Pentest-Report Psiphon 06.-07.2017

Cure53, Dr.-Ing. M. Heiderich, Dipl.-Ing. A. Aranguren, M. Wege, BSc. F. Fäßler,
Dipl.-Ing. A. Inführ, MSc. N. Kobeissi, MSc. N. Krein, BSc. C. Kean, BSc. D. Weißer, J. Larsson

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://www.psiphon3.com/en/index.html

This report documents the findings of a Cure53 security assessment of Psiphon entities. The test took place in late June and early July of 2017 and was carried out over the course of twenty-two days. A team of nine testers from the Cure53 team was involved in this project, which overall resulted in a positive outcome regarding the security of the tested Psiphon suite. The tests revealed a surprisingly small number of findings which, even more unexpectedly, had very low severity rankings.

The assessment set out to examine a rather large number of security realms at Psiphon. In scope were multiple components of the Psiphon software compound, including the tunnel-core client and server, the library glue, the Psiphon iOS app and, last but not least, the Psiphon iOS browser. This very broad premise and scope explain the necessity for involving a rather large number of testers with properly matched expertise in different arenas. In sum, the tests included code audits, actual penetration tests, protocol and configuration reviews, and a cryptographic audit.

As for the approach, the tests followed a white-box methodology. The Cure53 team was granted access to the entire Psiphon code base and received pre-built apps. The testers could consult documentation and set up local and remote test servers to analyze traffic, server robustness, and the general behaviors of the software compound under stress. In addition to the software security testing, the cryptographic implementations were given a thorough review as well. Further, Cure53 looked at the general communication protocol Psiphon uses, checked the server communications and configurations, and inspected the library code used by several components.

The tests progressed smoothly. During the process, the Cure53 testing team was in close contact with the Psiphon team via email. Email communication exchanges were occasionally used to ask for clarity within the updated documentation and pertained to requesting minor assistance during the software setup process. Early on, the testers had shared a conviction that the software compound greatly benefitted from a number of software security audits in the past. Needless to say, this is reflected in findings. Among the total nine issues discovered, only two were marked as security vulnerabilities and were further ascribed with the lowest "Informational" severity ranking. The remaining five findings comprised general weaknesses with similarly minimal bearing on Psiphon's overall security. As it stands, a combination of very clean code with a very precise and specific threat model leaves almost no realistic attack surface for adversaries.

The report will now describe the few findings on a case-by-case basis, then providing an itemized commentary on the cryptographic realm. In the final section, it will deliver a conclusion, again elaborating on test coverage and findings. The Cure53 testing team offers general impressions about the security level of the tested Psiphon software components in the closing paragraphs.

Note: *This report was updated on December 22nd 2017 to reflect all changes between report submission and release date.*

Scope

- **Psiphon Client**
  - https://github.com/Psiphon-Labs/psiphon-tunnel-core/tree/master/ConsoleClient
- **Psiphon Client & Server Binaries**
  - Binaries and sample configuration files were shared with Cure53
- **Psiphon Mobile Binaries**
  - Binaries were shared with Cure53
- **Psiphon Server**
  - https://github.com/Psiphon-Labs/psiphon-tunnel-core/blob/master/Server/README.md
- **Psiphon iOS Library**
  - https://github.com/Psiphon-Labs/psiphon-tunnel-core/tree/master/MobileLibrary/iOS/PsiphonTunnel/PsiphonTunnel
- **Psiphon iOS Browser**
  - https://github.com/Psiphon-Inc/endless

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in a 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-01-001*) for the purpose of facilitating any future follow-up correspondence.

PSI-01-001 Browser/Protocol: Server IP Leak via Traffic Fingerprinting *(Info)*

***Update:** This will be addressed in future releases.*

It was found that the iOS app makes frequent requests to CDN servers. These requests are easy to identify as they conform to a number of patterns which may be used by a malicious government or similar parties to block Psiphon servers. More specifically, heartbeat requests conform to the patterns enumerated below.

1. The requests occur very frequently; when the iOS browser is idle, they occur every one to five seconds.
2. The response is almost always 135 bytes in length for an empty request when the browser is idle.
3. The request is always POST, with the same Cookie and Content-Type; it occasionally switches the Host and User-Agent headers.
4. The host in the Host Header does not correspond to the IP address being queried and fails to DNS-resolve.
5. When a POST body appears, it is encrypted despite the request being clear-text HTTP.

Heartbeat requests can be found next.

Request:

```
POST / HTTP/1.1
Host: www.[redacted]motive.com
User-Agent: Opera/9.80 (Windows NT 5.1; U; en) Presto/2.10.289 Version/12.01
Content-Length: 0
Content-Type: application/octet-stream
Cookie: T=v6UNh9nDupvCw9wZqt-pvA
Connection: close

[...optional encrypted payload here...]
```

Response:

```
HTTP/1.1 200 OK
Content-Length: 0
```

```
Content-Type: text/plain; charset=utf-8
Date: Thu, 22 Jun 2017 14:18:50 GMT
Connection: close
```

It can be observed that the host in the Host Header sent to the CDN server does not resolve to any real domain.

Command:

```
host www.[redacted]motive.com
```

Output:

```
Host www.[redacted]motive.com not found: 3(NXDOMAIN)
```

It is recommended to make requests more indistinguishable from normal browsing-driven traffic. One approach would be to start using request bodies that seem normally URL-encoded instead of having completely encrypted items. Furthermore, the requests should vary in the response size, rely on different intervals for the heartbeats, and employ encodings that use the same character set as the one commonly found on websites. The goal should be to make it as challenging as possible for a censor to discern a normal browsing request from a Psiphon browsing request. Same logic behind the hardening should be applied to responses.

PSI-01-005 Browser: Trivial Censorship Attack via DNS Blocking *(Info)*

***Update:** This issue was fixed and the fixes were verified by Cure53*

It was found that a malicious government can trivially prevent users of the iOS Psiphon Browser from accessing unwanted domains. This can be accomplished by failing to resolve the censored domain names. When this takes place, the Psiphon Browser app lags and is unable to render any sites, including the *psiphontoday.com* start page. This issue can be replicated by changing the DNS settings on a test iDevice, pointing to a server where *dnschef*[1] is running. The aim is to have the DNS request accepted but then see that the DNS query never resolves. This can be accomplished by setting an upstream name server that does not exist, in this example 127.0.0.1

Command:

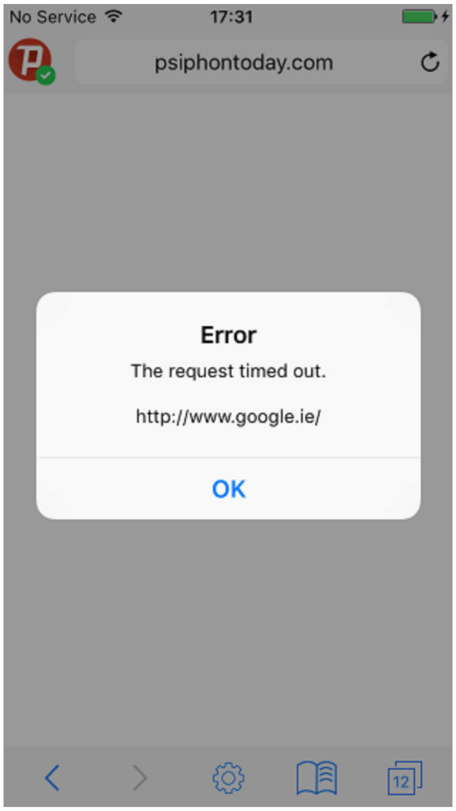
```
dnschef -i 192.168.7.231 --nameservers=127.0.0.1
```

[1] https://thesprawl.org/projects/dnschef/

Output:

```
[17:41:59] 192.168.7.196: proxying the response of type 'A' for
www.psiphontoday.com
[!] Could not proxy request: timed out
[...]
```

In the UI, the browser window remains blank. All attempts to load any pages will fail and after some minutes a pop-up might eventually appear. The displayed feedback is that the request timed out.



Error

The request timed out.

http://www.google.ie/

OK

Fig.: Psiphon is unable to load any page

An error can be tracked on the iOS logs when an attack takes place. This is illustrated next.

iOS logs output:

```
Jun 27 19:01:00 iPhone-5s-196 networkd[267] <Error>: -[NETProxyLookup url]
invalid URL scheme '55348'
```

The root cause for the issue appears to be located in the files specified below.

File:
External/JiveAuthenticatingHTTPProtocol/JAHPAuthenticatingHTTPProtocol.m

Affected Code:

```
- (id)initWithRequest:(NSURLRequest *)request cachedResponse:
(NSCachedURLResponse *)cachedResponse client:(id <NSURLProtocolClient>)client
{
[...]
        if (!_isOrigin) {
                if (![LocalNetworkChecker isHostOnLocalNet:[[mutableRequest
mainDocumentURL] host]] && [LocalNetworkChecker isHostOnLocalNet:
[[mutableRequest URL] host     ]]) {
                        [[self class] authenticatingHTTPProtocol:self
logWithFormat:@"[Tab %@] blocking request from origin %@ to local net host %@",
_wvt.tabIndex, [m     utableRequest mainDocumentURL], [mutableRequest URL]];
                        cancelLoading();
                        return nil;
                }
        }
```

File:
Endless/LocalNetworkChecker.m

Affected Code:

```
+ (BOOL)isHostOnLocalNet:(NSString *)host
{
[...]
        NSArray *ips = [[self class] addressesForHostname:host];
[...]
+ (NSArray *)addressesForHostname:(NSString *)host {
[...]
        CFHostRef hostRef = CFHostCreateWithName(kCFAllocatorDefault,
(__bridge CFStringRef)host);
        if (!CFHostStartInfoResolution(hostRef, kCFHostAddresses, nil)) {
                CFRelease(hostRef);
                return nil;
        }
```

The snippets above demonstrate that the *isHostOnLocalNet* function is invoked every time an HTTP request is made. This results in a DNS resolution via *addressesForHostname*, which in turn uses the *CFHostStartInfoResolution* function in synchronous mode.

As a consequence, execution of Psiphon code paths is blocked when the DNS resolution fails to complete. The explanation of this finding can be read from the Apple documentation[2]:

> *"In synchronous mode, this function blocks until the resolution has completed"*

The excerpt and earlier descriptions demonstrate how selective blocking of DNS responses for unwanted domains can be a very effective way for a censor to defeat Psiphon. It should be noted that the ease of success here is in part due to the leakage of user-activity via clear-text DNS requests over the network. This behavior is not considered an issue by Psiphon as its main goal is to provide censorship circumvention rather than anonymity.

It is recommended to implement a fallback method to resolve DNS queries. The leak should ideally be solved as well so that a censor has no way of knowing which domains are being visited. With a revised approach, blocking would be made substantially more difficult. For this purpose, DNS traffic should ideally be tunneled through a VPN connection.

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-01-002 Server: SSH Banner makes Fingerprinting trivial *(Info)*

Update: *This was flagged and confirmed as false alert, only affecting the development environment the tests took place on.*

It was found that the default configuration of Psiphon servers is to leak the fact that a server is running Psiphon on the SSH banner. This makes it trivial to a censor to fingerprint which IPs are running Psiphon on the Internet and effectively block all of them. This issue was identified with a Cure53 test server. The instructions for following the undertaken steps are furnished next.

Command:

```
nc -nv 35.184.164.24 3001
```

[2] https://developer.apple.com/documentation/cfnetwork/1426...oststartinforesolution?language=objc

Output:
```
(UNKNOWN) [35.184.164.24] 3001 (?) open
SSH-2.0-Psiphon
```

The root cause for this issue can be found on the specific file provided below. It appears that this is a known issue that will be addressed in the future.
File:
psiphon-tunnel-core/psiphon/server/config.go

Affected Code:
```
// TODO: vary version string for anti-fingerprint
sshServerVersion := "SSH-2.0-Psiphon"
```

It is recommended to set a default inconspicuous SSH server version string to make fingerprinting more difficult.

PSI-01-003 Browser: “Disable - JavaScript” Feature Bypass *(Medium)*

***Update:** The issue was addressed by removing the feature. Cure53 reviewed the change successfully.*

The Psiphon iOS application presents users with a setting to disable JavaScript while browsing the web. Factually the iOS app does not really disable JavaScript but instead adds a *Content-Security-Policy* header with the *script-src 'none'* content in hopes of restricting script execution. Although this seems like a simple and straightforward way to disable JavaScript, a website can include a meta tag to execute JavaScript despite CSP’s presence.

File:
endless-master/External/JiveAuthenticatingHTTPProtocol/JAHPAuthenticatingHTTPProtocol.m

Code:
```
BOOL disableJavascript = [[NSUserDefaults standardUserDefaults]
boolForKey:kDisableJavascript];
if (disableJavascript) {
CSPheader = @"script-src 'none';";
}
```

Proof-of-Concept:
```
<html>
<head>
<meta http-equiv="refresh" content="3;
URL=data:text/html,<script>alert(1337)</script>">
</head>
```

```
<body>
<h1>wait 3 seconds for an alert</h1>
<script>alert(1)</script>
```

Steps to reproduce:

- Open the Psiphon app;
- Go to *Settings -> Privacy - > Disable JavaScript;*
- Open the PoC;
- The code *alert(1)* will not be executed but, after three seconds, a redirect to the *data:uri* will happen;
- This will execute *alert(1337)* as the CSP rules are not applied.

As the iOS app already adds a JavaScript file for all loaded web pages, it is recommended to enforce that this file is loaded at the top of the page. Any meta tags specifying a redirect via JavaScript should be removed. The current behavior could lead to race conditions. Therefore, consideration should be given to filtering any HTML response for meta redirects as soon as JavaScript is disabled. Another advisable strategy would be to implement a HTML filter such as DOMPurify[3]. This would make it possible for HTML to be categorically filtered prior to being presented to the user. As a result it can be made sure that with the combination of CSP and filtered HTML, the chances for a bypass are minimized.

PSI-01-004 Browser: Popup Blocker Bypass via endlessipc:// Handler *(Low)*

The Psiphon app intercepts certain HTTP requests to be able to implement application-specific URL protocol handlers. One of these handlers is called "*endlessipc://*" and implements different functionalities based on the domain part of the URL. One of these functions is called "*window.open*" and, as the name suggests, it opens a new tab. The functionality can be invoked either through frames, links, or top-level navigations via JavaScript.

To protect users from malicious pop-ups, which is a standard for browsers, the app checks the *navigationType* of a new HTTP request for *UIWebViewNavigationTypeLinkClicked*.. This should indicate that the user has actually clicked a link. It was discovered, however, that it is possible to use JavaScript to emulate the click of a link, therefore bypassing the pop-up blocker without any user-interaction. Although this does not lead to a severe vulnerability, it is possible to spam the app with new tabs until it crashes.

Proof-of-Concept:

```
<!DOCTYPE html>
<body>
<script>
```

[3] https://github.com/cure53/DOMPurify

```
for (a=0;a<100;a++)
{
el = document.createElement("a");
el.href = "endlessipc://window.open/?https://cure53.de";
document.body.appendChild(el);
el.click();
}
</script>
```

Steps to reproduce:

1. Store the PoC on your webserver;
2. Open the PoC inside the Psiphon app;
3. The HTML page will open one hundred tabs, thus crashing the app without user-interaction.

File:
endless-master/Endless/WebViewTab.m

Affected Code:

```
else if ([action isEqualToString:@"window.open"]) {
 /* only allow windows to be opened from mouse/touch events, like a normal
browser's popup blocker */
 if (navigationType == UIWebViewNavigationTypeLinkClicked) {
  NSURL *newURL = [NSURL URLWithString:value];
  WebViewTab *newtab = [[[AppDelegate sharedAppDelegate] webViewController]
addNewTabForURL:newURL];
  newtab.openedByTabHash = [NSNumber numberWithLong:self.hash];
  [self webView:__webView callbackWith:@""];
```

It is recommended consider disabling the *endlessipc:// window.open/* functionality. iOS currently does not offer different *navigationType* for user- or JavaScript-initiated clicks which therefore cannot be distinguished. It could be possible to use client-side JavaScript to overwrite a "*click*" handler of all *HTMLAnchorElements* but it would not add any additional security as the website's JavaScript could either remove or bypass this protection without much effort.

PSI-01-006 Browser: WebView HTML Leaks reveal User IPs *(Medium)*

It was found that a censor may be able to accurately fingerprint and block a great number of Psiphon servers, leveraging HTML leaks in the iOS browser's webview. A malicious government could accomplish this with crafted HTML on a popular website. Alternatively, it could be possible to entice Psiphon users into visiting an attacker-controlled website. When a Psiphon user visits such a page, their real IP will be leaked to the censor. From there it is possible to correlate the received leaks with the established TCP connections for that origin's IP. Therefore, it is not difficult to acquire a very low false-positive rate for this process and determine which ones of the established obfuscated connections belong to the Psiphon servers.

Once Psiphon servers are made known, blocking them does not pose a challenge. This effectively defeats the censorship circumvention promises that Psiphon offers. A censor can further reduce the false-positives by tricking users into visiting the crafted website via the Psiphon Browser URL handlers. Implementing this attack provides censors with some assurance that the targeted users will indeed open the URL with the Psiphon iOS browser. The following HTML tag is sufficient to trigger the opening of the Psiphon app with an attacker-controlled URL.

HTML:

```
<img src="endlesshttps://cure53/exchange/962431725424/test.php" />
```

To determine possible HTTP leaks, a dedicated Github project was used at https://github.com/cure53/HTTPLeaks/blob/master/leak.html.

Successful leaks encompassed:

- */video-src*
- */video-source-src*
- */audio-source-src*

Simplified test case:

```
<video src="https://cure53.de/video-src">

<video controls>
<source src="https://cure53.de/video-source-src" type="video/mp4">
</video>

<audio controls>
<source src="https://cure53.de/audio-source-src" type="video/mp4">
</audio>
```

The following Proof-of-Concept contains the HTML tags which will send a request outside of the Psiphon tunnel. Additionally, a cookie called *psiphon_tracking* was here set with a random value. This resulted in the leaks sending the cookie along, with the same tracking ID, outside of the Psiphon tunnel. It can be inferred that the Psiphon's promise about protecting a user from stolen cookies on an insecure WiFi has been broken.

PoC:
https://cure53/exchange/962431725424/test.php

Code:
```
<?php
setcookie('psiphon_tracking', rand(0, 99999999), time() + (86400 * 30), "/");
?>
<!DOCTYPE html>
<body>
<h1>This test case shows that video and audio related resources are fetched with
the users real ip</h1>
<video src="https://cure53.de/video-src">
<video controls>
<source src="https://cure53.de/video-source-src" type="video/mp4">
</video>
<audio controls>
<source src="https://cure53.de/audio-source-src" type="video/mp4">
</audio>
```

Request:
```
GET /video-poster-2 HTTP/1.1
Host: cure53.de
Accept-Language: en-us
X-Playback-Session-Id: A0A59952-5452-44A6-ABAA-63DE9EBE8A21
Cookie: psiphon_tracking=26627618
Range: bytes=0-1
Accept: */*
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 9_3_3 like Mac OS X)
AppleWebKit/601.1.46 (KHTML, like Gecko) Version/9.0 Mobile/13G34
Safari/601.1.46/5058156304
Referer: https://cure53/exchange/962431725424/test.php
Connection: close
```

It is recommended to deploy a secure default configuration that avoids HTML leaks. This can be achieved by implementing a client-side HTML filtering by, for instance, using the DOMPurify tool. This JavaScript module can be used to sanitize a HTML string and then safely load it into the DOM. The DOMPurify is additionally highly configurable, allowing to block certain HTML tags, whitelist certain attributes, or even permit specific tags which are normally filtered. In combination with the currently deployed "Disable JavaScript" feature, it would allow the Psiphon app to have a more robust and more fine-grained restriction for any web page in place.

PSI-01-007 Browser: Psiphon User Agent leaks via Image Download *(Low)*

***Update:** This issue has been fixed and the fix was reviewed by Cure53*

The Psiphon iOS Webview defines a default iOS safari *"User-Agent"* for any HTTP request transmitted. This ensures that the presence of the Psiphon app is not leaked to a web server via the HTTP header of the *"User-Agent"*. It was discovered that the

"Download image" feature triggers HTTP requests for which *"User-Agent"* contain Psiphon information, therefore leaking the presence of the app.

As described in PSI-01-006, the permissive setting allows for Psiphon users to be identified and reduces the likelihood of false positives. The leak occurs as the application relies on the *"dataWithContentsOfURL"* function to retrieve the images. This type of content prevents setting a custom HTTP header.

Steps to reproduce:

- Open a web page which loads an image via an *img* tag;
- Keep pressing on the image with a finger until a *Menu* pop ups;
- Click "*Download image*";
- The triggered HTTP request will contain the following "*User-Agent*":

```
GET /picture.jpg HTTP/1.1" 200 51132 "-" "Psiphon/1013 CFNetwork/758.4.3
Darwin/15.5.0
```

File:
endless-master/Endless/WebViewTab.m

Affected Code:

```
UIAlertAction *saveImageAction = [UIAlertAction
actionWithTitle:NSLocalizedString(@"Save Image", @"Action title for long press
on image dialog") style:UIAlertActionStyleDefault handler:^(UIAlertAction
*action)
{
[self requestAuthorizationWithRedirectionToSettings];
NSURL *imgurl = [NSURL URLWithString:img];
[JAHPAuthenticatingHTTPProtocol temporarilyAllow:imgurl];
NSData *imgdata = [NSData dataWithContentsOfURL:imgurl];
```

It is recommended to implement the *"Download image"* feature with the *"NSMutableURLRequest"* function. "*NSMutableURLRequest"* is already used in the Psiphon app for certain features as it allows to set default headers for a HTTP request. With this approach the *"User-Agent"* leak would be eliminated.

PSI-01-008 Browser: IP Leaks via Links to external Apps *(Low)*

The Psiphon application intercepts certain HTTP requests before they are passed to the operating system. However, it does not enforce any rules for the allowed URLs. This behavior permits using iOS default URL schemes to open an external app from a Psiphon-rendered HTML page without any user-interaction. The opened app will send requests outside the Psiphon application and this behavior can be exploited in a similar way as described in PSI-01-006. Notably, it leaks the real IP address of a Psiphon user.

The following example depends only on the *map* feature of iOS to open the external maps application, yet iOS defines more default URL schemes[4].

Steps to reproduce:

1. Specify the following HTML tag in a HTML page :

```
<iframe src="http://maps.apple.com/?
q=Mexican+Restaurant&sll=50.894967,4.341626&z=10&t=s"></iframe>
```

2. Load the web page inside the Psiphon app.
3. The http://maps.apple.com URL will automatically trigger the start of the iOS "*Maps*" app.
4. This will trigger an external request outside of the Psiphon app

It is recommended to modify the currently deployed *"shouldStartLoadWithRequest"* function and implement a check for the loaded URLs able to trigger the opening of external apps. Although it is impossible have awareness of all possible URLs handled by external apps, it is highly recommended to block iOS default URL schemes[5] as well as any protocol handlers unknown to the Psiphon app. While this approach cannot block all of the possible external app requests, it adds additional security layer for the Psiphon application deployed on a default iOS system.

PSI-01-009 Server: Psiphon Server list world readable (*Info*)

It was discovered that certain server lists are world readable. This list contains 216 Psiphon servers and their corresponding configurations. If a malicious actor was to get access to this file it would be a trivial task to create an ACL or signature for their infrastructure, In other words blocking the servers and services published on that server list would not pose a challenge.

A clear recommendation here is to limit access to the server list. The aim is for attackers to require more time and skill when seeking out to get hold of the actual list. A token-based working scheme based on a user- and server-calculated hash could be one approach to achieve this. If the Psiphon binary is disassembled and debugged, it would be possible to access hash or the constructs to the hash, then swiftly proceed to the actual list. However, by implementing a token-based authorization and rotating the hashes regularly, the difficulty would be increased. Under the new premise, a malicious adversary could not get access to the server list without investing considerable time in reversing the binary. For more information about general advice, please see "Protocol Recommendations" section.

[4] https://developer.apple.com/library/content/featuredarticles...Reference/Introduction/Introduction.html
[5] https://developer.apple.com/library/content/featuredarticl...e_Reference/Introduction/Introduction.html

Cryptography-Specific Findings

Cure53 analyzed the core cryptographic components of Psiphon over the course of three days. As for a general conclusion, it should be emphasized that Psiphon's cryptographic components have been designed and engineered with unusually high quality. Findings pointing to negative outcomes were very much limited. The examined components warranted detailed comments and are thus enumerated under specific subheadings next.

Cryptographic Primitives

The core cryptographic primitives used across Psiphon, notably *Chacha20*, *Poly1305*, and *Curve25519,* are all modern and trusted. Not only are these primitives derived from the standard and trusted *Go* implementations, but they also appear to be utilized correctly. In brief, it was impossible to discern any serious issues with the way these primitives are implemented or deployed. Common pitfalls in this realm are successfully avoided by Psiphon.

Exotic Cryptographic Constructions

Psiphon employs two rarely-seen cryptographic constructions, namely OpenSSH Handshake Obfuscation and Shamir Secret Sharing for Revealing New Servers. Both were reviewed in this project and the findings are described under the respective subheadings below.

OpenSSH Handshake Obfuscation

Psiphon uses the OpenSSH protocol in order to negotiate handshakes with circumvention servers. In order for the handshake to be more resistant to DPI-based blocking attempts, it is disguised as random data using a well-known specification re-implemented in Go.

The implementation appears to be sound and robust. While RC4, a commonly weak encryption system, is used as a stream cipher for purposes of encrypting the initial handshake, it is not likely that this has negative impact on Psiphon's security goals in this instance. A switch to a stream cipher with less biased output, such as Chacha20, could be favorable yet is not necessary.

Shamir Secret Sharing for Revealing New Servers

Psiphon uses Shamir secret sharing in order to progressively reveal decryption information for servers, based on a time/usage calculation carried out on the server's behalf. No problems were detected with this implementation.

Cryptographic Protocol State and Flow

Psiphon derives its authentication flow by shipping a client that contains a static signature public key. All sensitive payloads then shipped from the server, be new server addresses or software update information, are then signed with this key. There was only one concern spotted in this realm. Specifically, the setup has all authentication for all Psiphon users centralized in a single signing key pair. Consequently, the compromise of this pair could easily delude users into shifting to an unsafe network space. Similarly, it could potentially lead to a mass malware installation.

Protocol Recommendations

Psiphon claims that blocking its server list is impossible without essentially blocking half of the Internet. This is supposed to be based in the list being hosted on a neutral large cloud provider's subdomain which cannot be blocked without the access to vital Internet resources across the web being blocked at the same time. While this premise holds, nothing seems to prevent an attacker from continually refreshing the list. This indicates that the list itself is not blocked, but the individual addresses that figure within it are affected (see PSI-001-009). This process can be automated and has a capacity to cause significant performance and circumvention roadblocks for Psiphon users globally.

It is proposed to employ a token-based proof of work scheme as a motivating example. This could potentially be used to make the problem more difficult to exploit and can be analyzed in more detail below.

- Upon an initial connection to the Psiphon network, a user is given an identifier of "*u*".
- Both the server and the client calculate *h = hash(u)*.
- In order to then fetch any server list, the client must first provide a hash *c,* which has a number of initial bits that matches the number of initial bits in *h*. The number of initial bits can be used to set a difficulty level for the proof of work function.

The above proposal is strictly a motivational example. It would presumably require a special authentication protocol to be implemented and carried out over the same cloud provider namespace.

General Notes

Cryptographic specification material provided by Psiphon for this test was very much useful yet outdated. It is highly recommended for the Psiphon Inc. to update their specification documents to accurately match the implementation. Discrepancies are present between the two dimensions at the time of writing.

Conclusions

This Cure53 assessment aimed at addressing numerous security-relevant items within the rather vast scope delineated by the Psiphon. As many as nine Cure53 testers with different skillsets and expertise were engaged in completing this project to achieve the best possible coverage. The team investigated Psiphon for a total of twenty-two days and arrived at a highly positive verdict about security being centrally embedded and maintained to high-standards within the tested project.

The Psiphon maintainers seem to approach security hands-on by delimiting the attack surface at the very core of their project. The threat model is not only very specific and narrow, but it also delivers in terms of relying on very clean and quality-driven code. Keeping security promises appears to be a top-priority for Psiphon, as evidenced by the technical specifications above, as well as a belief that security audits are executed on a regular basis. All these observations indicate maturity of the software development process at Psiphon Inc. It should be emphasized that the Cure53 sought to accomplish ideal coverage in the time permitted for this project. More general information about the covered areas and respective findings will be furnished next.

Psiphon server and client constituted main testing grounds. The Psiphon common code (*/psiphon/common* and */psiphon/server* without */psiphon/server/psinet*) was audited, together with the client/server-components (*/ConsoleClient* and */Server*). While the approach was quite quite-reaching, note that code from non-Psiphon repositories was considered out of scope. Manual inspection revealed no security issues and the quality of code is praiseworthy. This area is in a very good shape, with code characterized with technical savviness and being rooted in readability.

Another item in scope concerned the Psiphon iOS mobile application. The tests in this realm incorporate checks against file-system protections of iOS data on the application. The goal was to determine whether sensitive user-data could be leaked. Traffic analysis was carried out to gauge transport security level and map potential. Additional testing revolved around hardcoded secrets and similar issues, this time aiming to see if the application itself leaks data that might cause trouble for Psiphon Inc. and their users. Further analysis encompassed integration of the crypto core into the mobile app. Efforts proved futile here as Cure53 only discovered items developed and deployed with security in mind.

Next in line was Psiphon Library Glue, for which */MobileLibrary/Android/PsiphonTunnel* and */MobileLibrary/iOS/PsiphonTunnel* mobile components were audited, again following the premise that code from non-Psiphon repositories shall be excluded. No obvious problems were found during this brief inspection.

The Psiphon browser was also investigated and the tests focused on website-to-browser communication. Among other protocol handlers *endlessipc://* were examined, as some known risks about security and data leaks can generally be attributed to their behaviors. As tests were performed against HTTP leaks capable of unmasking a user’s IP, some issues surfaced. Finally, studying security features offered by Psiphon centered on “no JavaScript” mode, pop-up blocker, and other ways of harming user-privacy and IP masking.

In the area of server communication, protocol and configuration, the analyses looked for information disclosure in the windows binary. This led to the discovery of the server list, backup domains and son on. Some reference to the implementation were found during the VPN configuration analysis. These were present in sources and configuration despite being no longer in use. The Cure53 team briefly looked at the server binaries and sources to get a basic understanding for the project and look for possible weak configurations, but there is nothing to report on that. The tests further determined the absence of the known vulnerabilities in OpenSSH and ensured that no misconfigurations leading to security-related issues can be demonstrated. Same applied to the cloud provider instance, particularly around configuration issues in the storage buckets and common configuration mistakes, as well as scraping and enumerating publicly available resources. A thorough analysis of the protocol used throughout the configuration and infrastructure was performed. To facilitate testing, Cure53 developed a small tool to track blocking capabilities on a Windows 7 workstation with the Psiphon client installed and running.

Finally, substantial attention was dedicated to the implementations of cryptographic protocols. A close examination of those was covered in the separate section earlier in this report, but overall reinforced the positive outcome of this security assessment. Despite investing considerable time and personnel resources into attempting a compromise, the Psiphon components in scope held up to scrutiny and presented themselves strong and robust in face of adversarial efforts. The bottom line is that no noteworthy security risks could be unveiled. Given the threat model issued by the Psiphon maintainers, the security is sound and adheres to keeping the security promises at stake. On the one hand, it is impossible to fully predict and completely avoid all routes that adversaries can take to deobfuscate, obfuscate and block individual users or servers from being able to surf or function properly. On the other hand, the Psiphon suite clearly does very well in handling risks and delivers what it factually proposes and offers.

Cure53 would like to thank Mike Fallone and Irv Simpson of Psiphon Inc. for their excellent project coordination, support and assistance, both before and during this assignment.