Fine penetration tests for fine websites

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

Pentest-Report Psiphon api-gatekeeper 10.-11.2021

Cure53, Dr.-Ing. M. Heiderich, MSc. R. Peraglie, BSc. B. Walny

Index

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

Introduction

“This project is an OAuth authenticated reverse proxy. Once a user has been successfully authenticated, they are granted access to a "bearer URL" which allows them time-restricted access to an otherwise inaccessible backend service. In the default mode, any connection that is still open when the proxy session expires is allowed to complete (this can be disabled via configuration).”

From https://github.com/Psiphon-Labs/api-gatekeeper/

This report describes the results of a security assessment of the api-gatekeeper middleware managed by Psiphon. Carried out by Cure53 in autumn 2021, the project included a penetration test and a dedicated audit of the source code.

To give some details, this is not the first iteration of security-centered cooperation between Cure53 and Psiphon. Registered as *PSI-04,* the project has nevertheless been an initial look at the api-gatekeeper middleware.

The project was requested by Psiphon Inc. in August 2021 and then scheduled for autumn of the same year to allow ample time for preparations. As for the precise timeline and specific resources, Cure53 completed the examination in late September and early October, namely in CW40. A total of eight days were invested to reach the coverage expected for this assignment, whereas a team of three senior testers has been composed and tasked with this project’s preparation, execution and finalization.

For optimal structuring and tracking of tasks, the work was split into two separate work packages (WPs), one technical and one centered on logistics:

- **WP1**: Audits & penetration tests against Psiphon api-gatekeeper implementation
- **WP2**: Reporting, administration, communications.

It can be derived from above that white-box methodology was utilized. Cure53 was given access to the sources of the middleware implementation, with the handover done via private GitHub repository access. Additionally, a readily deployed environment with middleware and an API behind it were provided to make sure the project can be executed in line with the agreed-upon framework.

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

Besides some minor hiccups at the beginning of the test - mostly linked to getting the software to work locally and the setup of a test-environment - the project progressed effectively on the whole. All preparations were done in CW39 to foster a smooth transition into the testing phase in CW40.

Over the course of the engagement, the communications were done using a private, dedicated and shared Slack channel. This channel was established during earlier collaborations between Cure53 and Psiphon. The discussions throughout the test were very good and productive. The testers used it to offer frequent status updates about the test and the emerging findings. Overall, the scope was well-prepared and clear. Once the initial issues were resolved, no noteworthy roadblocks were encountered during the test.

The Cure53 team managed to get very good coverage over the delineated scope. Among eight security-relevant discoveries, four were classified to be security vulnerabilities and four to be general weaknesses with lower exploitation potential. It needs to be noted that the number of findings might very well be limited, yet the severities ascribed to the spotted problems are elevated and majorly concerning. Further note that two findings were classified to be false alerts after a post-audit discussion with the maintainer team.

In the following sections, the report will first shed light on the scope and key test parameters, as well as material available for testing. Next, all findings will be discussed in grouped vulnerability and miscellaneous categories, then following a chronological order in each group. Alongside technical descriptions, PoC and mitigation advice are supplied when applicable. Finally, the report will close with broader conclusions about this autumn 2021 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 complex - particularly related to the api-gatekeeper middleware and its codebase - are also incorporated into the final section.

***Note**: The report was amended with several notes for each finding listed in CW45. Those notes shed light on the state of those tickets after they have been discussed with the maintainer team. Some tickets have been classified as out-of-scope or false alerts.*

***Note**: The report was amended with additional fix notes in CW47. All issues have successfully been addressed at this time.*

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

Scope

- **Audits & Penetration Tests against Psiphon api-gatekeeper Implementation**
  - **WP1**: Audits & penetration tests against Psiphon api-gatekeeper implementation
    - **Repository**
      - https://github.com/Psiphon-Labs/api-gatekeeper/
    - **Environment to test against**
      - https://api-gatekeeper-test.psiphon.io
  - **WP2**: Reporting, administration, communication
- **Key focus areas for this audit & assessment**
  - Reviews targeting possible logic bugs & authentication flaws in codebase and API
  - Reviews targeting typical bugs & bypasses of OAuth & OIDC implementations & middleware
  - Reviews targeting possible ACL- & RBAC-related security issues in API codebase
  - Reviews targeting possible issues causing privilege escalation, data leakage or PII leaks
- **Test-supporting material was made available to Cure53**
- **All relevant sources were shared with Cure53**

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

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *PSI-04-001*) for the purpose of facilitating any future follow-up correspondence.

PSI-04-001 WP1: Unused *authorization* middleware *(High)*

***Note**: This issue has been re-classified as a false alert after a discussion with the maintainer team. The Cure53 team learned that the software in scope is not supposed to serve as authorization layer, hence this finding now classifies as invalid.*

It was found that the *authorization* middleware was not used in the Psiphon api-gatekeeper. This means that access is not restricted in any way, making it possible for unauthorized users to reach restricted resources.

Excerpt from shell:

```
/api-gatekeeper-master$ grep -iR 'AuthorizeActionMiddleware' -n
apiserver/auth/authorization.go:145:// AuthorizeActionMiddleware [...]
apiserver/auth/authorization.go:147:func AuthorizeActionMiddleware() [...] {
apiserver/auth/authorization_test.go:250: [...]
apiserver/auth/authorization_test.go:283: [...]
apiserver/auth/authorization_test.go:284: [...]
apiserver/auth/authorization_test.go:318: [...]
```

It is advisable to add the *authorization* middleware to the middleware chain of the proxy handler. By doing so, all requests going to the API will be restricted by the role-based access control defined by the user.

PSI-04-002 WP1: Authentication bypass in Oauth2 callback handler *(**Critical**)*

***Note:** This issue was fixed by the Psiphon team and the fix was confirmed to be working as expected by Cure53 during the audit.*

It was found that the *GET /oauth2/callback* handler activates a user-initiated session *before* verifying the authentication result of the provider and returning an error message. This allows attackers to activate a session that is not associated with any user without supplying valid credentials. In combination with the missing authorization described in PSI-04-002, attackers are allowed to access the API backend without any restrictions. This means a complete bypass of the API gatekeeper, resulting in a *Critical* severity.

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

Affected file:
api-gatekeeper-master/handlers.go

Affected code:
```
func (gk Gatekeeper) oauthLoginCallbackHandler() http.HandlerFunc {
   return func(w http.ResponseWriter, r *http.Request) {
      //log := hlog.FromRequest(r)
      sessionID := r.FormValue("state")
      session := gk.GetSession(sessionID)
      [...]

      if session.State() != SessionStateActive {
         [...] session.Activate(getBaseURLFromRequest(r)); [...]
         user, err := gk.provider.GetUser(r.Context(), r.FormValue("code"))
         if err != nil {
            jsonErrorResponse(w, [...], "failed to retrieve user details")
            return
         }
         session.SetUser(user)
```

It is recommended that the authentication result is verified before activating the session of the user. By doing so, attackers cannot set the session state to *active* without the providers' approval, thus mitigating this vulnerability.

PSI-04-004 WP1: Usage of non-cryptographically secure PRNG (*Critical*)

***Note**: The issue was fixed and the fix was verified by Cure53 in CW46, The issue no longer exists in the reviewed codebase.*

While auditing the session creation for potential weaknesses, it was found that a non-cryptographically secure PRNG, namely Golang's *math/rand* package, is used for the creation of random strings. This is later tied to the session as an identifier of an authenticated user's session.

Due to the missing cryptographic properties of the source of randomness, it is possible to determine the internal state of the random generator, which lets remote attackers predict all previous and future output. This leads to a hijacking of any session within the gatekeeper application. The relevant files and code parts are shown below.

Affected files:
- *api-gatekeeper-deploy\init.go*
- *api-gatekeeper-deploy\handlers.go*

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

Affected code:

init.go:

```
import (
[...]
    "math/rand"


func generateRandomString() string {ts of entropy, represented in 16 base64
bytes.

    b := make([]byte, 12)
    if _, err := rand.Read(b); err != nil {
```

handlers.go:

```
func (gk Gatekeeper) loginHandler() http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        var rendered bytes.Buffer
        err := gk.loginTemplate.Execute(&rendered, generateRandomString())
```

In order to obtain the internal state of the randomness generator, several different attack strategies can be applied. The most straightforward would be a brute-force attack, since the seed in Golang is truncated to be 32-bits. The server seed which initialized the provided pentesting instance was 402033405. With this knowledge, any output of the RNG would be remotely replicable. A sample demonstration is shown below. The exploit files to brute-force the secret or get the next session are rather simplistic, henceforth not added to the report for brevity's sake.

Proof-of-Concept:

```
$ curl -L api-gatekeeper-test.psiphon.io 2>/dev/null| grep -Po 'session_id=\
K.*(?=">)'
lBs8XH0MrCgMiWt8
$ go run next_session.go lBs8XH0MrCgMiWt8
[x] Found @Index with seed:  50736 402033405
[!] Next session will be:  RTdOhXuM6QWFl79H
$ curl -L api-gatekeeper-test.psiphon.io 2>/dev/null| grep -Po 'session_id=\
K.*(?=">)'
RTdOhXuM6QWFl79H
```

It is recommended to make use of cryptographically secure random generators for all operations that are security relevant, here specifically the session handling identifiers. Golang's official *crypto/rand* package is recommended.

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

PSI-04-005 WP1: Session fixation via login handler *(Low)*

***Note**: The issue was fixed and the fix was verified by Cure53 in CW46, The issue no longer exists in the reviewed codebase.*

It was found that the login handler receives a session parameter that will be used to instantiate a session in question. This is dangerous because it allows attackers to lure a victim on to a benign-looking link that allows attackers to hijack the session.

Steps to reproduce:

1. An attacker crafts malicious URL with chosen session ID:
   /login?session_id=attackersession
2. Victim visits the benign link and authenticates
3. Attacker can confirm and use the session via */session/attackersession*

The session ID should be transmitted in relevant HTTP authentication header fields like *Cookie* or similar. They should only be permitted to be set from the gatekeeper's origin by the gatekeeper itself. Additionally, it is advisable to only allow alphanumeric characters in session IDs to disarm the attackers' potential. By doing so, adversaries can no longer set the session ID by using *form* parameters within a benign URL address, therefore mitigating this vulnerability.

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-04-003 WP1: Proper CORS policy for *metrics* handler *(Info)*

***Note**: The issue was fixed and the fix was verified by Cure53 in CW46, The issue no longer exists in the reviewed codebase.*

It was found that the *sse* handler that delivers session *metrics* in an event-stream format suffers from a permissive CORS policy. As such, it allows cross-origin access to session *metrics*. Since this issue can only be exploited with PSI-04-005 or with the knowledge of the session ID, the flaw was rated as purely informational.

Affected file:
api-gatekeeper-deploy/handlers.go

Affected code:

```
func (gk Gatekeeper) sseHandler() http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        [...]
        // TODO: Proper CORS
        w.Header().Set("Access-Control-Allow-Origin", "*")
```

Just like in PSI-04-005, it is recommended to transmit the session ID only in the *Cookie* header and use a proper CORS policy that only grants cross-origin access to the gatekeeper domain. By doing so, the required cookies will awaken the CORS policy and prevent cross-origin reads from unknown domains.

PSI-04-006 WP1: Memory leak on expired sessions *(Low)*

***Note**: This issue has been re-classified to be a "resource exhaustion" issue than a "memory leak" after a discussion with the maintainers.*

***Note**: The issue was fixed and the fix was verified by Cure53 in CW46, The issue no longer exists in the reviewed codebase.*

It was found that the expired sessions are not deleted from the Map structure that holds all sessions. This induces the risk of attackers spamming pending sessions against the api-gatekeeper. In turn, this would increase the consumed memory and lookup time until the underlying resources are exhausted.

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

Affected file:
session.go

Affected code:

```
func (s *Session) Activate(baseURL string) error {
    s.m.Lock()
    defer s.m.Unlock()

    if s.state != SessionStatePending {
        return fmt.Errorf("user session not in pending state: %w",
              errSessionStateInvalid)
    }

    s.baseURL = baseURL

    s.activated = time.Now()
    s.expiryTimer = time.AfterFunc(s.duration, func() {
        s.Expire()
    })
    s.state = SessionStateActive

    return nil
}
```

Sessions need to be deleted from the Map on logout and expiration. This could be done with the *Remove* function that is already implemented on the *SessionMap* struct. With a revised approach, memory not required anymore is freed, preventing this memory leak that could be exploited for attackers' advantage.

PSI-01-007 WP1: General HTTP security headers missing *(Info)*

***Note**: The issue was fixed and the fix was verified by Cure53 in CW46, The issue no longer exists in the reviewed codebase.*

It was found that the api-gatekeeper is missing certain HTTP security headers in HTTP responses. This does not directly lead to a security issue, yet it might aid attackers in their efforts to exploit other problems. The following list enumerates the headers that need to be reviewed to prevent flaws connected to headers.

- **X-Frame-Options**: This header specifies whether the web page is allowed to be framed. Although this header is known to prevent Clickjacking attacks, there are many other attacks which can be achieved when a web page is frameable[1]. It is recommended to set the value to either **SAMEORIGIN** or **DENY**.

[1] https://cure53.de/xfo-clickjacking.pdf

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

- Note that the CSP framework offers similar protection to X-Frame-Options in ways that overcome some of the shortcomings of the aforementioned header. To optimally protect users of older browsers and modern browsers at the same time, it is recommended to consider deploying the *Content-Security-Policy: frame-ancestors 'none';* header as well.
- **X-Content-Type-Options**: This header determines whether the browser should perform MIME Sniffing on the resource. The most common attack abusing the lack of this header is tricking the browser to render a resource as an HTML document, effectively leading to Cross-Site-Scripting (XSS).
- **X-XSS-Protection**: This header specifies if the browser's built-in XSS auditors should be activated (enabled by default). Not only does setting this header prevent Reflected XSS, but also helps to avoid the attacks abusing the issues on the XSS auditor itself with false-positives, e.g. Universal XSS[2] and similar. It is recommended to set the value to either **0** or **1; mode=block**. Note that most modern browsers have stopped supporting XSS filters in general, so this header is only relevant in case older browsers are supported by the web application in scope.

Overall, missing security headers is a bad practice that should be avoided. It is recommended to add the aforementioned headers to every server response, including error responses like *4xx* items. More broadly, it is recommended to reiterate the importance of having all HTTP headers set at a specific, shared and central place rather than setting them randomly. This should either be handled by a load balancing server or a similar infrastructure. If the latter is not possible, mitigation can be achieved by using the web server configuration and a matching module.

PSI-04-008 WP1: Potential authorization bypass due to path normalization *(Info)*

***Note**: This issue has been classified to be a false alert after a discussion with the maintainers. The Cure53 team learned that the software in scope is not supposed to serve as authorization layer, hence this finding now classifies as invalid.*

It was found that the proxy handler directly forwards the received request to the upstream proxy without validating or normalizing the HTTP request. The future authorization logic uses a URL parameter of the HTTP request to identify the subject and allow or deny access. This introduces the risk of the future authorization logic being bypassed due to path normalization performed upstream on the backend. Attackers could abuse it for unauthorized access.

[2] http://www.slideshare.net/masatokinugawa/xxn-en

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

HTTP request (from attacker to gatekeeper):

```
GET /p/c/subject1/../subject2 HTTP/1.1
Host: 127.0.0.1:1337
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:92.0) Gecko/20100101
Firefox/92.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/
*;q=0.8
Connection: keep-alive
Content-Length: 0
```

HTTP request (from gatekeeper to upstream proxy):

```
GET /subject1/../subject2 HTTP/1.1
Host: 127.0.0.1:1337
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:92.0) Gecko/20100101
Firefox/92.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/
*;q=0.8
X-Forwarded-For: 127.0.0.1
Accept-Encoding: gzip
```

As already recommended in PSI-04-005, it is advisable to only allow alphanumeric characters in the session IDs. Further, it is recommended to perform a strict validation and normalization on the request URL before authorizing and sending the request to the backend. Performing authentication and authorization on the proxies' sides exposes similar risks by design. Therefore, it may be considered to let the proxy authenticate the user by forwarding the request with an authentication token that is verified and authorized again on the backend-side.

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

Conclusions

As already discussed in the *Introduction,* Cure53 was tasked with gauging the security posture of the api-gatekeeper maintained by Psiphon. After spending eight days examining the scope through pentesting and code auditing methods, three members of the Cure53 team concluded this project with mixed impressions.

For the most part, this verdict is dictated by the api-gatekeeper - at least as of October 2021 - giving off an unfinished impression. The fact that *Critical* items linked to authentication basically upend the general objective and aim of the project is worrisome. Moreover, it might indicate that the main idea behind the security of the api-gatekeeper project was not centralized enough within the broader development process.

More specifically, already at the beginning of this testing round the Cure53 team exposed a *Critical* logical flaw (PSI-04-002). This allowed for authentication to be completed successfully without any details. It was further found that any session handled by the gatekeeper could be hijacked due to the usage of weak cryptography (PSI-04-004).

Moreover, the gatekeeper does not scale in a cluster very well: sessions are kept in an ever-growing Golang Map (PSI-04-006) and cannot be shared among nodes. The design makes it impossible to add more gatekeeper nodes. For this reason, it is advised to use JWT tokens with a shared secret among nodes or a centralized database like Redis.

On the positive side, no issues have been found in the realms of PII leaks, privilege escalation or data leakage. This, however, might mostly be due to the general absence of features to provide any grounds for exploitation in that regard. The frontend JavaScript UI was found safe as far as XSS is concerned, despite making unnecessary use of potentially dangerous element properties like *innerHTML*. Here it is recommended to replace unnecessary usage of this item with a safer property like *innerText*.

The OAuth implementation was checked for common flaws and no issues have been found. The single provider integration with Google was found to follow best practices. Similarly, *redirect_uri* was configured properly and no open redirects on the application-side have been found. Potentially tampered with parameters are ignored and instead the IdP backend is being called for data retrieval.

Cure53 would like to thank Irv Simpson, Michael Goldberger and Tasker Mackersy from the Psiphon team for their great project coordination, support and assistance, both before and during this assignment.