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

Pentest-Report TOR Browser & OONI 02.-03.2023

Cure53, Dr.-Ing. M. Heiderich, J. Larsson, M. Elrod, M. Pedhapati, MSc. H. Moesl, P. Einkemmer, Dipl.-Ing. D. Gstir, R. Weinberger

Index

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

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 offers a comprehensive overview of the coverage, findings, and conclusions of a crystal-box penetration test and source code audit against a multitude of Tor Project aspects and components relating to censorship bypasses. Specific target areas include the OONI Probe desktop UI; rdsys software; BridgeDB software; Conjure implementation; building infrastructure, and specific Tor Browser alterations.

This widespread audit was requested by The Tor Project in November 2022 and initiated by Cure53 in February and March 2023, namely between CW07 to CW11. To fulfill this engagement's expected coverage levels, an seventy-two-day time frame was allocated for testing.

All assessment items were grouped into six distinct work packages (WPs), which read as follows:

- **WP1**: Crystal-box penetration tests & code audits against OONI Probe desktop UI
- **WP2**: Crystal-box penetration tests & code audits against rdsys software
- **WP3**: Crystal-box penetration tests & code audits against BridgeDB software
- **WP4**: Crystal-box penetration tests & code audits against Conjure implementation
- **WP5**: Crystal-box penetration tests & code audits against building infrastructure
- **WP6**: Crystal-box penetration tests & code audits against Tor Browser alterations

To conduct the preparation, assessment, and finalization phases for these WPs, a team comprising eight skill matched senior testers was established. These efforts were facilitated by the provision of sources, detailed documentation, URLs, key focus areas, and any other access item required. Preparatory measures were also implemented in CW06 February 2023 to enable a hindrance-free test period.

A dedicated, shared Signal group was created for communication between the Tor Project and Cure53 teams, with all involved employees from both organizations invited to participate. This platform, as well as the excellent preliminary scope handling, allowed for smooth discussions and ensured no notable roadblocks were encountered during the test.

Cure53 gave frequent status updates about the review and connected findings. Live reporting was offered and subsequently implemented for some pertinent issues.

Onto the findings, the test team's strong coverage across the six WPs raised a total of nineteen. A low proportion of three were deemed security vulnerabilities; the remaining sixteen incur little exploitation potential and were hence categorized as miscellaneous.

In context, the scope and predetermined time frame for this project was generous, which raised the expectancy of an extensive volume of discoveries. However, Cure53 positively noted that, contrary to this initial expectation, only a minimal number of actual security vulnerabilities were detected. The vast majority of issues rather pertain to general weaknesses and hardening guidance; implementing these should be relatively straightforward for the developer team.

Despite the low yield of exploitable vulnerabilities, Cure53 would like to underline the pressing importance of two assigned a *High* severity rating. These describe a couple of absent configurations: the first relating to authentication, and the second concerning a signature within the bridge list, as stipulated in tickets TTP-01-008 and TTP-01-009 respectively. Naturally, one can strongly advise mitigating these as soon as possible to ensure the platform's commendable resilience can be maintained.

All in all, Cure53 concludes this audit with a favorable impression of the inspected Tor Project aspects and components, which proved sufficiently robust and hardened against a multitude of common threats and attack vectors. Nevertheless, the number of tickets attests to the abundant opportunities for security upgrades. The Tor Project team should invest adequate resources into initiating follow-up actions and systematically resolving each finding in priority order of severity.

The report will now offer concise information concerning the scope, and, available materials, before extrapolating the *Test Methodology*. Here, Cure53 gives a detailed overview of all test procedures initiated and the ensuing coverage achieved. Next, all findings are listed in descending order of detection - first the *Identified Vulnerabilities*, then the *Miscellaneous Issues*. These are accompanied with a technical explanation, a Proof-of-Concept (PoC) where appropriate, and the recommended advice for mitigation.

Finally, the *Conclusions* section verifies Cure53's assessment of the myriad Tor Project components in scope with a definitive overview of the perceived security posture offered by the assessed features, in relation to censorship bypasses.

Scope

- **Pentests & code audits against multiple Tor Project software & components:**
  - **WP1**: Crystal-box pentests & code audits against OONI Probe desktop UI
    - **Main repositories:**
      - https://github.com/ooni/probe-desktop/
      - https://github.com/ooni/probe-engine
    - **Pull requests:**
      - https://github.com/ooni/probe-desktop/pull/88
      - https://github.com/ooni/probe-desktop/pull/95
    - **GitHub issues:**
      - https://github.com/ooni/probe-engine
      - https://github.com/ooni/probe-engine/issues/89
      - https://github.com/ooni/probe-engine/issues/90
      - https://github.com/ooni/probe-engine/pull/179
      - https://github.com/ooni/probe-engine/pull/180
      - https://github.com/ooni/probe-engine/pull/143
    - **Documentation:**
      - https://github.com/ooni/probe-desktop/wiki/Manual-Testing
  - **WP2**: Crystal-box penetration tests & code audits against rdsys software
    - **Sources:**
      - https://gitlab.torproject.org/tpo/anti-censorship/rdsys
    - **Documentation:**
      - https://gitlab.torproject.org/tpo/anti-censorship/rdsys/-/tree/main/doc
  - **WP3**: Crystal-box penetration tests & code audits against BridgeDB software
    - **Main repository:**
      - https://gitlab.torproject.org/tpo/anti-censorship/bridgedb
    - **GitLab issues:**
      - https://gitlab.torproject.org/tpo/anti-censorship/bridgedb/-/issues/34322
      - https://gitlab.torproject.org/tpo/anti-censorship/bridgedb/-/issues/40051
      - https://gitlab.torproject.org/tpo/anti-censorship/bridgedb/-/issues/40057
      - https://gitlab.torproject.org/tpo/anti-censorship/bridgedb/-/issues/40050
      - https://gitlab.torproject.org/tpo/anti-censorship/bridgedb/-/issues/40037
    - **Documentation:**
      - https://gitlab.torproject.org/tpo/anti-censorship/bridgedb/-/tree/main/doc
    - **URL:**
      - https://bridges.torproject.org/
    - **Scope:**
      - Only the management of *bridges.torproject.org*

- **WP4**: Crystal-box pentests & code audits against Conjure implementation
  - **Main repository:**
    - https://gitlab.torproject.org/tpo/anti-censorship/pluggable-transports/conjure/
  - **Branch:**
    - main
  - **Commit:**
    - d64c96414427c27f8eae8b6484ec0dccb5b710fd
  - **Merge requests:**
    - https://gitlab.torproject.org/tpo/applications/tor-browser-build/-/merge_requests/618
    - https://gitlab.torproject.org/tpo/applications/tor-browser-build/-/merge_requests/632
    - https://gitlab.torproject.org/tpo/applications/tor-android-service/-/merge_requests/2
  - **GitLab issues:**
    - https://gitlab.torproject.org/tpo/applications/tor-browser/-/issues/41361
  - **Documentation:**
    - https://gitlab.torproject.org/tpo/anti-censorship/pluggable-transports/conjure/-/wikis/How-it-Works
    - https://gitlab.torproject.org/tpo/anti-censorship/pluggable-transports/conjure/
- **WP5**: Crystal-box penetration tests & code audits against building infrastructure
  - **Primary audit areas:**
    - The software build tools and deployment modified during this project:
    - The rdsys build and deployment.
    - The Conjure infrastructure.
    - The GitLab CI used for deploying *bridges.torproject.org*
    - The infrastructure that builds and releases Tor Browser
  - **Documentation:**
    - https://gitlab.torproject.org/tpo/tpa/team
- **WP6**: Crystal-box penetration tests & code audits against Tor Browser alterations
  - **Sources:**
    - https://gitlab.torproject.org/tpo/applications/tor-browser
  - **Documentation:**
    - https://gitlab.torproject.org/tpo/applications/tor-browser/-/wikis/home
    - https://gitlab.torproject.org/tpo/applications/team/-/wikis/Development-Information/Tor-Browser/Building
  - **GitLab milestones:**
    - https://gitlab.torproject.org/groups/tpo/-/milestones/29
  - **GitLab issues:**
    - https://gitlab.torproject.org/tpo/applications/tor-browser/-/issues/40477
    - https://gitlab.torproject.org/tpo/applications/tor-browser/-/issues/27476

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

  - https://gitlab.torproject.org/tpo/applications/tor-browser/-/issues/34345
  - https://gitlab.torproject.org/tpo/applications/tor-browser/-/issues/40568
  - https://gitlab.torproject.org/tpo/applications/tor-browser/-/issues/40597
  - https://gitlab.torproject.org/tpo/applications/tor-browser/-/issues/40774
  - https://gitlab.torproject.org/tpo/applications/tor-browser/-/issues/40448
  - https://gitlab.torproject.org/tpo/applications/tor-browser/-/issues/40446
  - https://gitlab.torproject.org/tpo/applications/tor-browser/-/issues/40445
  - https://gitlab.torproject.org/tpo/applications/tor-browser/-/issues/40444
  - https://gitlab.torproject.org/tpo/applications/tor-browser/-/issues/40444
  - https://gitlab.torproject.org/tpo/applications/tor-browser/-/issues/40469
  - https://gitlab.torproject.org/tpo/applications/tor-browser/-/issues/41058
  - https://gitlab.torproject.org/tpo/applications/tor-browser/-/issues/40597
  - https://gitlab.torproject.org/tpo/applications/tor-browser/-/issues/40807
  - https://gitlab.torproject.org/tpo/applications/tor-browser/-/issues/31286
  - https://gitlab.torproject.org/tpo/applications/tor-browser/-/issues/27476
- **Focus areas:**
  - Examination of automatic censorship circumvention
  - Examination of automatic bootstrapping of bridge-list
  - Examination of new circumvention mechanisms
- **Scope:**
  - Only the censorship circumvention mechanisms.

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

Test Methodology

This section details the metrics and methodologies utilized to evaluate the security characteristics of the TOR Browser project and codebase. The results achieved against pertinent, individual areas of the project's security properties are discussed, which were either selected by Cure53 or pinpointed by other participatory parties for closer inspection.

Further clarification concerning the characteristics that were deep-dive assessed during this project is offered, particularly considering the small volume of flaws assigned a *High* severity rating.

In this assessment, several components of the Tor browser ecosystem were designated as key examination targets. Cure53 conducted an extensive source code analysis across the varying components of the Tor Browser software stack, as well as the individual projects declared in-scope. Whilst Cure53's overarching aim was to achieve widespread coverage across the targets, extensive audits of this kind are always limited by the budget allocation. As such, they require selectivity and isolated focal points, particularly for the code areas deemed most sensitive. In response, the Tor team provided a well-defined scope document with a list of focus areas, which helped Cure53 to outline a clear audit strategy. The following section pertains to the *Scope of Work*, as defined by Tor.

Scope of Work

The Tor Browser is a web browser that is specifically designed to protect its users' privacy and anonymity while browsing the internet. It is based on the Firefox browser and offers pre-configured settings and additional add-ons that allow users to connect to the Tor network, which routes their internet traffic through a series of relays or bridges. This renders tracing the user's online activity back to their physical location or identity significantly more difficult to achieve. In practice, the Tor Browser is often used to circumvent censorship by connecting to Tor bridge relays ("bridges"), comprising Tor relays that are not listed in the public Tor directory and are thus less susceptible to censorship.

The primary focus of this review related to the following areas, as defined by Tor:

- Evaluation of improvements regarding the methods by which users connect to bridges in the Tor Browser

This review comprised various components within the Tor Browser ecosystem, as detailed below:

- **OONI Probe**: OONI Probe is a technology used for identifying censorship events. During this project, the OONI team integrated Tor tests into their measurement kit, expanded OONI's methodologies to test protocol-based blocking, and measured the performance and blocking of other circumvention tools.

- **Rdsys**: Rdsys, which stands for *Resource Distribution System*, is utilized to distribute censorship circumvention proxies and related resources, such as download links, to individuals that are experiencing censorship. These resources are provided to censored users via various distribution methods and equip them with circumvention proxies to ensure they can access the open internet.

- **BridgeDB**: BridgeDB is a component that communicates with rdsys to retrieve the list of bridges that are distributed.

- **Conjure**: Conjure is a refraction networking system that routes traffic to endpoints in an ISP's unused IP address space. Conjure is integrated into Tor as a pluggable transport.

- **Review of the building infrastructure**: A review of the software's build tools and deployment was initiated via the following actions:

  - Rdsys build and deployment.
  - Conjure infrastructure.
  - GitLab CI.
  - Infrastructure that builds and releases Tor Browser.

- **Tor Browser alterations and customizations**: Here, Cure53 scrutinized a curated list of pull / merge requests declared relevant for this audit.

From a technology perspective, the majority of the source code repositories examined were written in JavaScript and Golang.

As communicated by the Tor team, the following components and attack vectors were considered out-of-scope:

- An examination of the Firefox codebase in general. For example, typical attack scenarios for browsers focusing on memory corruption vulnerabilities present in the code that renders web content, as well as those in the sandbox interfaces and other relevant areas, were not subjected to review.

- Passive attackers were also deemed outside this project's scope. This includes entities such as censors who simply observe the traffic during the bootstrapping phase of their automatic censorship circumvention efforts.

WP1: Penetration tests & code audits against OONI Probe desktop UI

The following section provides a list of tasks undertaken during this security project in relation to the OONI Probe desktop application. All coverage achieved by the testing team using white-box methods during the analysis of various endpoints, Electron configurations, frontend code, ElectronJS code, the binaries, and alternative, security-related aspects are subsequently offered below:

- For the frontend characteristics of the OONI Probe desktop application, the testing team instigated a plethora of advanced approaches. Firstly, given that the NextJS is currently utilized for the frontend (which uses React), the testing team deemed it apt to focus on potential security issues related to this framework during the initial stage of this assessment. The shared repositories were subjected to deep-dive assessment to determine erroneous usage of *dangerouslySetInnerHTML*, since this is often overused and introduces XSS issues.

- Due to the fact that the ReactJS framework does not handle URLs assigned to the *href* property of the HTML *anchor* tags, Cure53 probed the source code for any instances of this nature. Ultimately, these efforts proved unfruitful. Furthermore, since the NextJS *next/router* module allows navigation to JavaScript protocol URL, the code was searched for any erroneous usage of the router.

- Subsequently, the provided source code was audited for DOM XSS-related issues, with reference to any potential usage of *location.href*, *window.open*, or user-controlled URL parameters. The latter was subjected to further scrutiny to determine the potential for prototype pollution or client-side path traversal. Positively, Cure53 could not unveil any notable findings in this area, despite strenuous endeavors. Furthermore, the JSON prettifier module was audited, though this similarly yielded a lack of outcomes.

- Next, Cure53 concentrated its efforts on the OONI Probe desktop application's ElectronJS.

- Initially, the application was investigated for typical ElectronJS security recommendations[1]. These generally pertain to stricter *BrowserWindow* web preferences, suboptimal use of security features, and a lack of incorporating the Electron security features' inherent benefits. In this respect, testing confirmed that the OONI Probe desktop application fails to integrate some fundamental recommended security features. Aside from an additional defense-in-depth recommendation (see TTP-01-004), no severe issues were identified in the application.

- In addition, an explicit focus was placed on any features or function calls that could facilitate arbitrary code execution on the client-side. The usage of *shell.openExternal* and *shell.openPath* was thoroughly tested for any misconfigurations, though the test team could not locate any exploitable issues related to these misconfigurations. This primarily owes to the lack of sources from user-controlled input. However, a beneficial measure was suggested to prevent further exploitability of these issues, as stipulated in ticket TTP-01-002.

- Since deep links are often considered a prime vector for Electron vulnerabilities, the deep link handlers were audited for any potential security flaws. Fortunately, no usage of *open-url* was identified, which serves to remove the deep-link source from the application.

- Furthermore, the IPC communication between the renderer and main processes was evaluated from a security perspective, though the team confirmed that user input is not reachable via these sinks.

- Finally, the OONI Probe desktop application's specific code was scrutinized to ensure coding best practices were adhered to, as well as identify any logical weaknesses. For this purpose, the components related to the integration of OONI probe-cli with the desktop application and the OONI probe-cli were estimated. Similarly, these efforts yielded a lack of notable findings.

[1] https://github.com/electron/electron/blob/main/docs/tutorial/security.md

WP2: Penetration tests & code audits against rdsys software

The following passages provide an exhaustive list of actions undertaken against the rdsys software during this security project. All coverage achieved by the testing team using white-box methods during the analysis of various backend endpoints, frontend distributor endpoints, core logic, ipc mechanism, and other security-related configurations are discussed next:

- Firstly, the testing team reviewed the the rdsys software's available documentation[2] to quickly grasp the attack surface and any likely attack vectors.

- The rdsys is written in Golang; as such, particular scrutiny was placed on unearthing any typical programming errors for Golang, including common attack scenarios that may incur vulnerabilities such as Denial-of-Service (DoS) situations, Remote Code Execution (RCE), usage of outdated Golang modules, absent input validations, and privilege escalation.

- Subsequently, the backend APIs were inspected in an attempt to enumerate any general web vulnerabilities. The implemented authentication and authorization mechanisms were assessed for common bugs, vulnerabilities, and misconfigurations. One issue was identified in this regard, as detailed in ticket TTP-01-008.

- Furthermore, alternative endpoints - such as those related to resource, resource stream, and target - were deemed adequately safeguarded against common web security errors.

- Elsewhere, Cure53 rigorously analyzed the distributors, including the salmon, gettor, telegram, and others.

- For salmon specifically, the access control and authentication processes concerning invite, account, redeem, and proxies were thoroughly tested. The test team also attempted to enumerate the methods by which one could affect the innocence calculation, though no notable issues were identified in this area.

- Next, the aforementioned moat, gettor, and telegram distributors were subjected to in-depth investigation. Here, a minor issue was observed in relation to the lack of authentication for the *https* distributor, though this facilitates negligible security impact.

[2] https://gitlab.torproject.org/tpo/anti-censorship/rdsys/-/tree/main/doc

- Additionally, all inspections against the backend and distributor web server configurations, rdsys frontend apache configuration, and email client configurations utilized for gettor proved unsuccessful in locating any security vulnerabilities.

- Finally, following a comprehensive analysis of JSON parsing, Cure53 concluded that any potential vulnerabilities or logical bugs in the rdsys software's JSON parsing functionality had been effectively negated.

WP3: Penetration tests & code audits against BridgeDB software

The paragraphs offered below pertain to all BridgeDB source code repository coverage areas assessed by Cure53. This serves to provide a transparent overview of the characteristics granted deep-dive analysis by the test team during this assessment.

- BridgeDB comprises a collection of backend servers utilized to distribute bridges to clients. BridgeDB offers an HTTPS interface, email responder, SQLite database, and MOAT interface for this purpose, all of which are written in Python.

- The BridgeDB dependencies benefited from a substantial manual code review, including a package dependency analysis. Here, a few outdated dependencies were identified, as further explained in ticket TTP-01-003.

- During the code review, the test team noted that a crypto module leveraged by BridgeDB was vulnerable and susceptible to well-known attack scenarios, due to the fact that the module in question is no longer actively maintained. Supplementary guidance on this area of concern is offered in ticket TTP-01-010.

- The code was analyzed in relation to commonly-encountered vulnerabilities such as SQL injection, absent input validation, usage of insecure primitives, and general misconfigurations. These efforts unearthed a couple of notable issues, as stipulated in tickets TTP-01-014 and TTP-01-015.

- Finally, the BridgeDB hashring implementation was heavily studied, though no associated issues were identified in this regard.

WP4: Penetration tests & code audits against Conjure implementation

The advanced penetration techniques and ensuing coverage observed during testing against the Conjure component and software compound specifically are discussed in greater detail below.

- The available documentation was extensively studied to obtain a clear overview of the software compound, architecture, and methods of application by the Tor browser. This served to pinpoint any problematic areas and potential attack surfaces ahead of the active review phase.

- The Conjure implementation's code base, written in Go, benefited from a comprehensive manual code review and static code analysis using renowned tools such as gosec[3]. These endeavors could only unveil one issue in this area, which is described in ticket TTP-01-016. Generally speaking, Go's facilitation of a higher degree of memory safety in comparison with other languages that compile to native code is irrefutable. One will rarely encounter direct memory safety issues concerning Go, which repels erroneous behaviors typically found in C and C++. The myriad benefits offered by a language such as Go render it highly recommendable for these purposes. As such, Cure53 was not surprised that it was selected as the language of choice for implementing Conjure.

- Furthermore, the catalog of merge requests pertaining to Conjure were subjected to meticulous review. Upon closer inspection, Cure53 confirmed that the majority of the merge requests within the scope of this review were linked to build-related modifications, which did not evoke any security concerns in this regard.

- A comprehensive inventory of outdated and vulnerable dependencies was generated by scrutinizing the dependencies of the Conjure implementation with the assistance of *govulncheck*. This in-depth process served to uncover any potential security weaknesses or outdated components that may negatively impact the system's stability and reliability. In light of this, a few outdated dependencies were identified, as stipulated in ticket TTP-01-007.

- The primary purpose of the implementation is to provide a connection between *gotapdance's* Conjure package and *goptlib*'s pluggable transport (PT) functionality. Specifically, Conjure facilitates a phantom proxy registration process that establishes PT proxies on both the client and server side.

[3] https://github.com/securego/gosec

- For this assessment, the Tor team deliberately decided to exclude *goptlib* and *gotapdance* from the scope, given that amendments were not required for these repositories. However, to clearly understand the pluggable transport concept employed by *goptlib*, a brief review of this component was conducted. Based on this assessment, the test team verified that *goptlib* conforms to the PT specification[4], with no relevant issues identified in this respect.

WP5: Penetration tests & code audits against building infrastructure

This section aims to provide a comprehensive overview of the tasks completed by Cure53 during its WP5 review against the building infrastructure. Coverage and findings concerning the build process and infrastructure are extrapolated next.

- The Cure53 team members commenced the WP5 review by familiarizing themselves with the different components in question. The relevant build systems and setup processes for the varying features were inspected, followed by scrutiny of the continuous integration systems and deployed mechanisms.

- The key intention for this work package was to enumerate any build-system-related vulnerabilities. Henceforth, each specific step of the building process was checked for the presence of any undesired triggerable side effects.

- One particular attack scenario granted due consideration represented the ability for malicious build loads to compromise the build systems and CI systems, then subsequently detect any means of altering and persisting inside the build system.

- Another potential vector referred to the potential for malicious builds to exfiltrate secrets and access tokens from the build system. Here, Cure53 sought to validate whether the attack surface could be reduced and associated risks categorically mitigated. No significant findings were encountered in this respect.

- The next attack vector evaluated during this review pertained to a compromised build system, consequential integrity compromisation, and whether any defensive precautions had been taken to counteract this. Naturally, protecting against this particular attack scenario remains challenging. However, Tor's effort to leverage reproducible builds - as well as build signing and building on two machines in parallel - had a significantly positive impact against this threat. With this in mind, a minor vulnerability allowing for privilege escalation from *nobody* to *builduser* was deemed exploitable during the building process; see ticket TTP-01-005 for additional information.

[4] https://gitweb.torproject.org/torspec.git/tree/pt-spec.txt

- Deploy scripts were honed in on for all components that rendered deploy scripts available, including rdsys, to unveil any side effects that could be exploited during the deployment by either a local or remote attacker. Next, Cure53 strove to validate whether any overly permissive configurations would persist after deployment, for which the primary subsequent finding is detailed in ticket TTP-01-001.

- To complement the build and deployment process reviews, several classic supply chain issues were analyzed. In this process, the dependencies of the differing components were inspected to verify whether the dependencies were up to date or if the software relied on outdated libraries and components. This examination raised the presence of two flaws, as detailed in TTP-01-003 and TTP-01-006.

- Finally, the presence and effectiveness of any automated tooling to retain the latest software dependencies was checked. The presence of the two tickets above can be attributed to a lack of respective automatisation.

WP6: Penetration tests & code audits against Tor Browser alterations

The following items of description detail the tasks completed by the test team against WP6 specifically, including an analysis of the Tor Browser code alterations and generic assessment of the censorship circumvention feature. The ensuing degree of coverage concerning the security offering exhibited by the integrations and surrounding components is further elaborated.

- The available documentation was extensively studied to obtain a clearer overview of the software compound, architecture, and automatic censorship circumvention feature comprising the Tor Browser. This area was pre-defined as a primary scope item and was therefore examined with utmost scrutiny. This preliminary perusal of all available documentation allowed the test team to identify any problematic areas and potential attack surfaces. Cross-organization discussions were also held to further evaluate the underlying threat model, which helped the test team to gain an optimal understanding of viable threat actors that should be taken into account during this review.

- The Tor team shared a list of merge requests, which involved incorporating a censorship circumvention feature into Tor. Upon conducting a thorough analysis of this list, the observation was made that the majority of the modifications pertained to the user interface, rendering the review unproductive in identifying the fundamental components that constituted the censorship circumvention feature.

- A general audit of the relevant folders and files related to censorship circumvention was undertaken to isolate potential security weaknesses such as insecure input handling, general lack of input validations, and auxiliary attack vectors instigatable from a censor perspective.

- A test setup was installed based on an official tutorial[5] provided by the Tor team. The installation of the test setup provided a controlled environment to facilitate Tor component modification and experimentation, allowing for greater flexibility in testing the censorship circumvention feature. Furthermore, a Man-in-the-Middle (MitM) environment was created that enabled inspection of the packets sent to the rdsys / BridgeDB instance. This process offered valuable insight into the underlying protocol used for domain fronting. Despite these actions, no notable findings were reported.

- Upon request, the Tor team provided directions on the critical source files that were associated with censorship circumvention. These particular source files were rigorously surveyed based on the most up-to-date revision available at the time. This ensured that the source code adhered to best practices for maintaining the integrity and security of the censorship circumvention feature, as well as its resilience against censors. The source files considered most pertinent represent the following:

    - *browser/modules/Moat.jsm*
    - *browser/modules/TorConnect.jsm*
    - *browser/modules/TorSettings.jsm*
    - *browser/modules/BridgeDB.jsm*

- Last but not least, the code was probed in an attempt to identify and address any common JavaScript vulnerabilities that could potentially be exploited by a censor with access to a rdsys / BridgeDB instance. This may enable the censor to send packets containing malicious content, such as a manipulated CAPTCHA image or QR code, resulting in a Cross-Site Scripting (XSS) attack on the Tor client. The potential for logic errors and API calls was stringently vetted, which raised the presence of the issues documented in tickets TTP-01-009, TTP-01-012, and TTP-01-017.

[5] https://gitlab.torproject.org/tpo/applications/tor-browser/-/wikis/Hacking#building-just-firefox

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, all tickets are given a unique identifier (e.g., *TTP-01-001*) to facilitate any future follow-up correspondence.

TTP-01-001 WP5: Privilege escalation from *nobody* to *rdsys* in deploy script *(Medium)*

During the analysis of the rdsys project, a local privilege escalation vulnerability was detected in a deploy script that allows an attacker with rdsys server write privileges to gain the privileges of the *rdsys* daemon user. To retrieve these privileges, the attacker would need to wait for a deploy process to initiate then hijack the deployment process, which can be achieved by exploiting a narrow time frame (approximately a few seconds) in which the binary is world writable. An attacker could overwrite and possibly backdoor the latter via *rdsys* executed binary.

Affected code:
https://gitlab.torproject.org/tpo/anti-censorship/team/-/wikis/Survival-Guides/Rdsys-Survival-Guide#deploying-a-new-version

Affected code:
```
#!/bin/bash
[...]
path="$1"
executable=$(basename "$path")

scp "$path" POLYANTHUM:/tmp
ssh -t POLYANTHUM \
    "chmod 777 /tmp/${executable} && " \
    "sudo -u rdsys bash -i -c '" \
      "systemctl --user stop rdsys-backend && " \
      "cp /tmp/${executable} /home/rdsys/bin/rdsys-backend && " \
      "systemctl --user start rdsys-backend' && " \
    "rm -f /tmp/${executable}"
```

PoC:
```
#!/usr/bin/env bash
# replace "executable" with name of binary
FILE="/tmp/executable"

while true
do
    if test -f "${FILE}"
    then
```

```
        echo 'evil payload' > ${FILE}
        echo 'done'
        exit
    fi
done
```

To mitigate this issue, Cure53 advises leveraging stricter Unix file privileges than those used after the binary is placed in the */tmp* folder. Alternatively, retracting usage of a world read and writable folder to temporarily store the binary would also mitigate the present issue.

TTP-01-008 WP2: Lack of resource registration authentication *(High)*

Whilst assessing the rdsys source code, the observation was made that the rdsys backend lacks authentication for the resource registration endpoint. This allows an adversary to register arbitrary malicious resources for distribution to users.

The following snippet underlines the affected code, whereby the resource registration endpoint does not offer any form of authentication check, as compared to other comparable endpoints.

Affected file:
rdsys/internal/backend.go

Affected code:
```
func (b *BackendContext) resourcesHandler(w http.ResponseWriter, r
*http.Request) {

       switch r.Method {
  [...]
       case http.MethodPost:
              if r.URL.Path == b.Config.Backend.ResourcesEndpoint {
                     b.postResourcesHandler(w, r)
       [...]
}
func (b *BackendContext) postResourcesHandler(w http.ResponseWriter, req
*http.Request) {

       body, err := ioutil.ReadAll(req.Body)
       [...]
       rTypes := map[string]struct{}{}
       for _, r := range rs {
              b.Resources.Add(r)
              rTypes[r.Type()] = struct{}{}
              log.Printf("Added %s's %q resource to collection.",
req.RemoteAddr, r.Type())
```

```
        }

        for rType := range rTypes {
                b.rStore.Save(rType)
        }

  [...]
}
```

The aforementioned issue can be reproduced by executing the following cURL request.

PoC:

```
 curl http://localhost:7100/resources  -i --data '[{"type": "obfs2", "address":
"1.2.3.2", "port": 1235,
"fingerprint":"10282810115283F99ADE5CFE42D49644F45D715D"}]' -XPOST

# Other endpoints which are missing authentication
curl -i -s -k -X $'GET' \
    -H $'Host: localhost:7100' \
    $'http://localhost:7100/status?id=10282810115283F99ADE5CFE42D49644F45D715D'

curl -i -s -k -X $'GET' \
    -H $'Host: localhost:7100' \
    $'http://localhost:7100/rdsys-backend-metrics'
```

To mitigate this issue, Cure53 strongly advises implementing robust authentication mechanisms for all endpoints, with particular consideration for resource registration. This will help to ensure that only authorized users are able to register resources, thereby reducing the risk of unauthorized access.

TTP-01-009 WP6: Bridge list lacks signature *(High)*

During a source code review of the Tor Browser's censorship circumvention feature, the discovery was made that the Tor Browser obtains a list of bridges from rdsys / BridgeDB via an API interface entitled *MoatRPC*, which supports several RPC commands. The received list from rdsys / BridgeDB contains various bridge notes, along with the supported protocol and a hash of the bridge certificate.

Due to the fact that this list is returned to the Tor Browser prior to Tor network connectivity, a malicious censor may tamper with the bridge list to force the user into connecting to a censor-controlled bridge instance, for example. An attack of this nature is considered plausible since the bridge list is not cryptographically signed via usage of a private key, whereby the corresponding public key is baked into Tor and subsequently leveraged for verification.

Notably, this issue can only be exploited by malicious actors with elevated access; for instance, those that are able to eavesdrop on this connection or hold access to the server (rdsys / BridgeDB) providing the bridge list.

As deducible in the code snippet below, a list of bridges is received from the rdsys / BridgeDB instance and the JSON result is stored directly in TorSettings without performing any signature check.

Affected file:
browser/modules/Torconnect.jsm

Affected code:
```
if (settings?.settings && settings.settings.length) {
  this.settings = settings.settings;
} else {
  try {
    this.settings = await this.mrpc.circumvention_defaults([
      ...TorBuiltinBridgeTypes,
      "vanilla",
    ]);
  } catch (err) {
    [...]

    );
  }
}

[...]

for (const [
    index,
    currentSetting,
  ] of this.settings.entries()) {
    if (this.transitioning) {
      break;
    }
    [...]
    TorSettings.setSettings(currentSetting);

    [...]
}
```

To mitigate this issue, Cure53 recommends incorporating a cryptographic signature into the JSON response containing the bridge list results, which would verify that the bridge list has been indeed provided by Tor and remains untampered.

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-01-002 WP1: Unsanitized *shell.openExternal* usage *(Info)*

Whilst auditing the source code for dangerous sinks in the Electron application that may facilitate RCE, the test team noted that *shell.openExternal* was utilized without any argument sanitization passed through the function. This behavior can incur grave consequences in the event user-controlled input passes through these sinks.

Nonetheless, since no user-controlled input is passed through these functions, this vulnerability could not be tangibly exploited at the time of testing. However, if the code is subjected to alterations in the future and user-input is passed, or OONI Probe Engine results are adversary controlled, this issue would become exploitable.

The following snippet verifies the affected code.

Affected file:
renderer/components/utils.js

Affected source:
```
export const openInBrowser = (url, event) => {
  var shell = require('electron').shell
  event.preventDefault()
  shell.openExternal(url)
}
```

To mitigate this issue, Cure53 advises adopting an allow-list of protocols, such as *http* and *https*, to be passed through the *shell.openExternal* sink. By doing so, code execution will be prevented even if unexpected user input is passed through this function.

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

TTP-01-003 WP3: BridgeDB requires outdated packages with known CVEs *(Low)*

During the BridgeDB security assessment, the observation was made that several software packages leveraged outdated versions that are vulnerable to a host of security risks. The following software packages were identified as out-of-date and potentially insecure. Notably, the version information provided is based on data collected at the time of testing. Whether these vulnerabilities are exploitable or not depends on the relevant functionality usage in the targeted application. Please note: this ticket should not be deemed to incur application risk but serves to highlight the lack of supply chain security automation.

Affected files:

- *bridgedb/requirements.txt*
- *bridgedb/.test.requirements.txt*
- *bridgedb/.travis.requirements.txt*

Selected vulnerable packages:

CVE ID	Component	Installed version
CVE-2022-21712 CVE-2022-21716 CVE-2022-24801 CVE-2022-39348	*Twisted*	21.7.0
CVE-2021-32837	*Mechanize*	0.4.5
CVE-2021-34552 CVE-2022-22817 CVE-2022-24303 [...]	*Pillow*	8.2.0
CVE-2023-25577	*Werkzeug*	2.2.2

Notably, the testing team was unable to comprehensively prove any potential impact during the limited time frame granted for this review. As such, implications remain unknown at this point and should be further researched by the developer team.

Generally speaking, the provision of robust supply chain security can be considerably challenging. Quite often, an easy or comprehensive solution simply cannot be offered, and the results of the selected protection framework can entirely depend on the integrated version of the deployed libraries.

To mitigate the present issues to the highest degree, Cure53 recommends upgrading all affected libraries and establishing a policy to ensure libraries remain up-to-date moving forward. This will help the aspects to benefit from patches rolled out for previously-detected weaknesses across different solutions. For this purpose, some package managers offer a functionality to automate package updates to non-vulnerable versions. Note, however, that the degree of protection may vary; up-to-date retention typically becomes more difficult to achieve with additional third-party libraries in play.

Under certain circumstances, one may have to resort to either sending PRs to the library maintainer or even forking the library. The Tor Project could consider assigning a developer as the task owner to ensure this issue is not relegated to the backlog. Lastly, replacing certain libraries with actively-maintained alternatives may become a necessity over time.

TTP-01-004 WP1: Electron application best-practice implementation *(Info)*

The observation was made that the OONI Probe desktop application lacks general Electron application security recommendations. These do not directly incur security vulnerabilities in isolation, though may prove useful for attackers to exploit other areas of weakness with greater ease. The following list enumerates all issues that require reviewing and subsequent mitigation:

- **Disable Node.js integration[6]:** If this is not disabled, an attacker can use any Node.js feature simply by utilizing the *require()* function and achieving RCE via that call. To disable this, set the *nodeIntegration* property to *false* in the *BrowserWindow* constructor's argument.
- **Enable context isolation[7]:** If this remains disabled, a web page's JavaScript can affect the execution of the Electron's internal JavaScript code on the renderer and preload scripts. Since the Electron's internal code and preload scripts retain access to Node.js features, in the worst-case scenario an attacker can perform RCE by accessing powerful features via a specifically-crafted JavaScript code on the web page. To enable this, set the *contextIsolation* property to *true* in the BrowserWindow constructor's argument.
- **Enable sandbox[8]:** This limits access to most system resources and hence mitigates any damage incurred by malicious code. This is considered an important facet toward hindering an attacker's opportunities in the eventuality the renderer is compromised. Without the sandbox, arbitrary code execution can be achieved via publicly-known Chromium bugs in the event an attacker is able to execute arbitrary JavaScript inside the renderer. To enable sandbox mode for all

[6] https://www.electronjs.org/docs/latest/tutorial/security#2-do-not-enable-nodejs-[...]-content
[7] https://www.electronjs.org/docs/latest/tutorial/context-isolation
[8] https://www.electronjs.org/docs/latest/tutorial/sandbox

renderers, call the *app.enableSandbox()* API before the app's ready event is emitted.

- **Lack of navigation limits:** The OONI Probe desktop application does not limit the navigation to arbitrary origins using *new-window* and *will-navigate events*[9]*.* The navigation to arbitrary sites in an Electron application may facilitate RCE; by leveraging these events, all external navigation can be restricted. For instance, if a user-provided URL is somehow navigated and rendered in the OONI Probe desktop application, RCE can be achieved since node integration is enabled.
- **Absent CSP:** The observation was made that the OONI Probe application does not leverage Content-Security-Policy and its intrinsic benefits. This security feature serves as an additional layer of defense, allowing one to define policies for certain HTML tags such as script elements, which includes the origin a resource can be loaded from and more. Generally speaking, the primary feature of this CSP is to ensure that abusive HTML injection is either completely deterred or rendered highly difficult to achieve. Therefore, one can recommend deploying CSP for the OONI Probe application to guarantee that the platform can fully benefit from the security features offered.

To conclude, Cure53 strongly advises adhering to these generic Electron security recommendations to negate any potential RCE sinks, even though running remote content in the OONI Probe desktop application is currently not possible.

TTP-01-005 WP5: Priv esc from *nobody* to *build* in Tor browser build script *(Info)*

***Note**: Due to this vulnerability's limited exploitability, this ticket was assigned as miscellaneous. In this scenario, a user would need to have previously compromised the build machine. Additionally, the Tor Project makes use of reproducible builds that are created and signed on multiple machines. As such, this issue should still be addressed to conform with defense-in-depth principles.*

An inspection of the Tor Browser's build process led to the discovery of an insecure *PATH*[10] concatenation, which allows the local user *nobody* to gain the privileges of the *building* user. To exploit this weakness, an attacker with *nobody* privileges would create the directory */var/tmp/dist/rust/bin* containing a malicious binary named *tar*. This binary would then be called by the build process as the *building* user. Even though the build bash script would create the folder */var/tmp/dist/rust/bin* with correct and secure permissions, exploitation is possible in this regard due to the fact that this will also run if directory creation fails. This is caused by the previous, maliciously created folder with the same name offering insecure permissions.

[9] https://www.electronjs.org/docs/latest/tutorial/security#13-disable-or-limit-navigation
[10] https://en.wikipedia.org/wiki/PATH_(variable)

Affected file:
https://gitlab.torproject.org/tpo/applications/tor-browser-build/-/blob/main/rbm.conf

Affected code:
```
cargo_vendor: |
  #!/bin/bash
  [...]
  export PATH="/var/tmp/dist/rust/bin:$PATH"
  tar -xf [% project %]-[% c('version') %].tar.gz
```

To mitigate this issue, one potential solution would be to insert *set -e*[11] into the inline bash script, which will abort the execution in the event of an error and therefore block an attacker from placing malformed files into the *tmp* directory. The bash script would then desist if a folder of this nature is present. Generally speaking, the insertion of the /tmp folder to the *PATH* variable is discouraged since a vast array of detrimental side effects may be incurred.

TTP-01-006 WP2: Rdsys depends on outdated packages with known CVEs *(Low)*

During the security assessment of rdsys, the observation was made that several software packages leveraged outdated versions that are vulnerable to a host of security risks. The following software packages were identified as out-of-date and potentially insecure. Notably, the version information provided is based on data collected at the time of testing. Whether these vulnerabilities are exploitable or not depends on the relevant functionality usage in the targeted application. Please note: this ticket should not be deemed to incur application risk but serves to highlight the lack of supply chain security automation.

Affected files:
rdsys/go.mod

Vulnerable packages:

CVE ID	Component	Installed version
CVE-2022-27664 CVE-2022-41717 CVE-2022-41723	*golang.org/x/net*	0.0.0-20220520000938-2e3eb7b945c2
CVE-2022-32149	*golang.org/x/text*	0.3.7
CVE-2022-41727	*golang.org/x/image*	0.0.0-20190802002840-cff245a6509b

[11] https://ss64.com/bash/set.html

Notably, the testing team was unable to comprehensively verify any potential impact during the limited time frame callocated for this review. As such, implications remain unknown at this point and should be reviewed by the developer team.

Generally speaking, the provision of resilient supply chain security can be considerably challenging. Typically, one cannot simply offer an easy or comprehensive solution here. The outcomes of the chosen protection framework can entirely depend on the integrated version of the deployed libraries.

To mitigate the existing issues to the highest possible degree of efficacy, Cure53 recommends upgrading all affected libraries and establishing a policy to ensure libraries remain up-to-date moving forward. As a result, the components can benefit from patches integrated for previously-detected flaws across a number of different solutions. For this purpose, some package managers offer the capability to automate package updates to non-vulnerable versions. Notably, the degree of protection may vary; up-to-date retention typically becomes more difficult to achieve with additional third-party libraries in play.

Under certain circumstances, one may have to resort to either sending PRs to the library maintainer or even forking the library. The Tor Project could consider assigning a developer as the task owner to ensure this issue is not relegated to the backlog. Lastly, replacing certain libraries with actively-maintained alternatives may become a necessity over time.

TTP-01-007 WP4: Conjure depends on outdated packages with known CVEs *(Low)*

Testing confirmed that various software dependencies related to the Conjure source code are outdated and vulnerable. The following software packages have been specifically pinpointed as outdated and vulnerable against multiple vulnerabilities.

Please note that all version information (used and to-be used) was based on data gathered at the time of the assessment or at the time of writing this document. This ticket does not pertain to application security weaknesses, per se, but rather serves to highlight the lack of supply chain security automation.

PoC:
The *govulncheck*[12] utility was leveraged to obtain a list of outdated and vulnerable Golang software dependencies. The following snippet demonstrates the method by which one can scan for vulnerable dependencies within the *coordinator* component:

[12] https://pkg.go.dev/golang.org/x/vuln/cmd/govulncheck

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

```
$ cd conjure
$ govulncheck ./...
[...]

Scanning for dependencies with known vulnerabilities...

Found 3 known vulnerabilities.

Vulnerability #1: GO-2023-1570
[...]

Vulnerability #2: GO-2023-1571
[...]

Vulnerability #3: GO-2022-1039
[...]
```

The following overview lists all identified vulnerable Golang dependencies. All issues flagged with an *Informational* severity marker originate from packages imported by Conjure, though the code does not appear to call any vulnerable functions.

Vulnerable dependencies:

- Vulnerabilities:
  - GO-2023-1570[13]
  - GO-2023-1571[14]
  - GO-2022-1039[15]
- Informational:
  - GO-2023-1569[16]
  - GO-2022-1144[17]
  - GO-2022-0493[18]

In light of this, Cure53 is keen to emphasize that all Conjure application aspects written in Go should be evaluated for outdated and vulnerable dependencies. To mitigate this issue and ensure airtight application security, Cure53 recommends updating all pertinent software to the latest available version. This recommendation owes to the fact that older versions incur known (and unknown) vulnerabilities that may be leveraged by attackers.

[13] https://pkg.go.dev/vuln/GO-2023-1570
[14] https://pkg.go.dev/vuln/GO-2023-1571
[15] https://pkg.go.dev/vuln/GO-2022-1039
[16] https://pkg.go.dev/vuln/GO-2023-1569
[17] https://pkg.go.dev/vuln/GO-2022-1144
[18] https://pkg.go.dev/vuln/GO-2022-0493

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

TTP-01-010 WP3: Deprecated *pycrypto* module used by BridgeDB *(Info)*

Whilst accessing the repository containing the BridgeDB sources, the observation was made that the deprecated pycrypto[19] library is utilized, which is no longer maintained and contains a number of known security vulnerabilities. The lack of active maintenance for this library means that any proliferating vulnerabilities found within this library will never be patched.

Affected files:

- *https://gitlab.torproject.org/tpo/anti-censorship/bridgedb/-/blob/main/bridgedb/bridges.py*
- *https://gitlab.torproject.org/tpo/anti-censorship/bridgedb/-/blob/main/bridgedb/crypto.py*

Affected code:

```
[...]
from __future__ import print_function

import base64
import codecs
import hashlib
import ipaddr
import logging
import warnings

from Crypto.Util import asn1
from Crypto.Util.number import bytes_to_long
from Crypto.Util.number import long_to_bytes

Excerpt from crypto.py
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA
```

To mitigate this issue, Cure53 advises retracting usage of the deprecated and vulnerable pycryto library and replacing it with the pyca/cryptography[20] library, its actively-maintained successor. Supplementary defense-in-depth can also be achieved by initiating regulatory security audits against all third-party libraries and dependencies used within the production codebase.

[19] https://www.pycrypto.org/
[20] https://cryptography.io/en/latest/

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

TTP-01-011 WP6: Lack of country cross-check during auto bootstrapping *(Info)*

During a source code review of the autobootstrapping mechanism, the observation was made that the Tor Browser has not established a cross-check to verify that the selected country actually matches the configuration returned by rdsys / BridgeDB.

Even though the absence of this cross-check may not facilitate immediate security risk, Cure53 must emphasize that a user could inadvertently and obliviously connect to a bridge node located in a different country to that which they explicitly selected.

Affected file:
tor-browser/browser/modules/TorConnect.jsm

Affected code:
```
TorConnectState.AutoBootstrapping,
new StateCallback(TorConnectState.AutoBootstrapping, async function(
             countryCode
             ) {
             [...]
             try {
                    [...]
                    const settings = await this.mrpc.circumvention_settings(
                                   [...TorBuiltinBridgeTypes, "vanilla"],
                                  countryCode
                           );
                    if (settings?.country) {
                           TorConnect._detectedLocation = settings.country;
                    }
                    [...]
             }
             [...]
      }
```

This issue was dynamically tested by instrumenting the Tor Browser source code; specifically, the *TorConnect.jsm* file was extended to log the received bridge settings. Testing confirmed that a country amendment would remain unnoticed by the user in question.

To mitigate this issue, Cure53 advises including an additional cross-check within *TorConnect.jsm*, which would prevent any scenario whereby the Tor Browser connects to a bridge node located in a different country than that explicitly selected.

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

TTP-01-012 WP6: TODO's indicate FW & proxy may be set by rdsys *(Info)*

Whilst conducting a review of the source code for Tor's Moat module, the test team noted that rdsys / BridgeDB generates a bridge list in JSON format, which is subsequently stored in the Tor settings module. However, the source code responsible for transforming the JSON output into a format compatible with the Tor settings module contained several TODO comments indicating that the authors are considering incorporating the conversion of firewall and proxy settings within this process. Tor Browser client security may be significantly compromised if a remote entity under the control of an attacker, e.g. a censor, can manipulate the Tor client's firewall and proxy settings. In this situation, an attacker could reroute client traffic to a proxy server of their choice, allowing them to potentially eavesdrop the communication or block communication.

Affected file:
tor-browser/browser/modules/Moat.jsm

Affected code:
```
_fixupSettings(settings) {
      [...]
      try {
      let retval = TorSettings.defaultSettings();
      [...]
            if ("proxy" in settings) {
              // TODO: populate proxy settings
            }
            if ("firewall" in settings) {
              // TODO: populate firewall settings
            }
            return retval;
}
```

To mitigate this issue, Cure53 recommends scrutinizing the proposed modifications to the source code mentioned in the preceding TODOs, which would enable remote configuration of the firewall and proxy settings of Tor users via bridge-list updates.

TTP-01-013 WP6: Potential risk via QR code returned by */check* API call (*Info*)

During a source code review of Tor's Moat module, the test team noted that bridges can be retrieved from rdsys / BridgeDB by solving a CAPTCHA challenge. The solution to the challenge is uploaded to rdsys / BridgeDB via the */check* API call and optionally returns a QR code for the requested bridge. In the event an rdsys / BridgeDB instance is controlled by an attacker, the aforementioned QR code can be abused to redirect the Tor user to a malicious website or download malware onto the user's device.

Affected file:
tor-browser/browser/modules/Moat.jsm

Affected code:
```
async check(transport, challenge, solution, qrcode) {
  const args = {
    data: [
      {
        id: "2",
        version: "0.1.0",
        type: "moat-solution",
        transport,
        challenge,
        solution,
        qrcode: qrcode ? "true" : "false",
      },
    ],
  };
  const response = await this._makeRequest("check", args);

  [...]
 }
```

Notably, the audited Tor version invokes the */check* API with the *qrcode* parameter set to *false*, thereby guaranteeing insusceptibility to this issue at the time of testing.

Nevertheless, to mitigate this issue, Cure53 advises extensively reviewing the optional QR code generation on the server side and considering implementing QR code generation exclusively on the client side.

TTP-01-014 WP3: Potential risk via Python *pickle* (*Info*)

Whilst reviewing the BridgeDB repository, the observation was made that that BridgeDB's current state is saved in a *.state* file using Python *pickle*, which represents a serialization library used to convert a Python object into a stream of bytes that can be saved to disk or transferred over a network. Whilst *pickle* offers a convenient method for saving and loading complex Python objects, some security risks are facilitated.

The primary security risk incurred by *pickle* is the ability to execute arbitrary code. Since *pickle* is able to serialize not only data but also code, an attacker could potentially construct a malicious *pickle* payload that, when unpickled, executes code that performs undesirable actions, such as deleting files, modifying system settings, or even accessing sensitive data.

Another *pickle*-related security risk pertains to deserialization attacks, in the event an attacker supplies a malicious serialized object that, when deserialized, exploits vulnerabilities in the application that processes the object. This, in turn, can lead to remote code execution, privilege escalation, and other attack scenarios.

Here, Cure53 is keen to underline that a remote attack cannot alter the contents of the file deserialized using Python *pickle* via the aforementioned scenario. Hence, exploitation is currently impossible and this ticket alternatively serves to highlight the potential implications of employing Python *pickle*.

Affected file:
bridgedb/bridgedb/persistent.py

Affected code:
```
[...]
    def load(self, statefile=None):
        [...]
        quo= fh = None
        err = ''

        try:
            if isinstance(statefile, (str, bytes)):
                fh = open(statefile, 'rb')
            elif not statefile.closed:
                fh = statefile
        [...]
        else:
            try:
                status = pickle.load(fh)
            except EOFError:
                err += "The statefile %s was empty." % fh.name
            else:
                quo = jelly.unjelly(status)
                if fh is not None:
                    fh.close()
                if quo:
                    return quo

        if err:
            raise MissingState(err)
```

To mitigate this issue, Cure53 advises retracting usage of *pickle* for the purpose of serializing and deserializing data from untrusted sources, since *pickle* is considered notoriously insecure[21]. If *pickle* remains a necessary requirement, the following best practices should be adhered to:

- Never unpickle untrusted data from an untrusted source. Only unpickle data from a trusted source that is known to be safe.
- Utilize the latest version of *pickle*, since additional security improvements are offered.
- Use a whitelist of allowed classes when unpickling data and ensure that the classes are safe to unpickle.
- Leverage a checksum or digital signature to verify data integrity before unpickling.

TTP-01-015 WP3: HTTP *X-Forwarded-For* config used for HTTPS distributor *(Low)*

During the source code review of the BridgeDB repository, the test team noted that multiple bridge list distributors are supported by the backend, i.e. via HTTPS, Moat, or email. The configuration file and BridgeDB source code highlight that distribution via HTTP could also be used, though this option is deactivated by default.

Two of the many configuration options offered here represent *HTTP_USE_IP_FROM_FORWARDED_HEADER* and *HTTPS_USE_IP_FROM_FORWARDED_HEADER*, which are used to determine the client's IP address. If this option is set to *TRUE*, the IP is ascertained by the *X-Forwarded-For* header in the client's requests. Due to the fact that the client IP plays an important role in the resulting bridge list returned to the client, setting this to *TRUE* could enable a censor to receive a greater volume of bridge IP addresses than anticipated by modifying the *X-Forwarded-For* headers of multiple requests.

[21] https://medium.com/ochrona/python-pickle-is-notoriously-insecure-d6651f1974c9

Even though HTTP is not activated by default, the HTTPS configuration relies on the *HTTP_USE_IP_FROM_FORWARDED_HEADER* configuration option. If a BridgeDB administrator were to activate *HTTP_USE_IP_FROM_FORWARDED_HEADER*, a backend misconfiguration may subsequently be covertly facilitated, impacting the HTTPS configuration.

Affected file:
bridgedb/bridgedb/distributors/https/server.py

Affected code:
```
fwdHeaders = config.HTTP_USE_IP_FROM_FORWARDED_HEADER
numBridges = config.HTTPS_N_BRIDGES_PER_ANSWER
fprInclude = config.HTTPS_INCLUDE_FINGERPRINTS
```

To mitigate this issue, Cure53 advises utilizing the correct configuration option, namely *HTTPS_USE_IP_FROM_FORWARDED_HEADER*, to mitigate any potential misconfigurations for the BridgeDB backend.

TTP-01-016 WP4: Conjure client leaks SOCKS connection handles *(Low)*

A stringent review of the Clojure pluggable transport wrapper verified that the wrapper fails to close the SOCKS connection in the event of a connection error. This will result in a resource leak that could be abused to deplete the connection resource pool and consequently prevent further connections from being established.

If an attacker is capable of triggering this code path, they will be able to instigate a DoS attack and prevent any further client connections from being made by the Tor Browser. Notably, the standard use case in this scenario stipulates that the Tor Browser connects to the pluggable transport wrapper on localhost. As such, an attacker would already have compromised the host to trigger these connections.

Affected file:
conjure/client/conjure.go

Affected code:
```
func handler(conn *pt.SocksConn, config *ConjureConfig) error {

        shutdown := make(chan struct{})

        bridgeAddr, err := net.ResolveTCPAddr("tcp", conn.Req.Target)
        if err != nil {
                conn.Reject()
                return err
        }
        config.bridgeAddress = conn.Req.Target
        log.Printf("Attempting to connect to bridge at %s", conn.Req.Target)

        // optimistically grant all incoming SOCKS connections and start
buffering data
        err = conn.Grant(bridgeAddr)
        if err != nil {
                return err
        }
        buffConn := NewBufferedConn()

        go func() {
                for {
                        [...]
                }
        }()

        proxy(conn, buffConn)
        log.Println("Closed connection to phantom proxy")
        close(shutdown)
        return nil
}

func acceptLoop(ln *pt.SocksListener, config *ConjureConfig) error {
        defer ln.Close()

        for {
                conn, err := ln.AcceptSocks()
                if err != nil {
                        if e, ok := err.(net.Error); ok && e.Temporary() {
                                pt.Log(pt.LogSeverityError, "accept error:
"+err.Error())
                                continue
                        }
                        return err
                }
                log.Printf("SOCKS accepted: %v", conn.Req)
                getSOCKSArgs(conn, config)
                go func() {
                        err := handler(conn, config)
                        if err != nil {
                                log.Println(err)
                        }
                }()
        }
        return nil
}
```

To mitigate this issue, Cure53 advises sufficiently closing the connection in the event of an error. Even though the likelihood of exploiting this issue is minimal, this will undoubtedly strengthen the security posture of the application in case future alterations modify this behavior.

TTP-01-017 WP6: Enhanced MitM protection recommendation *(Info)*

Generally speaking, the Tor Browser connects to the MoatRPC backend at *https://bridges.torproject.org* to detect the optimal method for bypassing censorship. If successful, this will yield a list of bridges (bridge strings) that are then used when attempting to establish a successful Tor connection.

Cure53 noted that the server certificate is not pinned to a known value, even though the connection uses HTTPS. This may allow a powerful censor to launch a MitM attack and impersonate the backend. Consequently, malicious bridge strings could potentially be injected.

Affected file:
conjure/client/conjure.go

Affected code:
```
_makeHttpHandler(uriString) {
  [...]

  const httpHandler = Services.io
    .getProtocolHandler("http")
    .QueryInterface(Ci.nsIHttpProtocolHandler);
  const ch = httpHandler
    .newProxiedChannel(uri, proxyInfo, 0, undefined, loadInfo)
    .QueryInterface(Ci.nsIHttpChannel);

  // remove all headers except for 'Host"
  const headers = [];
  ch.visitRequestHeaders({
    visitHeader: (key, val) => {
      if (key !== "Host") {
        headers.push(key);
      }
    },
  });
  headers.forEach(key => ch.setRequestHeader(key, "", false));

  return ch;
}
```

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

To mitigate this issue, Cure53 advises certificate pinning for connections to the backend, which will help prevent MitM attacks from powerful censors. One downside with this implementation is the increased maintenance effort required for frequent certificate renewal. As such, an alternative approach would constitute signing responses returned by the backend, as discussed in ticket TTP-01-009.

TTP-01-018 WP2: Out-of-memory DoS via buffered reader *(Low)*

Further scrutiny of the rdsys *https* delivery mechanism indicated that a buffered reader is utilized to read data from the backend, which reads and buffers data until the message delimiter occurs. As a consequence, a hostile backend could send endless data lacking the expected delimiter, resulting in memory exhaustion and therefore a possible DoS vector.

Affected file:
rdsys/pkg/delivery/mechanisms/https.go

Affected code:
```
func (ctx *HttpsIpcContext) handleStream(req *core.ResourceRequest) {
[...]
		for {
				line, err := reader.ReadBytes(InterMessageDelimiter)
				if err != nil {
					retChan <- err
					return
				}
				incoming <- bytes.TrimSpace(line)
		}
[...]
}
```

To mitigate this issue, Cure53 advises reading data chunk-wise with a fixed limit to avoid memory exhaustion. Whilst this approach would require additional lines of code, beneficial robustness is offered, permitting connection abortion if a malicious entity attempts to flood the service with unlimited data.

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

TTP-01-019 WP4: DoS via unlimited concurrent connections *(Info)*

Cure53's analysis of the pluggable transport server component revealed that no maximum limit of concurrent connections had been established. Whilst Golang is extremely efficient when managing routines in the background, this pattern can become a bottleneck and a potential DoS vector.

Pertinently, this behavior is not limited to the active client socket only and also pertains to the memory reserved for every connection; the full implications can only be determined by the behavior of the *proxy*() function.

Affected file:
anti-censorship/pluggable-transports/conjure/-/blob/main/server/server.go

Affected code:
```
func acceptLoop(ln net.Listener) {
    for {
        conn, err := ln.Accept()
        if err != nil {
                if err, ok := err.(net.Error); ok && err.Temporary() {
                        continue
                }
                log.Printf("Error accepting conjure connection: %s", err)
                break
        }
        log.Printf("Received client connection from %s",
conn.RemoteAddr().String())
        go func() {
                defer conn.Close()
                or, err := pt.DialOr(&ptInfo, conn.RemoteAddr().String(),
"conjure")
                if err != nil {
                        log.Printf("Error dialing OR port: %v", err)
                }
                defer or.Close()
                proxy(or, conn)
                log.Printf("Done proxying client connection from %s",
conn.RemoteAddr().String())
        }()
    }


}
```

To mitigate this issue, Cure53 advises implementing a limit on the volume of concurrent connections to avoid a potential DoS situation.

Conclusions

This report marks the inaugural security evaluation against the Tor Browser project and varying characteristics by Cure53.

The conclusory outcomes of this CW07 to CW11 2023 Tor Browser project examination primarily indicate adequate robustness and sound design implementations. Eight members of the Cure53 testing team documented nineteen issues that were deemed to have a detrimental impact on the Tor security landscape. Three of the tickets were categorized as exploitable vulnerabilities, two of which were considered *High* in nature and the other *Medium*. An exhaustive list of steps and methodology applied during this security assessment are outlined in the *Test Methodology* chapter.

In context, communication was achieved via a shared Signal channel, cross-team queries regarding certain findings and functionality were promptly answered, and the engineering team provided immediate assistance to the testing team when required.

Prior to the test initiation, the client provided comprehensive documentation that served to define key areas of interest and scope designation. This proved highly assistful and allowed the testing team to quickly familiarize itself with all in-scope features.

Conversely, a number of aspects and attack vectors were explicitly declared out of scope for this security evaluation, as follows:

- The Firefox codebase in general: For example, typical attack scenarios for browsers focusing on memory corruption vulnerabilities present in the code that renders web content, as well as those residing in sandbox interfaces and other relevant areas, were not inspected.

- Attackers deemed passive in nature: This includes entities such as censors that simply observe traffic during the bootstrapping phase and as part of their automatic censorship circumvention efforts.

Compositionally, the codebases of the various work packages were mostly developed in Golang and JavaScript. The codebases were written to a first-rate standard and evidently conformed to secure coding practices. However, due to the sheer complexity and magnitude of the software, identifying the most crucial points and relevant areas to examine remained challenging.

Despite simultaneous work package coverage from a number of highly-skilled test team members and independent source code repository evaluations for optimal code coverage, only three vulnerabilities were identified.

Moving forward, Cure53 will now comment on the erroneous behaviors encountered for each WP specifically, starting with WP1 as follows:

- The testing team initiated a deep-dive assessment of the OONI Probe desktop application frontend in an attempt to detect any presence of client-side security issues such as XSS, prototype pollution, or other user-input-related issues.

- The general absence of significant findings can be attributed to the fact that the OONI Probe desktop application utilizes the React framework, which leverages a battle-tested escaping mechanism that inherently prevents XSS issues by default. Furthermore, use of *dangerouslySetInnerHTML* and other framework issues specific to React were avoided.

- Following the stringent frontend analysis, the testing team switched focus to the ElectronJS code. Initially, Cure53 sought to determine whether the application adhered to general ElectronJS security recommendations and best practices, or conversely persisted deep link misconfigurations. Here, the verification was made that the OONI Probe application does not comply with the majority of advised Electron guidance, as detailed in ticket TTP-01-004.

- Subsequently, the common ElectronJS sinks and sources were audited for potential vulnerabilities, which included (but was not limited to) deep link misconfigurations, Shell API usage in conjunction with untrusted user content, improper utilization of sandbox functionalities, navigation misconfigurations, command injections, and insecure IPC handlers. In this respect, only one minor issue was noted, as detailed in ticket TTP-01-002.

- To summarize for WP1, the OONI Probe desktop application exhibits a healthy security foundation, as evidenced by the considerably low volume of exploitable vulnerabilities unearthed here. This outcome certainly indicates that the OONI Probe desktop application offers commendable resilience.

Next, Cure53 will now comment on the findings encountered during the WP2 assessment:

- Firstly, the rdsys software was thoroughly probed to alleviate any common web-security vulnerabilities.
- The security posture exhibited by the examined rdsys application was evidently robust, as confirmed by the detection of only one issue classified as vulnerability (see TTP-01-008) and one miscellaneous in nature (see TTP-01-006).
- To summarize for WP2, the complete lack of noteworthy injection-based issues - including SQLi, split API, and command injections - underscores the praiseworthy outcome. However, this result also results from adequate usage of the Golang language, which provides stable protection against a number of attack scenarios by default.

Moving on, Cure53 will now comment on the impressions gained following the completion of testing against WP3:

- The BridgeDB repository consists of a frontend and backend implementation written in Python. The backend implementation serves as a collection of backend servers providing bridge lists for clients.
- The evaluation of the BridgeDB component constituted an in-depth architecture and design review for potential security issues that could reside within the build and deployment process, as well as the source code itself.
- With this in mind, the Python dependency and attached libraries were inspected to enumerate any proliferating flaws that may incur security-related risk to the service. These efforts led to the detection of the issue described in ticket TTP-01-010, which specifically pertains to a deprecated crypto library within the BridgeDB source code that currently suffers from several known vulnerabilities. This should be addressed at the earliest possible convenience due to the fact that the dependency is no longer maintained and, as such, said concern will never be fixed.
- A miscellaneous issue was identified whereby the backend state is persisted to disk using Python *pickle*, as documented in ticket TTP-01-014. Since Python *pickle* may be leveraged by an attacker that is able to alter unpickled data and achieve code execution, the test team deemed it necessary to document this ticket and raise awareness of the associated risk.

- Lastly, a configuration issue was detected concerning the implementation of an HTTP config option for the HTTPS distributor, as detailed in ticket TTP-01-015.

- To summarize for WP3, the BridgeDB repository tested favorably on the whole, as corroborated by the clean composition and astute coding practices offering optimal security implementation.

Onto the next work package evaluated, WP4, and all notable findings discussed below:

- The Conjure implementation is written in Go and the test team was easily able to comprehend its structure. However, some dependencies were confirmed out-of-date, as stipulated in ticket TTP-01-007.

- The repository primarily implements glue code between the tapdance implementation for registration to phantom proxies (client-side) and the pluggable transports implementation. The server-side implementation serves as a bridge of Conjure stations.

- Here, the code was meticulously appraised for connection setup and logic issues, which unearthed a potential resource leak as described in ticket TTP-01-016. Additionally, a potential DoS vector was identified and subsequently outlined in ticket TTP-001-019.

- Pertinently, the audit team did not examine the tapdance and go-proxyproto go dependencies, since these libraries were not altered during the Conjure implementation and were not specifically requested by Tor. However, *goptlib* - which implements the pluggable transport solution through IPC with the Tor process and serves for traffic transformation purposes - was subjected to a brief assessment. The key focal point here was the detection of any connection setup, authentication, and logic issues, though no associated flaws were encountered.

- To summarize for WP4, the Conjure implementation was found to offer a reasonable security foundation in general.

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

Next, Cure53's general impressions concerning the WP5-related analysis are discussed below:

- WP5 comprised a review and audit of the build process, scrutinizing the varying CI processes and deployment scripts. This particular engagement was initiated with a check to determine the exact methods by which the different projects were built and deployed. Following this, the CI/CD, build environment, the building process itself, and the software dependencies were subjected to targeted examination.

- Whilst most connected components were deemed concisely and securely designed, the test team noted an antipattern regarding usage of overly permissive UNIX file and path permissions. The consequential issues are further outlined in tickets TTP-01-001 and TTP-01-005.

- To summarize for WP5, Cure53 is pleased to confirm that the The Tor Project building procedure is praiseworthy, highly advanced, and deliberately security focused. This primarily owes to the use of reproducible building processes in tandem with build signing and parallel building, which all integrate considerable defense-in-depth for the components in focus.

Last but not least, Cure53 will now discuss the findings encountered during testing against WP6, as follows:

- A significant emphasis was placed on scrutinizing the censorship circumvention feature, particularly concerning attacks that may be launched by malicious censors that had already compromised an rdsys / BridgeDB instance. This was deemed a crucial aspect of the testing process, since the feature was designed to circumvent censorship in areas whereby internet access was heavily restricted and attackers may seek to exploit vulnerabilities in the system.

- This WP pertains to selected merge requests regarding the censorship circumvention implementation in Tor. The code consisted of JavaScript and CSS code (UI related) for the most part.

- The merge requests do not offer clean and concise separation of concerns with regards to features. As such, the test team found following the code challenging at first. Furthermore, strict separation between UI code and logic within the MRs was not enforced. The overall code size of alterations was vast. In light of this, Cure53 would like to stress the inherent difficulty of gaining a comprehensive overview of the censorship circumvention by merely perusing a list of curated merge and pull requests, as well as GitLab issues. These individual assets did provide valuable insight into the project's development process, though Cure53 still required supplementary efforts to gain a holistic understanding of the overall objective and design.

- Throughout the audit process, heightened scrutiny was placed on four critical modules that formed the backbone of the censorship circumvention feature, specifically *Moat.jsm*, *TorConnect.jsm*, *TorSettings.jsm*, and *BridgeDB.jsm*.

- During this WP examination, one vulnerability was identified stipulating that the bridge list returned by the rdsys / BridgeDB instance is not cryptographically signed (see TTP-01-009).

- To enhance the security posture of the censorship circumvention feature, Cure53 advises implementing additional security controls, which have been listed within the *Miscellaneous Issues* chapter. Here, the majority of the discoveries were merely considered of *Info* severity and should be trivially easy to resolve. For instance, several enhancements regarding the censorship circumvention feature were proposed, as discussed in tickets TTP-01-011, TTP-01-012, TTP-01-013, and TTP-01-016.

Across all working packages, supply chain aspects were rigorously inspected. Here, some dependencies were confirmed outdated (see TTP-01-003, TTP-01-006, and TTP-01-007). Since the Tor Project leverages third-party libraries for specific operations, such as tapdance and go-proxyproto for Conjure, Tor's security offering also heavily depends on the resilience offered by the deployed third-party libraries. As a result, Cure53 recommends applying a strict update regime to these dependencies, which will help to mitigate any security issues that they persist at the point of emergence.

To conclude, the vast majority of the tested features withstood Cure53's advanced penetration skill set, successfully negating a host of potential flaws. This evidently reflects approvingly on the Tor security landscape in general. All in all, the Tor Browser and ecosystem are in a healthy state from a security perspective, even considering the relatively high yield of nineteen findings. However, this total is not unusual for a project of this enormity.

Moving forward, the Tor browser software complex would certainly profit from recurrent security assessments. Notably, the immense complexity of all working packages and components is challenging to handle from a security perspective. As such, any amendments implemented within one system area may incur a detrimental butterfly effect for other seemingly unrelated aspects.

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

The multilayered and rich feature-set of the various components in scope requires a plethora of deep dive audits to provide the most comprehensive evaluation assessment.

To finalize, the testing team achieved strong coverage over all WPs and did not alleviate any major security-relevant discoveries during this February-March 2023 project. Cure53 hopes that all findings and recommendations offered in this report are adhered to for subsequent security developments within the Tor Browser software complex.

Cure53 would like to thank Micah, Gaba, and all other participatory personnel from the Tor Project team for their excellent project coordination, support, and assistance, both before and during this assignment.