Fine penetration tests for fine websites

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

Review-Report Psiphon psipy Library 07.2020

Cure53, Dr.-Ing. M. Heiderich, N. Hippert, BSc. F. Fäßler

Index

Introduction

> *“Psiphon is a circumvention tool from Psiphon Inc. that utilizes VPN, SSH and HTTP Proxy technology to provide you with uncensored access to Internet content. Your Psiphon client will automatically learn about new access points to maximize your chances of bypassing censorship.*
>
> *Psiphon is designed to provide you with open access to online content. Psiphon does not increase your online privacy, and should not be considered or used as an online security tool.”*
>
> From https://psiphon3.com/en/index.html

This report describes the results of a security audit and best-practices review focused on the Psiphon psipy library and its codebase. Carried out by Cure53 in summer 2020, the project was executed with a strong focus on features such as PKI, secure rendering of returned content and the XBacked components

The assessment belongs to a broader cooperation between Cure53 and Psiphon, with this July 2020 instance marking the third assignment. As for the resources, three members of the Cure53 team spend ten days on the scope in CW28. Contrary to

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

previous assignments Cure53 has executed for Psiphon, this exercise not only targeted security and privacy aspects of the library but also assessed the more general coding practices, as well as entailed reviews of safeguards for secure usage in less secure environments.

The work was split into two different work packages, one being more general in scope and the second one addressing the targets with a tad more specificity. In WP1, Cure53 completed a source code review centered on best practices on the Psiphon psipy library and API. In WP2, the focus has shifted to Psiphon psipy library & API security audits, with special attention placed on PKI, rendering and XBacked components.

The chosen methodology was white-box and Cure53 got access to the source code of the library via GitHub repository. In addition, code snippets and example software leveraging the library and its features were furnished to the testing team. Cure53 further was briefed about the expected focus areas by the Psiphon team. A communications channel was set up as well, just as last time. More precisely, the Psiphon and the Cure53 teams met in a dedicated private Slack channel to discuss the progress of the test, questions and results. The communications were productive and helpful; no road-blocks hindered the test and audit from progressing well.

The work carried out managed to yield three findings, yet none of them were classified to be security vulnerabilities. All items represent general weaknesses marked by lower exploitation potential. Most of the findings describe possible risks not mitigated properly by the library upon software using it insecurely.

In the following sections, the report will first shed light on the scope and key test parameters. In the absence of findings, the report will detail the coverage reached by Cure53 and describes briefly what the audit focused on and how. Next, all findings will be discussed in a chronological order alongside technical descriptions, as well as PoC and mitigation advice when applicable. Finally, the report will close with broader conclusions about this 2020 project. Cure53 elaborates on the general impressions and reiterates the verdict based on the testing team's observations and collected evidence. Tailored hardening recommendations for the Psiphon psipy library compound are also incorporated into the final section.

Note: *This report was updated on August 19th 2020 to reflect all changes between report submission and release date.*

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

Scope

- **Psiphon psipy Library & API Source Code Review for Security Best Practices**
  - **WP1**: Psiphon psipy Library & API Source Code Review for Security Best Practices
  - **WP2**: Psiphon psipy Library & API Security Audit, PKI, Rendering & XBacked
    - The tests and reviews focused on the following areas of interest:
      - Reviews targeting possible Logic Bugs & Authentication Flaws in library and API; Reviews targeting possible ACL- & RBAC related Security Issues in the API codebase
      - Reviews targeting possible issues causing Data Leakage or PII Leaks
    - Cure53 further focused on the following areas:
      - Reviews targeting risky rendering and templating code patterns (i.e. template injections); Reviews targeting type confusion issues leading to privilege escalation and information leaks
      - Reviews targeting the PKI logic and checking for insecure certificate generation & validation; Reviews targeting possible flaws in deployment of API endpoints and library code
      - Reviews targeting possible unsafe use of de-serialization of attacker provided strings; Reviews targeting possibly unsafe use of eval or eval-like code constructs in Python
      - Reviews targeting other Python-specific secure coding pitfalls
  - **Sources & Material were shared with Cure53**

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

Test Methodology

The following section documents the testing methodology applied during this engagement and sheds light on the various areas of the code base to inspection and audit. It further clarifies which areas were examined by Cure53 but did not yield any findings.

SCA: Code Audit against the Psiphon Library

The information below describes the coverage achieved for the source code audit against the Psiphon library, which will soon be used in production. The section comments on which areas were investigated and what kinds of approaches were used.

- The documentation was examined to understand the provided functionality and acquire examples of what a real-world integration of the components would look like.
- The project's external and third-party dependencies were cross-checked for problematic components; this revealed a broader lack of dependency versioning and patch management.
- The validation and verification logic, along with potentially incomplete regular expressions, were checked for flaws and bypasses.
- Potential user-input sources were examined for flaws within the critical handling of path-dependent business cases as well as general data-access patterns.
- PKI-related business cases have been checked for possible process shortcomings.
- The HTTP concerned backend calls and integration of user-data within untrusted HTTP handling have been checked for potential injections.
- The template-based configurations have been checked against known injection and validation problems.
- The source code was searched for dynamic database queries with user-input but none could be identified.
- Cure53 performed an in-depth dependency mapping, to determine relevant and potentially dangerous use-case scenarios within upstream application integrations.

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

Miscellaneous Issues

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

PSI-03-001 LocalStore: Path traversal in *namespace* parameter *(Info)*

In order to ensure the robustness of the psipy library, it is important to guarantee that the library prevents path traversals by functions that are intended to provide a safe file location to the developer. It was found that *psipy.localstore* takes two parameters used as a file path, which may include path traversal of type "*../*".

Example Code:

```
from psipy.localstore import LocalStore

x = LocalStore.get("XXXX", "YYYY/../")
```

Example Files:

As can be seen, the code above creates the file *XXXX* on the same level as the namespace *YYYY*

```
/var/folders/3n/b2s1zrsn2x10nxl_vbz0ys480000gn/T/26941eb9-2e59-4930-9eba-
17d3f71d4f5a/
total 0
-rw-r--r--  1 user  staff   0 Jul  7 10:21 XXXX
drwxr-xr-x  2 user  staff  64 Jul  7 10:21 YYYY
```

The example code in *examples/metrics/drawing.py* also shows that the returned path is often used and extended with *.joinpath(...)*, which may also lead to path traversals if the application script is not sanitizing the input properly. Thus, it might make sense to overwrite this function and create a safe join path method. In many other places, potential user-controlled data is string-concatenated and used as a path. For example, *hashed* in *SubprocessBackedTask* is used as a path for the logfile.

In order to prevent risky programming errors, it is important to strongly sanitize the calculated path and ensure that it is still inside the expected temporary folder and *namespace.* The web-framework *Django* implements a *safe_join*[1] function to prevent these kinds of mistakes, so it could be used as inspiration here. Other locations which employ user-input for paths and filenames should ensure that the input is not only free of

[1] https://github.com/django/django/blob/eed2e740f7efa9a9290fb913f79643...go/utils/_os.py#L9

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

filesystem-relevant characters such as “.” or “/” (dot or slash), but also make sure that no *null*-bytes are included and capable of causing an exception.

Note: *The issue was fixed by the Psiphon team and the fix description was reviewed and confirmed by Cure53 in August 2020.*

PSI-03-002 Injection risks in templated configuration strings *(Info)*

The *psipy* library uses templated config files in various places. It mainly uses *jinja2* as the engine for implementing such functionality. *jinja2* is widely used by web-frameworks as the templating engine of choice and successfully prevents (HTML) injection issues in many cases. *psipy* does not benefit from this, however, and is using it like a regular format string. If user-controlled input is rendered into one of the templates, it could lead to arbitrary code execution.

Example Affected File:
/psipy/clients/ciphershare.py

Example Affected Code:

```
# [...]
context = {
            "scripting_client_runner": self.config.scripting_client_runner,
            "db_path": self.config.db_path,
            "host": self.config.host,
# [...]
}
context.update(kwargs)
command_template = jinja2.Environment(
            loader=cast(Optional[jinja2.BaseLoader], jinja2.BaseLoader)
        ).from_string(command_stub)
return command_template.render(**context)
# [...]
```

Example Template:

```
WINE_CIPHERSHARE_IMPORT_COMMAND = """
{{ scripting_client_runner }}
  ImportDocument
    -DatabasePath '{{ db_path }}'
    -ServerHost '{{ host }}'
# [...]
    -AddVersionIfExists
    -IgnoreKeyTrust
""".replace(
    "\n", ""
)
```

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

The example above shows the creation of a command line string without any form of sanitization. It might not be likely that user-input is passed into those places, but the coding pattern is spread across the whole library and, therefore, increases such risks.

It should be noted that *jinja2* has built-in escaping which can be enabled in the *Environment* via the flag *autoescape=True*. This escaping is intended for HTML contexts however, and *psipy* uses *jinja2* for command or *Kubernetes* job templates. Thus, context-aware escaping should be implemented in the places which use *jinja2*. For example, command templates should ensure input cannot break out of quoted strings. Similarly, input for *Kubernetes* jobs should not be capable of breaking out and defining or overriding configuration variables.

Note: *The issue was fixed by the Psiphon team and the fix description was reviewed and confirmed by Cure53 in August 2020.*

PSI-03-003 TLS verification in Kubernetes-based infrastructure disabled *(Info)*

During the analysis of the codebase for Kubernetes integration, it was noticed that the library stack utilizes TLS. Proper peer verification has been disabled, however, which effectively allows an attacker to mount Man-in-the-Middle attacks between the applications utilizing the software and the employed Kubernetes cluster.

Affected File:
psipy/clients/kubernetes.py

Affected Code:
```
k8s_conf.verify_ssl = False
```

It is recommended to fix the underlying problem which made it necessary to disable TLS peer verification in the first place. This needs to be accomplished before the library is put into production.

Note: *The issue was fixed by the Psiphon team and the fix description was reviewed and confirmed by Cure53 in August 2020.*

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

Conclusions

This summer 2020 project demonstrates both strengths and weaknesses of the examined Psiphon psipy library. After spending ten days on the targeted scope in July 2020, three members of the Cure53 observe that the library is still in the early stages of development.

More importantly, it leaves the impression of having been created by extracting the functionality from the existing projects, with only minor generalizations to produce some form of API abstraction. At the same time, Cure53 believes that clear and clean API structure has not yet been defined, which means that it might lead to problems in the future, potentially requiring modifications that might be incompatible with older releases. This is something that the developers should look into and plan for.

The overall code quality and coding style are good and not many general problems have been found. However, performing a code review on a library without context and intended usage can be difficult as the threat vectors are unknown. Without more complex examples or more thorough documentation, these remain difficult to judge. While a few example scripts for psipy were provided, they do not cover nearly enough features.

The library uses modern Python coding style, including type hints. This should help during development to keep cleaner code and prevent programming mistakes that could lead to issues with “confused-types”. Most functions and classes contain docstrings explaining their usage and parameters well, which is important for a healthy codebase. Typical risks of libraries are functions that pass input to dangerous sinks such as file operations, network requests or command executions. A general pattern that could be observed was the string and path concatenation for several file operations. Many functions passing input into those places do not make it clear to the developer that file access could happen. Thus, there is a risk that a vulnerability is introduced (PSI-03-001).

A slightly related pattern could be identified with the usage of template strings used in different places, such as command or Kubernetes configuration files. Unsanitized input given to certain functions could lead to injection vulnerabilities (PSI-03-002). These should ideally be addressed prior to the project moving forward.

To conclude, the library itself did not contain any serious security issues, which was to be expected in a functional library. However, this summer 2020 project revealed multiple patterns that might lead to serious security issues if the application utilizing the library integrates it the wrong way or does not validate certain input data properly. In Cure53’s

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

expert opinion, those items and behaviors should either be directly stated in the document or must be addressed in the underlying codebase.

Cure53 would like to thank Michael Goldberger, Jacob Klein, Mike Fallone, Irv Simpson and the rest of the Psiphon Inc. team for their excellent project coordination, support and assistance, both before and during this assignment.