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 Conduit Library 06.2023

Cure53, Dr.-Ing. M. Heiderich, MSc. N. Krein, Dr. M. Conde, MSc. D. Weißer, MSc. F. Fäßler, BSc. B. Walny, MSc. R. Peraglie, M. Pedhapati

Index

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

Introduction

“Psiphon Inc. is a company based in Toronto, producing open-source multi-platform software that helps over 3 million people every week connect to content on the Internet. We’re a team focused on delivering the best software we can, introducing new products regularly and making sure we develop to the needs of our constantly growing global audience.”

From https://www.psiphon.ca/en/about.html

This report describes the results of a penetration test and source code audit against the Psiphon Conduit library project and its core components. The work was requested by Psiphon Inc., in May 2023. Cure53 carried out the project in the frames of the long-term cooperation with Psiphon.

Registered as *PSI-06,* the examination was completed on schedule. Eight senior testers formed a team assigned to this project’s preparation, execution and finalization. All tasks were completed in June 2023, namely in CW23 and CW24. A total of thirty-two days were invested to reach the coverage expected for this project. It should also be explained that even though Cure53 evaluated various components run and managed by Psiphon, *PSI-06* marks the first assessment of the *inproxy* Conduit library.

Given the type of work necessitated by the project’s goals, all tasks were tracked under a single work package (WP):

- **WP1**: Source code audits & penetration tests against Psiphon Conduit library project

Cure53 was provided with sources, a list of the relevant core-components, as well as all further means of access required to complete the tests, the methodology chosen here was white-box.

For the sake of transparency, Cure53 wishes to add that the originally scheduled audit included a second WP. That second assessment was aimed at related and unrelated Psiphon projects, but was ultimately removed because the scope of the current WP was deemed to be sufficient for this iteration.

All preparations were done in late May and early June 2023, namely CW22, so Cure53 could have a smooth start. Communications during the test were done using a dedicated shared Slack channel connecting the teams of Psiphon and Cure53.

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

Discussions were productive and needed, especially in the face of the complexities characterizing the components examined during this project. Still, the scope was well-prepared and clear, hence no noteworthy roadblocks were encountered during the test.

Cure53 gave frequent status updates about the test and the related findings. While live-reporting was not specifically requested for this audit, tickets have nevertheless been shared prior to the finalization of this report.

The Cure53 team managed to get very good coverage over the WP1 targets. Of just five findings, four were classified to be security vulnerabilities and one was deemed to represent a general weakness with lower exploitation potential.

The impression gained throughout this test is generally positive, which is reflected by the small number of findings. Additionally, it can be positively acknowledged that the overall severity of the findings did not exceed a *Medium* level, indicating that the Conduit library and components have already received proper hardening against various threats and attack scenarios.

In essence, the Psiphon team did an excellent job ensuring that their *inproxy* Conduit library has a good security posture, with only some more work and improvements to be done. However, it should be noted that it might be worth retesting the components as soon as they are live. Cure53 believes that a test-environment with test-user accounts should be available to guarantee an even better overview of the library after it is fully integrated into Psiphon.

The report will now shed more light on the scope and test setup as well as the available material for testing. After that, all findings are listed in chronological order, first the spotted vulnerabilities and then the general weakness. Each finding will be accompanied with a technical description, a PoC where possible as well as mitigation or fix advice. Finally, a conclusion in which Cure53 elaborates on the general impressions gained throughout this test are presented. Recommendations for the Psiphon *inproxy* Conduit library project and its core components are reiterated in the last section as well.

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

Scope

- **Penetration tests & source code audits against Psiphon Conduit library**
  - **WP1**: Source code audits & penetration tests against Psiphon Conduit library
    - **Sources:**
      - https://github.com/Psiphon-Labs/psiphon-tunnel-core/tree/inproxy/psiphon/common/inproxy
    - **Relevant core components:**
      - **Package overview:**
        - https://github.com/Psiphon-Labs/psiphon-tunnel-core/blob/inproxy/psiphon/common/inproxy/api.go
      - **Broker:**
        - https://github.com/Psiphon-Labs/psiphon-tunnel-core/blob/inproxy/psiphon/common/inproxy/broker.go
      - **BrokerClient (run by client and proxy):**
        - https://github.com/Psiphon-Labs/psiphon-tunnel-core/blob/inproxy/psiphon/common/inproxy/brokerClient.go
      - **Client:**
        - https://github.com/Psiphon-Labs/psiphon-tunnel-core/blob/inproxy/psiphon/common/inproxy/client.go
      - **DialParameters (tactics and replay integration):**
        - https://github.com/Psiphon-Labs/psiphon-tunnel-core/blob/inproxy/psiphon/common/inproxy/dialParameters.go
      - **NAT Discovery:**
        - https://github.com/Psiphon-Labs/psiphon-tunnel-core/blob/inproxy/psiphon/common/inproxy/discovery.go
      - **Support for DTLS ClientHello randomization:**
        - https://github.com/Psiphon-Labs/psiphon-tunnel-core/blob/inproxy/psiphon/common/inproxy/dtls.go
      - **Client/proxy matching engine used by broker:**
        - https://github.com/Psiphon-Labs/psiphon-tunnel-core/blob/inproxy/psiphon/common/inproxy/matcher.go
      - **NAT classifications:**
        - https://github.com/Psiphon-Labs/psiphon-tunnel-core/blob/inproxy/psiphon/common/inproxy/nat.go
      - **Session obfuscation layer:**
        - https://github.com/Psiphon-Labs/psiphon-tunnel-core/blob/inproxy/psiphon/common/inproxy/obfuscation.go
      - **UPnP-IGD, etc.:**
        - https://github.com/Psiphon-Labs/psiphon-tunnel-core/blob/inproxy/psiphon/common/inproxy/portmapper.go

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

- **Proxy:**
  - https://github.com/Psiphon-Labs/psiphon-tunnel-core/blob/inproxy/psiphon/common/inproxy/proxy.go
- **Marshaling:**
  - https://github.com/Psiphon-Labs/psiphon-tunnel-core/blob/inproxy/psiphon/common/inproxy/records.go
- **Server (used by psiphond):**
  - https://github.com/Psiphon-Labs/psiphon-tunnel-core/blob/inproxy/psiphon/common/inproxy/server.go
- **Secure Noise session (for client/proxy<->broker and broker<->server):**
  - https://github.com/Psiphon-Labs/psiphon-tunnel-core/blob/inproxy/psiphon/common/inproxy/session.go
- **WebRTC conn (used by client/proxy):**
  - https://github.com/Psiphon-Labs/psiphon-tunnel-core/blob/inproxy/psiphon/common/inproxy/webrtc.go

- **Test-supporting material was shared with Cure53**
- **All relevant sources were shared with Cure53**

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

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, all tickets are given a unique identifier (e.g., *PSI-06-001*) to facilitate any future follow-up correspondence.

PSI-06-001 WP1: Unvalidated STUN transaction ID facilitates NAT confusion *(Low)*

***Note**: The issue was fixed and the fix was verified by Cure53 who had access to the diff.*

It was found that the Psiphon STUN client failed to validate the Transaction ID or IP address found in the *OTHER-ADDRESS* field of the returned STUN response. Hence, attackers who are responding to the STUN request of the client can supply a local or internal IP address which the STUN client will send a UDP packet to.

As a result, the Conduit clients and proxies are at risk of being manipulated via sending of UDP packets to internal network services. However, as the contents of the UDP packet cannot be controlled by the attackers, this issue is only rated as *Low*.

Steps to reproduce:

1. Drop the delivered Golang test file *cure53_stun_test.go* into the *./psiphon/common/inproxy/cure53_stun_test.go* directory.
2. Run the test by invoking from the *root* Go directory:
   `go test -v -run TestSTUNTransactionConfusion`

An excerpt from the execution of the test-case can be seen in the following snippet. The test will emulate an attacker answering a NAT and discovering a STUN client with two forged STUN response packets that do not carry a valid transaction ID. This illustrates that the client successfully performs a UDP dial to an internal service listening on the local address *127.0.0.1:36033*.

Excerpt from *shell*:

```
=== RUN   TestSTUNTransactionConfusion
cure53_stun_test.go:32: StunServer: 127.0.0.1:47639
    StunClient: 127.0.0.1:43783
    Attacker: 127.0.0.1:46325
    Service: 127.0.0.1:36033
cure53_stun_test.go:42: Server now attempting to read Message 1/2
cure53_stun_test.go:47: Server received 20 bytes
cure53_stun_test.go:54: Server received STUN packet with transaction ID: "\xff
<\xb8-\x8b\xceNj\x95t\x17"
```

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

```
cure53_stun_test.go:71: Attacker sent STUN message with different Transaction
ID: "0123456789AB"
cure53_stun_test.go:42: Server now attempting to read Message 2/2
cure53_stun_test.go:47: Server received 20 bytes
cure53_stun_test.go:54: Server received STUN packet with transaction ID: "e\
x14,\xb6X\xb1\x99\xcb\xfd\x0fJ/"
cure53_stun_test.go:71: Attacker sent STUN message with different Transaction
ID: "0123456789AB"
cure53_stun_test.go:88: A connection was established to the service by the
client 127.0.0.1:43783
cure53_stun_test.go:95: Client has performed a failed NAT discovery
--- PASS: TestSTUNTransactionConfusion (2.00s)
```

Within the following source code, it can be observed that the *doSTUNRoundTrip* function generates a new transaction ID but discards the result by storing it in the blank identifier (_) variable.

Affected file:
psiphon/common/inproxy/discovery.go

Affected code:

```
func doSTUNRoundTrip(
    request *stun.Message,
    conn net.PacketConn,
    remoteAddress string) (*stun.Message, bool, error) {
    remoteAddr, err := net.ResolveUDPAddr("udp", remoteAddress)
    if err != nil { [...] }

    _ = request.NewTransactionID()
    _, err = conn.WriteTo(request.Raw, remoteAddr)
    if err != nil { [...] }

    conn.SetReadDeadline(time.Now().Add(discoverNATRoundTripTimeout))
    var buffer [1500]byte
    n, _, err := conn.ReadFrom(buffer[:])
    if err != nil { [...] }

    response := new(stun.Message)
    response.Raw = buffer[:n]
    err = response.Decode()
    if err != nil { [...] }

    return response, false, nil
}
```

Psiphon should validate the transaction ID of the STUN response, ensuring that it matches the one sent with the request. In addition, any potentially dialed IP address - especially from the *OTHER-ADDRESS* field of a STUN message - must be validated to

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

not be local, internal or reserved. Mitigation could be achieved with Psiphon's *isBogon* function to maintain code-reuse. Henceforth, attackers would have to know the transaction ID beforehand in order to confuse the NAT-type-mapping. Misleading of the UDP packets to internal or reserved IP addresses would be prevented by the filter.

PSI-06-002 WP1: DoS via missing rate-limit on *handleProxyAnnounce* (Medium)

***Note**: The issue was fixed and the fix was verified by Cure53 who had access to the diff.*

While auditing the proxy and client matching, it was noted that announcements and offers are handled via queues. These queues were found to have an upper bound. Due to missing rate-limiting, malicious users could fill the queue of the proxy announcements with bogus proxies, such that the full queue will reject any other legitimate proxy. The result would be a complete DoS on the targeted broker. The affected lines of code are shown below.

Affected file:
psiphon-tunnel-core/psiphon/common/inproxy/matcher.go

Affected code:
```
matcherAnnouncementQueueMaxSize = 100000
[...]
func (m *Matcher) Announce(..)
   [...]
    m.addAnnouncementEntry(announcementEntry)
```

Affected file:
psiphon-tunnel-core/psiphon/common/inproxy/broker.go

Affected code:
```
func (b *Broker) handleProxyAnnounce(
[..]
       <no rate-limiting>
[..]
       clientOffer, err = b.matcher.Announce(
```

Since the project for now relies on the caller of the *inproxy* broker component to handle rate-limiting, the real impact remains unclear. Cure53 recommends adding a rate-limiting mechanism that is specific to proxy announcements, thus restricting given proxyIPs to only announce a small, fixed amount of proxies. For a bypass, this would require attackers to have access to a greater number of IP addresses.

Further, the queues could have a higher limit. If the memory impact is too high, however, this could be replaced with other data storing mechanisms, such as a database.

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

PSI-06-003 WP1: Potential DoS by draining queues via *ClientOfferRequests* *(Low)*

***Note**: The issue was fixed and the fix was verified by Cure53 who had access to the diff.*

Issue PSI-06-002 describes a DoS issue based on filling up the announcement queue. Similar to that problem, this ticket explores the opposite effect of draining the queue.

In this scenario, a malicious client can spam *ClientOfferRequests,* which adds the client to the offer queue. The client offers and proxy announcements are then matched by the matcher, which removes the announcement from the queue. This means a malicious client can spam offer entries, thus keeping the announcement queue as empty as possible.

Affected file:
psiphon-tunnel-core/psiphon/common/inproxy/matcher.go

Affected code:

```
func (m *Matcher) Announce(
	ctx context.Context,
	proxyAnnouncement *MatchAnnouncement) (*MatchOffer, error) {
	// [...]
	m.addAnnouncementEntry(announcementEntry)
	// Await client offer.
	var clientOffer *MatchOffer
	select {
	case <-ctx.Done():
		m.removeAnnouncementEntry(announcementEntry)
		return nil, errors.Trace(ctx.Err())

	case clientOffer = <-announcementEntry.offerChan:
	}

func (m *Matcher) Offer(
	ctx context.Context,
	clientOffer *MatchOffer) (*MatchAnswer, *MatchAnnouncement, error) {
	// [...]
	m.addOfferEntry(offerEntry)
```

In this DoS scenario, the malicious client tries to overwhelm the proxy announcements. If that is achieved, then most real clients will not be able to get valid proxy offers. While this DoS is not always effective, it can still be disruptive. Generic IP based rate-limit could be added to prevent a single malicious client from spamming too many *ClientOfferRequests*. Alternatively, the offer queue itself could be restricted to only allow one client with a given IP. In the case of a client using CGNAT, a malicious client would not be fully prevented from disrupting other clients, but the impact would have been reduced to only one affected IP.

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

PSI-06-004 WP1: DPI via randomized *Hellos* in DTLs *(Medium)*

The Psiphon *inproxy* WebRTC implementation offers an anti-DPI feature to randomize the TLS *ClientHello* and *ServerHello* messages, with the aim of evading DPI using DTLs fingerprinting. However, it was noticed that the evasion techniques can be used to fingerprint the Psiphon DTLs by comparing multiple *ClientHello* and *ServerHello* DTLS packets. This would translate to checking if they are randomized all the time.

As for the latter, it is not the case with popular WebRTC communications, including the modern browsers Chrome and Firefox, as well as Skype, Discord and Google Meet. This means an adversary can check a single or multiple *ClientHello* messages for containing randomized TLS extensions and randomized TLS extension parameters. Based on that, they could identify Psiphon connections.

Within the following excerpt, the *ClientHello* messages of DTLS handshakes captured in Firefox WebRTC sessions in **.pcapng* files are inspected with the program *tshark*. It can be seen that neither of the two browsers randomizes the majority of the DTLS handshake, resulting in the same byte-constant contained in the packet. This byte-constant can then be used to filter unknown DTLS participants like Psiphon peers, as illustrated in the last part of the excerpt. Other programs can be fingerprinted and sieved out in a similar way.

Excerpt from *shell*:

```
$ tshark  -r dtls_handshake_firefox.pcapng -x  'dtls.handshake.type == 1'
0000  00 00 03 04 00 06 00 00 00 00 00 00 39 fb 08 00   ............9...
[...]
0070  a9 cc a8 c0 0a c0 09 c0 13 c0 14 01 00 00 6a 00   ..............j.
0080  17 00 00 ff 01 00 01 00 00 0a 00 08 00 06 00 1d   ................
0090  00 17 00 18 00 0b 00 02 01 00 00 10 00 12 00 10   ................
[...]
$ tshark  -r dtls_handshake_chrome.pcapng -x  'dtls.handshake.type == 1'
0000  00 00 03 04 00 06 00 00 00 00 00 00 e1 52 08 00   .............R..
[...]
0080  35 01 00 00 44 00 17 00 00 ff 01 00 01 00 00 0a   5...D...........
0090  00 08 00 06 00 1d 00 17 00 18 00 0b 00 02 01 00   ................
[...]
$ tshark  -r dtls_handshake_in_proxy_clients.pcapng  'dtls.handshake.type == 1
&& !(udp.payload contains
00:17:00:00:ff:01:00:01:00:00:0a:00:08:00:06:00:1d:00:17:00:18)'
 6957 161.925806803 10.0.2.15 → 10.0.2.15      DTLS 179 Client Hello
 6992 161.926979410 10.0.2.15 → 10.0.2.15      DTLS 175 Client Hello
13181 162.008366077 192.168.56.102 → 192.168.56.102 DTLS 181 Client Hello
13257 162.009754093 127.0.0.1 → 127.0.0.1      DTLS 175 Client Hello
[...]
24790 162.224478799 127.0.0.1 → 127.0.0.1      DTLS 175 Client Hello
52379 168.227620643 127.0.0.1 → 127.0.0.1      DTLS 175 Client Hello
```

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

It is recommended to offer the same number and order of ciphers, extensions and their parameters, as it would be expected in a widely-used modern browser or similar program. The handshake fingerprints should be indistinguishable from the regular program. This could also be achieved by using and running the browser or program while steganographically obfuscating the information within the application's data stream[1].

[1] https://www.cs.cornell.edu/~shmat/shmat_oak13parrot.pdf

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

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.

PSI-06-005 WP1: No DTLS randomization in Go vendor folder *(Info)*

***Note**: The issue was fixed and the fix was verified by Cure53 who had access to the diff.*

During the inspection of PSI-06-004, it was noticed that the latest commit *9998e8d67c56eab3fabdfc543e13732040af49d4* of the branch *in_proxy* is shipped with an outdated DTLS implementation within the *vendor* directory. This means that it does not make use of the DTLS randomizations.

The *vendor* directory is the preferred directory for *go build* and *go test* commands. Therefore, it renders the DTLS changes made within the *./replace/dtls* directory unapplied.

Excerpt from *shell*:

```
$ diff vendor/github.com/pion/dtls/v2/flight1handler.go
replace/dtls/flight1handler.go
[...]
120a125,200
>     cipherSuites := cipherSuiteIDs(cfg.localCipherSuites)
>
>     // [Psiphon]
>     // Randomize ClientHello
>     seed, err := inproxy_dtls.GetDTLSSeed(c.LocalAddr())
[...]
>       PRNG.Shuffle(len(cipherSuites), func(i, j int) {
>             cipherSuites[i], cipherSuites[j] = cipherSuites[j],
cipherSuites[i]
>       })
>       cipherSuites = cipherSuites[:cut(len(cipherSuites))]
```

It is recommended to synchronize the dependencies to the vendor directory before committing them to the Psiphon core repository. The novel DTLS intrinsics should clearly be implemented.

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

Conclusions

As already mentioned in the *Introduction,* the overall impression of the *inproxy* integration to Psiphon is quite positive. During this *PSI-06* project, only more or less minor flaws could be spotted and confirmed by Cure53. Most of the findings can be considered as additional hardening mitigations.

Looking at the list of the issues from this June 2023 examination, Psiphon should first focus on preventing DoS vectors uncovered in this test. Next, fixing strategies should be developed and deployed towards one DPI vector that could allow stronger censorship enforcement and poses risks in terms of bypassability. Fortunately, none of the findings have direct implications for the privacy and security of the Psiphon users.

While the overall impression was of course positive, the audit itself was rather difficult to complete. This stems from the current *inproxy* solution not being "live" yet, which means that no “finished” Psiphon client was ready for usage within the frames of the audit. Instead, dynamic testing, for example to observe network traffic, had to be conducted through various unit-tests that spun up a working network of a broker and a bunch of clients and proxies.

Moreover, the dynamic testing additionally required extended unit tests with customized code. For those unfamiliar with Psiphon's complex codebase, this is not a trivial task. Nevertheless, thanks to many helpful discussions with the Psiphon developers, Cure53 tried to make the best out of the situation and conducted a fruitful audit.

The *PSI-06* audit revolved around various security claims that the *inproxy* solution offers. For example, proxies should not see the client traffic within the relayed Psiphon tunnel. Similarly, brokers verify that client destinations are valid Psiphon servers only, so proxies cannot be misused for non-Psiphon relaying. Many of these claims are jotted down in various comments throughout the source code of the *inproxy* solution. Cure53 invested their efforts into correctly verifying these claims and making sure that the underlying code reflects them.

Lots of attention was given to reviewing the different message handlers implemented by the broker, as they offer the largest attack surface. This includes the message obfuscation layer implementation, the unmarshaling and parsing, as well as the implemented logic itself.

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

From a cryptographic perspective, the obfuscation layer built above the Noise protocol to add an anti-replay mechanism was found to be implemented by applying an AES-GCM encryption of the initiator timestamp. In addition, there was a packet using a different key for each direction of the communication from the initiator to the responder, or vice versa.

Nonce reuse, which has a negative security impact for AES-GCM and thus for obfuscation, is avoided via the maintenance of a replay history by the verifier. These two keys are derived from a 32-bytes *root* obfuscation secret using a different but fixed context string. These can be used as a salt, depending on the direction, which in particular implies that the packet obfuscation keys are deterministic for the same root obfuscation secret (which has a configured time). Although this layer only provides obfuscation and this is not an issue, it should be revisited in the future.

Regarding the Noise protocol, instanced using the XK handshake pattern, it has been verified that the surface attack does not increase that of the protocol itself, which is already verifiably secure.

The logic of message handlers were first reviewed in isolation. Cure53 explored whether they implemented any insecure functionality leading to injection issues. No such flaws could be found. Then, the shared states and interactions between messages were reviewed. This led to several Denial-of-service issues.

Specifically, it was found that the shared queue has a maximum limit which could be reached, severely disrupting the broker (PSI-06-002). After that, it was noticed that the inverse also leads to a similar DoS issue. A malicious client could drain the proxy offer requests, leading to regular clients not receiving any proxy entries (PSI-06-003).

It appears that a generic IP-rate-limit in front of the broker could solve most of these DoS issues. Alternatively, the queue could allow only one client or proxy IP at a time. This could cause some issues in CGNAT environments, however, the DoS then only affects clients in the same CGNAT. Unfortunately, DoS protections often lead to their own DoS issues.

Compared to the existing Psiphon proxy solution with regard to the control over the proxy infrastructure, the new *inproxy* implementation intends to make use of a larger network of voluntary proxies. The new solution is giving up some of the availability guarantees, but has the potential to gain more anti-censorship abilities. Cure53 understands this trade-off, especially with regard to the DoS issues. To that end, it is understood that a perfect solution likely does not exist and DoS might end up being acceptable.

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

Still, there are a lot of pitfalls that can arise during the integration of the *inproxy* implementation into the rest of the Psiphon client and server. The *inproxy* implementation expects certain dial parameter configurations that ship with certain security constraints that must be obeyed by callers. In a similar way, certain mitigations like API rate-limits are expected to be implemented by parties in production. For this reason, it is strongly advisable to validate those constraints in a subsequent security audit, namely once the *inproxy* is fully integrated into the Psiphon client and server.

Again, the given *inproxy* solution was evaluated positively and Cure53 can confirm that, with the minor exceptions discussed as a result of this *PSI-06* project, all security claims of Psiphon could be verified.

Cure53 would like to thank Rod Hynes, Joe Arshat, Mike Fallone, Irv Simpson and everyone else from the Psiphon Inc. team for their excellent project coordination, support and assistance, both before and during this assignment.