1 vulnerable, as well as affecting users running unpatched Android devices. The latter is still common in the modern era.

A malicious app could leverage this weakness to manipulate the way in which users interact with the app. Specifically, this could be instigated by relocating a malicious attacker-controlled activity within the screen flow of the user, which may be useful toward instigating phishing or Denial-of-Service (DoS) attacks, as well as theft of user-credentials.

Affected file:
fenix/app/src/main/AndroidManifest.xml

Affected code:
```
 <activity
       android:name=".HomeActivity"
       android:exported="true" android:configChanges="keyboard|
keyboardHidden|mcc|mnc|orientation|screenSize|layoutDirection|
smallestScreenSize|screenLayout"
       android:launchMode="singleTask"
       android:resizeableActivity="true" [...]
```

[2] https://www.helpnetsecurity.com/2020/05/28/cve-2020-0096/
[3] https://www.helpnetsecurity.com/2019/12/03/strandhogg-vulnerability/
[4] https://s2.ist.psu.edu/paper/usenix15-final-ren.pdf

To aid understanding of this vulnerability, a demonstration of a potential exploitation scenario was created and can be consulted via a video linked next.

PoC video:
https://cure53.de/exchange/97865826534172365/TaskHijacking%20PoC.mov

To mitigate this issue, Cure53 advises implementing a selection of appropriate countermeasures. One potential solution would be to set the task affinity of the exported activities to an empty string via *android:taskAffinity=""*. This forces Android to create a random *name* which any future attacker would have difficulty predicting. Additionally, setting the *launchMode* to *singleInstance* can be encouraged, as this approach enforces the creation of a new task for each activity.

TTP-03-011 WP3: Potential DoS due to Deep Link misusage *(Low)*

The Android application employs Deep Links for various tasks, such as initiating the opening of a new tab. Deep Links are URLs designed to guide users directly to specific sections within an application.

The observation was made that no limitation is imposed on the number of tabs that can be opened, presenting an opportunity for a malicious app to exploit this vulnerability. This could potentially result in the initiation of numerous tabs, leading to a Denial-of-Service (DoS) scenario. In practice, high memory usage would take effect on the victim's phone.

To aid understanding of this vulnerability, a demonstration of a potential exploitation scenario was created and can be consulted next in the linked video.

PoC video:
https://cure53.de/exchange/97865826534172365/DoS%20PoC.mov

PoC app:
https://cure53.de/exchange/97865826534172365/Dos_poc.apk

To mitigate this issue, Cure53 recommends refraining from opening a new tab for each execution. This action should be restricted, based on the package name of the application sending the intent.

For a more enduring solution, it is recommended to reconsider the reliance on Deep Links, favoring either App Links or modifying the design to impose restrictions on arbitrary access from other applications. It needs to be acknowledged that malicious applications can easily disrupt user-activity now by launching the exported activities and abusing the Deep Links available.

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may 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 method by which to be called. Conclusively, whilst a vulnerability is present, an exploit may not always be possible.

TTP-03-004 WP1: Limited SSRF attack through Bridgestrap *(Info)*

Bridgestrap was discovered to have no protection against SSRF attacks. Specifically, the Webtunnel transport allows an attacker to send HTTP *GET* requests to an arbitrary path on web servers. If a bridge is tested that resolves to an internal IP, a limited SSRF attack could be abused against internal services on the Tor Project's infrastructure.

There are a number of advanced SSRF techniques that could result in an RCE if specific types of internally hosted services become available. For example, using the TLS mode of Webtunnel, an attacker could abuse the TLS Poison[5] attack against an internal Memcached database.

It is recommended to ensure that deployments of Bridgestrap have zero access to private resources. Alternative solutions involve ensuring that bridges do not resolve to private IP addresses that are vulnerable to DNS rebinding attacks. This would require additional protections to mitigate the issue in a comprehensive manner.

TTP-03-005 WP4: Potential DoS of address in *receive* mode *(Low)*

Testing confirmed that the text submitted during the file upload process in the *receive* mode is unrestricted. This potentially lets malicious users upload a file along with an extensive message, consequently leading to a surge in memory usage.

By sending repeated requests with identical payloads, this exploit could trigger a Denial-of-Service in the *receive* mode. As such, it would result in excessive memory consumption on the user's PC that utilizes OnionShare.

To mitigate this issue, Cure53 recommends incorporating checks and imposing restrictions on the lengths of the *text* parameter, so as to prevent any potential misuse of this item.

[5] https://i.blackhat.com/USA-20/Wednesday/us-20-Maddux-When-TLS-Hacks-You.pdf

TTP-03-007 WP4: *History* tab handles newlines incorrectly *(Info)*

Testing confirmed that the *history* tab lacks optimal handling for newlines when operating in either *receive* or *share* mode. This inadequacy creates a potential issue, allowing for the manipulation of history logs and opening avenues for spoofing. Manipulation of this kind could deceive users or provide false information regarding the actual authenticity of the logs.

Notably, the impact of this problem was appropriately downgraded to *Info* because attacks would be limited to phishing attempts or tricking the user in the current usage context.

Steps to reproduce:

1. Start OnionShare and launch the *receive* or *share* mode.
2. Visit the OnionShare address and append the following to the URL:
   http://kufbf2eh5wcxxxxxmht2toi246qaaedtdid.onion/ ***%0a%0d%0a%0d***
3. Observe mishandling of newlines within the *history* tab.

To mitigate this issue, Cure53 recommends stripping newlines from user-input before showing the URL in the *history* tab.

TTP-03-012 WP3: Lack of *root* detection and anti-debugging defenses *(Low)*

Testing confirmed that the current implementation failed to offer *root* detection and anti-debugging mechanisms. As stipulated in the *OWASP MASTG*[6] guidelines, it is paramount for every Android application to incorporate these features to enhance the overall effectiveness of the anti-tampering schemes, as well as to strengthen the mobile app's security resilience in general.

To mitigate this issue, Cure53 recommends incorporating a *root* detection library. With the revised protection, the applications would alert users running on rooted devices. Although this is not considered a comprehensive safeguard, the implementation would suffice toward informing users about the possible dangers associated with operating the app on rooted devices.

[6] https://github.com/OWASP/owasp-mastg

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

Conclusions

Cure53 generally concludes this January-February 2024 test of The Tor Project on a positive note. Multiple components of the Tor desktop applications and censorship circumvention tools were found to be hard-to-bypass and safe from major security flaws. As a sole exception, the OnionShare component has been proven to require more work, as it was associated with the most severe issues spotted during *TTP-03.* More broadly, it is hoped that the list of findings - which envelops eight vulnerabilities and four general weaknesses - can inform fine-tuned security measures for The Tor Project.

To reiterate, most of the censorship evasion tools consisted of clean, idiomatic Go code with good error handling. The parameters made the code easy to read and audit. *Lox* was an outlier here, instead opting for Rust. Still, this item retained high quality, without any use of *unsafe* being noted. OnionSprout was the only project that used a scripting language (Python), but it was judged as well-written, with no bugs to report.

As for the findings, Cure53 noted that the Tor Project has Sybil attack protections against the majority of the bridge distributor platforms. However, the Snowflake bridge distributor had no such protection (see TTP-03-001), which could allow attackers to cripple the network. The attack would rely on creating a large number of fake *snowflakes* or requesting to connect to large numbers of *snowflakes* without actually sending any data.

Next, *rdsys* also had a possible avenue for the Sybil attack depending on how the deployment is actually done. If it is used without a reverse proxy in front of it, an attacker could spoof the incoming IP of their requests to mount an effective Sybil attack on the platform. This could result in the entire bridge network being exposed. More care should be given to ensuring that all request body handlers for Go-based applications use body size limits to prevent DoS caused by request body excessive in size (TTP-03-002).

The Tor team provided a detailed list of commits and file review lists before the security assessment began. Additional commits were made available during the test, which proved extremely valuable in identifying critical areas of interest and sharpening the scope and its definition. This material allowed the testing team to swiftly familiarize themselves with all relevant features and changes, enabling them to plan and focus their efforts effectively.

With the aid of the list of commits, the testing team conducted a comprehensive code review, focusing on the relevant code changes. Specifically for WP1, no vulnerabilities were found, demonstrating a very good outcome for the included targets.

Regarding the client-side and UI components in scope, testers searched for *postMessage* issues, prototype-pollution, DOM XSS sinks, and similar input-manipulation issues but found none.

Attempts to access privileged pages (such as *about:preferences* and *about:torconnect*) and expand the attack surface were made through client-side and server-side redirects, as well as other features that allow opening pages and tabs. No significant findings were made.

The changes in the *captive* portal were analyzed, with additional checks conducted on the *redirection* functionality. It was determined that proper safeguards were implemented to prevent access to privileged pages and access was restricted accordingly. The usage of unsafe protocols appears properly blocked.

Similarly, several regex checks were implemented to prevent the usage of corrupted data for malicious bridges and proxies, which were thoroughly tested for potential issues. Attempts to corrupt the *torc* file saved to disk were fruitless.

The *Lox* browser implementation was reviewed, especially in terms of the *lox-wasm* library that is exposed to privileged pages in the browser. Dynamic testing was conducted against the exposed functions.

The Tor Browser is implemented as a series of patches over Mozilla Firefox. This helps to greatly reduce the attack surface of the Tor Browser, as it is built over a battle-tested browser with a strong security foundation. As such, the team gave a lot of attention to areas of the browser implementing new or custom features that are not present in the base Firefox deployment.

The Tor functionality of the browser was heavily scrutinized, as finding a way to bypass the Tor network and leak a user's IP address or identity would be a significant security issue. Common methods to leak IPs such as WebRTC were disabled, and the testing team was unable to find any routes to bypass the Tor circuit in this fashion.

Other browser features such as the new identity feature were also examined, with the security team attempting to use various JavaScript APIs to bypass the clearing of user data, but no issues were found either.

When examining the components in scope, Cure53 had access to the binaries of the Tor browser mobile application. These were perused via the shared build, while access to relevant sources was fostered by the generally accessible GitHub repository.

Before the discussion of findings, it should be noted that the mobile application has been mainly written with Kotlin, with which all of the code parts for the app are being handled. Cure53 therefore started their inspection with a look at the platform-specific implementations of the app. Afterwards, the testers continued the audit by investigating the code that is shipped in the app with a focus on the commits shared by the Tor Browser team.

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

The attack surface of the app was explored, with Cure53 focusing on how the Android application fit into its ecosystem. To that end, handling of communication between the app and the platform APIs was also checked. As it turned out, the application does not provide the best protections in this area. Specifically, possibilities of successful abuse through well-known attacks such as Android task hijacking were noted, as outlined in TTP-03-010.

In addition, the exported activities of the Android application miss out on proper input validation. This allows malicious applications installed on the device to start some kind of Denial-of-Service attacks against the vulnerable application. More details on this matter can be found in TTP-03-011. It was also noted that the application did not have any anti-debugging or root/jailbreak detection mechanisms. This significantly facilitated the debugging process, as highlighted in TTP-03-012.

While inspecting for occurrence of confidential data exposure, Cure53's analysis delved into the examination of unsafe data logging and storage practices. However, no issues have been identified in this realm.

Certain pivotal components within the application were deliberately not exported, effectively constraining the attack surface and proactively mitigating potential exploit avenues for malicious applications. The Cure53 team meticulously scrutinized the source code to pinpoint vulnerabilities that could potentially compromise access to these protected components, thereby creating a cascade effect of impactful vulnerabilities when exploited together. As of now, no issues of concern have been identified.

A comprehensive examination was conducted on the various modes of the OnionShare desktop app, aiming to identify vulnerabilities such as path traversals and web application issues, including Cross-Site Scripting, Server-Side Template Injection (SSTI), or CSP bypasses. The assessment of the generated websites within each mode yielded positive results, showcasing a commendable level of security that effectively mitigated common vulnerabilities.

Upon closer inspection of the generated web apps in the *receive* mode, it was observed that the application lacked proper enforcement of limits on the *text* values. This deficiency posed a potential risk of a Denial of Service (DoS) attack, as explained in TTP-03-005.

Cure53's meticulous analysis of the Python code confirmed secure handling of user-input. However, within this domain, a minor issue was identified in the *history* tab. This item could be susceptible to manipulation with newlines, as detailed in TTP-03-007.

Overall, OnionShare is a well-written Python web application with effective protections against most types of attacks related to file sharing and website functionality. It boasts robust protections against all forms of local file inclusion (LFI) attacks and no significant vulnerabilities were identified in either of the two services.

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

Nevertheless, more care should be taken to ensure that the chat service is protected against various methods of session manipulation that resulted in possible impersonation attacks (see TTP-03-006, TTP-03-008 and TTP-03-009) as well as problems that can be deemed as posing silent surveillance attack risks (see TTP-03-006 and TTP-03-008).

Most vulnerabilities were brought on by incorrect assumptions on how the flask *socket.io* integration works, such as the assumption that the *disconnect* event can only be triggered when a client's session has been destroyed (TTP-03-006) or that errors would cause the *connect* handler to cancel the connection (TTP-03-008). Username validation was also insufficient. Through the use of unicode control characters, it was possible to impersonate other users (TTP-03-009). In general, explicit error handling should be used when events are received to prevent cases where an event handler executes halfway before erroring out on a certain line and failing to complete a critical task.

The overall security posture of the Tor Browser mobile app received a favorable rating, with only minor issues identified during this Cure53 assessment. The app's security measures contribute to an elevated security posture of the mobile application on the whole.

In sum, the analysis - as conducted by Cure53 in early 2024 - generally revealed high-quality code being used throughout the application, with clear and informative comments accompanying key functionalities. Secure coding practices and input validation/sanitization are consistently implemented, effectively mitigating common security vulnerabilities. Addressing all of the reported findings, however, remains crucial for further strengthening the app's security.

The Cure53 team's overall evaluation of the Tor Browser mobile app's security is positive. The application demonstrated a strong foundation with no major vulnerabilities detected in the frames of *TTP-03*. Resolving the highlighted minor flaws can further enhance the application's security and user-protection.

Cure53 would like to thank Gaba, Shelikhoo, Cecylia, Micah and Richard from the Tor Project team for their excellent project coordination, support and assistance, both before and during this assignment.