

REGRESSION TESTING

Project Name: Talaria VPN Protocol Security Test

Prepared for: LightningLink Networks Pte. Ltd.

Date: October 21, 2026

Vantage Point Security Pte Ltd

Report Id: VPQ-2026000549

Version: 1.1

Regression Testing

Author: Azka Ainul Maarij

CREST Id: 1973620492



Table of Contents

1. Executive Summary	3
1.1. Project Overview	3
2. Project Details	6
2.1. Project Scope	6
2.2. Status	7
2.3. Contact Information	7
2.4. Version History	7
3. Risk Assessment	8
3.1. Summary of Vulnerabilities	8
3.2. Vulnerability Statistics	9
4. Detailed Description of Vulnerabilities	10
4.1. Client downlink writes do not validate that payload is one complete IP packet	10
4.2. Client TLS configuration preserves unsafe entropy, clock, and key-log hooks	12
4.3. Missing minimum RSA certificate strength	14
4.4. TaloriaTCP accepts non-canonical SOCKS session credentials	16
5. Appendix	18
5.1. Disclaimer	18
5.2. Risk Rating	18

1. Executive Summary

1.1. Project Overview

Vantage Point Security Pte Ltd was engaged by LightningLink Networks Pte Ltd to conduct a combined white-box source code review and protocol-level penetration test of the Talaria reference implementation at commit 5a9a7f919ad3a64bcbf46df0c638cd85931d68af. The retest was performed against commit 6d4df662d59f7870e3f33e22892f9c8b068aca6a. The assessment considered the end-to-end security properties of the TalariaUDP and TalariaTCP implementations against applicable regulatory expectations and security best practices. It combined manual and tool-assisted review of the in-scope production code with dynamic validation through approved production interfaces and, where expressly approved, non-assessed test certificates, authentication fixtures, demos, runners, command-line entry points, harnesses, and lab configurations used solely to exercise the production implementation.

Secure Tunnel Establishment.

Vantage Point examined the TalariaUDP and supported TalariaTCP tunnel-establishment paths, including TLS handshake processing, certificate-chain and server-identity validation, trust configuration, protocol-level authentication gating, session creation, early-data handling, and fail-closed behaviour following trust or authentication failure. Source code review traced these decisions from connection acceptance through creation of a usable session. Dynamic testing exercised invalid, expired, malformed, wrong-identity, and unknown-CA certificates, invalid chains, interrupted handshakes, failed protocol authentication, early traffic, and traffic following trust or authentication failure. Within the approved runtime paths and test conditions, invalid trust or authentication material did not produce a usable authenticated session or authorised forwarding path. The assessment nevertheless identified defence-in-depth observations concerning caller-provided TLS entropy, clock and key-log hooks, minimum RSA certificate strength, and canonical TalariaTCP SOCKS credentials, these did not demonstrate an unauthorised tunnel-establishment path under the reviewed trust model.

Traffic Protection.

The review assessed how TalariaUDP and TalariaTCP use Go TLS and related cryptographic facilities to provide confidentiality and integrity, including key and session handling, nonce and counter assumptions, sensitive-data boundaries, certificate policy, and direct logs, diagnostic output, panic traces, and error paths produced by the in-scope Talaria code. Dynamic validation used controlled marker traffic, external packet capture, and inspection of direct Talaria output to check for plaintext inner traffic, keys, tokens, credentials, session material, or sensitive configuration values. No such sensitive material was observed outside the expected encrypted endpoint context in the exercised paths. The TLS-configuration and RSA-strength observations are relevant hardening opportunities for this objective, but no weakness in the underlying TLS cryptographic primitives was identified.

Session Binding and Lifecycle, and Forwarding Control and Isolation.

Source review covered session creation, binding, timeouts, reconnect and close handling, keepalives, session-table and resource cleanup, replay and duplicate handling assumptions, source rebinding, state transitions, mux-flow binding, destination-policy enforcement, and the order in which authentication, session and policy checks occur before relay or tunnel forwarding. Dynamic testing included replayed, duplicated, out-of-order and stale traffic, stale keepalives, traffic after disconnect or timeout, repeated reconnect, source rebinding, off-path UDP injection, session or flow reuse after close, traffic before a valid session, unauthorised destinations, cross-session reuse, and mux-flow collision scenarios where applicable. Within the exercised TalariaUDP and supported TalariaTCP paths, testing did not confirm stale-session acceptance, cross-session forwarding, forwarding after close or timeout, destination-policy bypass, or unsafe session-state mutation. The non-canonical credential and client downlink packet-validation observations identify opportunities to make the authentication and forwarding boundaries more explicit and robust, but did not establish a demonstrated cross-session or policy-bypass vulnerability.

Anomalous and Adversarial Input Handling, and Secure Failure and Resilience.

The assessment reviewed production parsers and handlers for wire-format messages, raw-data headers, UDP mux packets, checksums, MTU and offset handling, TalariaTCP stream and obfuscation parsing, TalariaUDP packet handling, cross-protocol dispatch, failure propagation, resource bounds, goroutine and queue lifecycles, forced-close paths, reconnect loops, and exceptional-condition logging. Dynamic validation applied malformed, truncated, oversized, tampered and wrong-protocol inputs, invalid lengths and checksums, fragmented or coalesced TCP delivery, partial headers and payloads, disconnect during partial input, cleartext presented to TLS paths, TLS-like input presented to non-TLS paths, repeated reconnect and forced close, timeout, packet-buffer pressure, race-aware workloads, and bounded reproducible fuzz-style and resource-pressure tests. The exercised paths rejected or closed invalid inputs without a confirmed exploitable crash, panic, parser confusion, forwarding-before-validation condition, data race, or uncontrolled resource growth. The downlink packet-completeness observation shows where validation should be strengthened at the client TUN boundary. Availability and DDoS-style resilience validation was deliberately limited to controlled, reproducible, lab-safe pressure and concurrency conditions, an Internet-scale or production distributed-denial-of-service exercise was not performed.

The six Security Objectives were therefore addressed through both workstreams where approved runtime exposure was available. In-scope paths that could not be reached through an approved production interface or expressly approved non-assessed execution aid were assessed at source-code level only and are subject to the corresponding runtime limitation. These boundaries mean that conclusions apply only to the reviewed commit, supported protocol paths, approved operating systems, supplied materials, and documented test conditions; they should not be interpreted as validation of excluded platform or production infrastructure behaviour.

The assessment identified four Observational findings representing implementation-hardening opportunities rather than demonstrated vulnerabilities with a direct impact on confidentiality, integrity, or availability under the reviewed trust model. They concern exact client downlink packet validation, caller-provided TLS entropy, clock and key-log hooks, minimum RSA certificate strength, and canonical TalariaTCP SOCKS session credentials. Addressing them would further strengthen input validation, TLS configuration hygiene, certificate policy, authentication canonicalisation, and protocol robustness.

The engagement is documented in a single consolidated technical report combining the white-box source code review and protocol-level penetration-testing results, including scope, evidence, and conclusions for all six Security Objectives. Subject to the stated scope and testing limitations, the assessment did not identify a demonstrated flaw that undermines the security design of the Talaria VPN protocol or its underlying cryptographic primitives, and the reviewed design and use of cryptography were found to be aligned with security best practices.

2. Project Details

2.1. Project Scope

Project / App Name	Talaria VPN Protocol Security Test
Testing Window	8 September 2026 to 24 September 2026
Pentest ID	VPQ-2026000549
Items Completed	- TalariaVPN Protocol Source code: Talaria (https://github.com/X-VPN/Talaria) - 5a9a7f919ad3a64bcbf46df0c638cd85931d68af (original assessment) - TalariaVPN Protocol Source code: Talaria (https://github.com/X-VPN/Talaria) - 6d4df662d59f7870e3f33e22892f9c8b068aca6a (retest) - TalariaVPN Protocol-Level Penetration Testing
Scope Details	WP1: Source Code Review and Security Assessment of the Talaria VPN Protocol Implementation Assessment Target: The Talaria VPN reference implementation with the agreed frozen tag and exact commit hash. Key Focus: - Security of the VPN protocol implementations supported in Talaria. - Security-relevant core code paths in the reference client and server. - Session management, authentication, packet handling, traffic forwarding, and secure failure behaviour. - Security integration with cryptographic and TLS facilities and third-party dependencies. - Dynamic security validation of the in-scope implementation, informed by source code understanding. WP2: Security Review of TLS Usage in Talaria Assessment Target: Talaria's use of TLS facilities, including Go TLS, as well as encryption and session handling in the TalariaUDP and TalariaTCP protocols. Key Focus: - Whether TLS, certificate validation, server identity verification, and trust configuration are used correctly.

	- Whether mechanisms such as encryption, integrity protection, session keys, and nonce/counter handling are used securely. - Handling of invalid certificates, handshake failures, anomalous encrypted traffic, replay, tampering, and downgrade-style inputs.
Issue Closing Date	16 October 2026 - Client downlink writes do not validate that payload is one complete IP packet [Observational] - Client TLS configuration preserves unsafe entropy, clock, and key-log hooks [Observational] - Missing minimum RSA certificate strength [Observational] - TalariaTCP accepts non-canonical SOCKS session credentials [Observational]

2.2. Status

Component	Review Type	Status
Source Code Review + VPN Protocol-Level Penetration Test	Source Code Review + Penetration Test	Completed

2.3. Contact Information

The following personnel were involved in this assessment and may be contacted for any queries regarding this report.

Name	Email	Signature
Jia Wen Zheng [Crest ID: 5442227827]	JiaWen.Zheng@vantagepoint.sg	<i>Jia Wen Zheng</i>
Azka Maarij [Crest ID: 1973620492]	azka.maarij@vantagepoint.sg	<i>Azka Maarij</i>

2.4. Version History

Date	Version	Release Name	Author
28 September 2026	1.0	Final Report	Azka Ainul Maarij
21 October 2026	1.1	Regression Testing	Azka Ainul Maarij

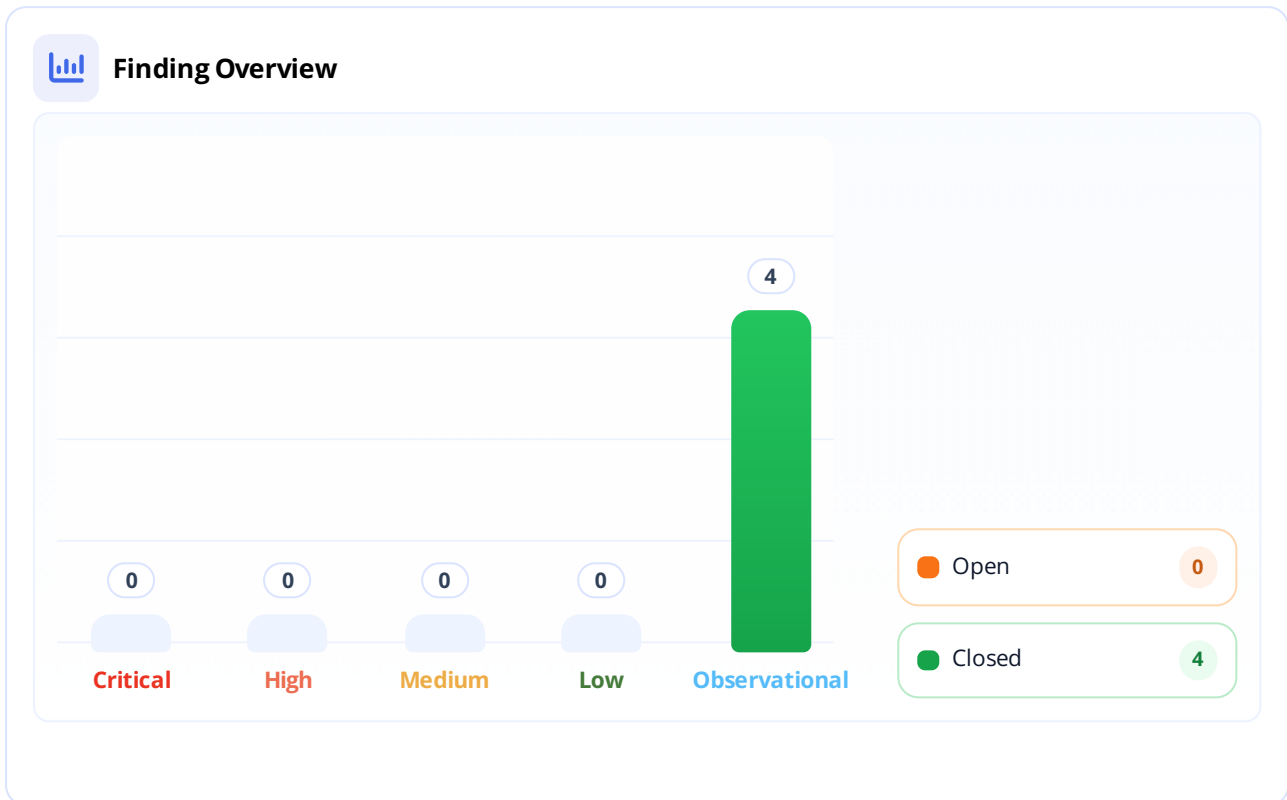
3. Risk Assessment

This chapter contains an overview of the vulnerabilities discovered during the project. The vulnerabilities are sorted based on their score in descending order. The category OBSERVATIONAL refers to vulnerabilities that have no risk score and therefore have no immediate impact on the system.

3.1. Summary of Vulnerabilities

No.	Vulnerability Title	Risk Score	Status
1.	Client downlink writes do not validate that payload is one complete IP packet	Observational	Closed
2.	Client TLS configuration preserves unsafe entropy, clock, and key-log hooks	Observational	Closed
3.	Missing minimum RSA certificate strength	Observational	Closed
4.	TalariaTCP accepts non-canonical SOCKS session credentials	Observational	Closed

3.2. Vulnerability Statistics



4. Detailed Description of Vulnerabilities

4.1. Client downlink writes do not validate that payload is one complete IP packet



Observational / Score: 0.0

Closed

CVSS:4.0/AV:N/AC:L/AT:P/PR:H/UI:N/VC:N/VI:N/VA:N/SC:N/SI:N/SA:N

Vulnerability Raised: 22/09/2026

BACKGROUND

After login, Talaria sends protected VPN data to the client TUN. Before that write, the client should check that the data is exactly one full IPv4 or IPv6 packet.

DESCRIPTION

The VPN decoder checks only the 13-byte Talaria data header. It treats every byte after that header as packet data. Client receive paths then send non-empty data to the TUN without checking that it is one full IP packet. Tests showed that the decoder and shared copy code accepted a one-byte non-IP value. Source review links this code to all client receive paths. The test did not send malformed data through a real operating-system TUN.

Talaria-main/protocol/vpnpacket/protocol_decode.go:23-29

```
case CmdData, CmdDataReturn:
    if len(b) < dataHeaderOverHeadV1 {
        return fmt.Errorf("[ReadVpnPacketFromBytes] CmdData len(b)[%d]
<"+strconv.Itoa(dataHeaderOverHeadV1), len(b))
    }
    packet.DataSequenceId = binary.LittleEndian.Uint32(b[9:dataHeaderOverHeadV1])
    packet.Data = b[dataHeaderOverHeadV1:]
    return nil
```

File location Talaria-main/client/session.go:948-958

```
switch packet.Cmd {
case vpnpacket.CmdData, vpnpacket.CmdDataReturn:
    if len(packet.Data) == 0 {
        continue
    }
    pkt := pool.Get()
    if err := copyTunPayload(pkt, packet.Data); err != nil {
        pool.Put(pkt)
        return fmt.Errorf("Talaria client: copy tun payload: %w", err)
    }
    if err := downlinkWriter.writeBuffer(pkt, pool); err != nil {
```

IMPACT

A malicious or compromised authenticated VPN server can deliver truncated, concatenated, trailing, or otherwise malformed IP bytes across the client's TUN boundary. Depending on the operating system and downstream components, this can cause inconsistent packet handling, traffic corruption, or localized availability effects. An unauthenticated network attacker cannot directly exercise the path.

RECOMMENDATIONS

- Use one shared IPv4 and IPv6 check before every client TUN write.
- Reject wrong IP versions, bad header or payload lengths, cut-off packets, invalid extension headers, and extra trailing bytes.
- Choose one clear response for bad data: drop it with limited logging, or close the session. Use the same response in every receive path.

COMMENTS

jiawen.zheng@vantagepoint.sg • 09/10/2026

Retest result: Fixed

The updated branch now validates each client downlink buffer before any TUN write (under the case that the application only supports IPv4). Under the implemented IPv4-only policy, malformed, truncated, and non-IPv4 payloads are dropped, only a successfully parsed IPv4 packet reaches the TUN. Download and TUN statistics are recorded only for bytes actually written. The original unchecked payload path could not be reproduced.

Fixed code: client/session.go

```
if ipPacket, err := ippacket.ParseIPPacket(pkt.Bytes()); err != nil ||
!ipPacket.IsIPv4() {
    return 0, nil
}

n, err := writeTunPacketBuffer(w.ctx, w.tun, item.pkt)
if err != nil || n == 0 {
    return err
}
```

Result: the malformed downlink payload is rejected before the client TUN boundary. No regression was observed in valid IPv4 writes.

REFERENCES

MITRE CWE-1286 Improper Validation of Syntactic Correctness

<https://cwe.mitre.org/data/definitions/1286.html>

MITRE CWE-1288 Improper Validation of Consistency within Input

<https://cwe.mitre.org/data/definitions/1288.html>

4.2. Client TLS configuration preserves unsafe entropy, clock, and key-log hooks



OBSERVATIONAL

Observational / Score: 0.0

Closed

CVSS:4.0/AV:L/AC:L/AT:P/PR:H/UI:N/VC:N/VI:N/VA:N/SC:N/SI:N/SA:N

Vulnerability Raised: 22/09/2026

BACKGROUND

Talaria locks TLS to version 1.3, but its client implementation retains optional Rand, Time, and KeyLogWriter hooks from a caller-supplied Go `tls.Config`. The client package implements the Talaria VPN client session, and a thick client is known to use the Talaria protocol. This finding applies if that thick client uses the reviewed Talaria client implementation and supplies one or more of these hooks. If the thick client uses a separate implementation or leaves these fields unset, the identified condition is not present.

DESCRIPTION

Talaria copies the client's TLS configuration and keeps the Rand, Time, and KeyLogWriter fields. These fields are useful for testing, but they can be unsafe in production. If a developer enables one of these fields in a shared production build, every client using that build or setting may be affected. A remote server cannot turn these fields on through the network.

File location Talaria-main/transport/socks5tls/config.go:89-113

```
func normalizeClientTLSConfig(cfg *tls.Config) (*tls.Config, error) {
    // checks omitted
    clone := cfg.Clone()
    if cfg.RootCAs != nil {
        clone.RootCAs = cfg.RootCAs.Clone()
    }
    clone.MinVersion = tls.VersionTLS13
    clone.MaxVersion = tls.VersionTLS13
    clone.SessionTicketsDisabled = true
    return clone, nil
}
```

File location Talaria-main/protocol/udpTLS/protocol.go:902-930

```
func newClientTLSConfig(req NewDialerReq) (*tls.Config, error) {
    // checks omitted
    tlsCfg := req.TLSConfig.Clone()
    tlsCfg.ServerName = serverName
    tlsCfg.RootCAs = req.TLSConfig.RootCAs.Clone()
    tlsCfg.MinVersion = tls.VersionTLS13
    tlsCfg.MaxVersion = tls.VersionTLS13
    tlsCfg.SessionTicketsDisabled = true
    return tlsCfg, nil
}
```

IMPACT

KeyLogWriter can save TLS session secrets. Anyone who gets those secrets and a copy of the network traffic may decrypt that traffic. A weak Rand source may weaken TLS keys. A false Time function may change certificate date checks.

RECOMMENDATIONS

Talaria should reject client configurations containing these fields:

```
if cfg.Rand != nil || cfg.Time != nil || cfg.KeyLogWriter != nil { return nil,
errors.New( "client TLS entropy, clock, and key-log hooks are not supported", ) }
```

Rejecting them produces an immediate configuration error, preventing unsafe test or debugging settings from silently reaching production.

- If these hooks are needed for tests, allow them only in an explicit debug or test build.
- Never store TLS key logs in a place that other users or systems can read.

NOTE

Not remotely settable by an unauthenticated peer, which limits severity but violates the built-in secure-profile boundary.

COMMENTS

jiawen.zheng@vantagepoint.sg • 09/10/2026

Retest result: Fixed

The updated branch now rejects caller-supplied TLS entropy, clock, and key-log hooks before a TalariaTCP or TalariaUDP client configuration can be used. Retesting confirmed that Rand, Time, and KeyLogWriter are rejected for both transports; unsafe hooks are no longer silently preserved.

Fixed code: transport/socks5tls/config.go

```
if cfg.Rand != nil || cfg.Time != nil || cfg.KeyLogWriter != nil {
    return nil, errors.New("client tls Rand, Time, and KeyLogWriter hooks are not
supported")
}
```

Fixed code: protocol/udptls/protocol.go

```
if req.TLSConfig.Rand != nil || req.TLSConfig.Time != nil || req.TLSConfig.KeyLogWriter
!= nil {
    return nil, fmt.Errorf("TalariaUDP tls Rand, Time, and KeyLogWriter hooks are not
supported")
}
```

Result: the unsafe caller-controlled TLS hook path is no longer accepted.

REFERENCES

Go crypto tls Config documentation

<https://pkg.go.dev/crypto/tls#Config>

4.3. Missing minimum RSA certificate strength

 OBSERVATIONAL	Observational / Score: 0.0
	Closed
	CVSS:4.0/AV:N/AC:H/AT:P/PR:N/UI:N/VC:N/VI:N/VA:N/SC:N/SI:N/SA:N
	Vulnerability Raised: 22/09/2026

BACKGROUND

Both Talaria transports use TLS 1.3 certificates to prove server identity. Certificate setup should also enforce the approved minimum key strength.

DESCRIPTION

Talaria checks the RSA key type and that the certificate matches the key, but it does not require a minimum RSA key size. The UDP certificate path has the same gap. Tests showed that both TLS setups accepted a trusted RSA-1024 certificate signed with SHA-256. They rejected the tested RSA-2048 certificate signed with SHA-1. This shows the missing key-size rule, not a broken TLS 1.3 connection.

File location Talaria-main/transport/socks5tls/config.go:164-169

```
switch key := certificate.PrivateKey.(type) {
case *rsa.PrivateKey:
    publicKey, ok := leaf.PublicKey.(*rsa.PublicKey)
    if key == nil || key.N == nil || key.E <= 0 || !ok ||
!key.PublicKey.Equal(publicKey) {
        return nil, errors.New("server tls certificate private key is invalid")
    }
}
```

File location Talaria-main/protocol/udptls/protocol.go:950-967

```
GetCertificate: func(info *tls.ClientHelloInfo) (*tls.Certificate, error) {
    var cert *tls.Certificate
    err := extension.Do("TalariaUDP certificate callback", func() error {
        var callbackErr error
        cert, callbackErr = getCertificate(info)
        return callbackErr
    })
    if err != nil {
        return nil, err
    }
    if cert == nil {
        return nil, fmt.Errorf("TalariaUDP get_certificate returned nil")
    }
    return cert, nil
},
MinVersion:          tls.VersionTLS13,
SessionTicketsDisabled: true,
```

IMPACT

A server that uses a trusted RSA key below 2048 bits has weaker identity protection. If an attacker recovers that private key, the attacker may copy the server.

RECOMMENDATIONS

- Reject RSA keys below 2048 bits when loading a certificate.
- Write down the allowed key types, curves, signatures, and certificate-chain rules for each transport.
- Keep normal certificate trust and host-name checks. Apply the key-strength rule to every certificate path.

COMMENTS

jiawen.zheng@vantagepoint.sg • 09/10/2026

Retest result: Fixed

Both TalariaTCP and TalariaUDP now enforce a minimum 2048-bit RSA key strength for the server leaf certificate and every RSA certificate in the supplied chain. Retesting confirmed that 1024-bit and 2047-bit keys are rejected, while 2048-bit RSA and supported ECDSA chains are accepted.

Fixed code: transport/socks5tls/config.go

```
if key.N.BitLen() < 2048 {
    return nil, errors.New("server tls RSA key must be at least 2048 bits")
}
for _, der := range certificate.Certificate[1:] {
    ca, err := x509.ParseCertificate(der)
    if err != nil {
        return nil, errors.New("server tls CA certificate is invalid")
    }
    if key, ok := ca.PublicKey.(*rsa.PublicKey); ok && key.N.BitLen() < 2048 {
        return nil, errors.New("server tls CA RSA key must be at least 2048 bits")
    }
}
```

The TalariaUDP listener performs the equivalent leaf and CA-chain checks before accepting the certificate.

Result: RSA keys below 2048 bits are rejected consistently by both transport certificate paths.

REFERENCES

MITRE CWE-326 Inadequate Encryption Strength

<https://cwe.mitre.org/data/definitions/326.html>

4.4. TalariaTCP accepts non-canonical SOCKS session credentials



OBSERVATIONAL

Observational / Score: 0.0

Closed

CVSS:4.0/AV:N/AC:L/AT:N/PR:L/UI:N/VC:N/VI:N/VA:N/SC:N/SI:N/SA:N

Vulnerability Raised: 22/09/2026

BACKGROUND

TalariaTCP uses a decimal client ID and an unpadded Base64URL form of a 32-byte token. Each credential should have only one accepted text form.

DESCRIPTION

The server reads the client ID with strconv.ParseUint but does not require one standard decimal form. For example, it accepts leading zeroes. It also reads the token with non-strict Base64URL rules. Tests accepted client ID 007, a Base64URL token with CR or LF, and a token with changed unused bits. Each value still pointed to client ID 7 and the same 32-byte token. The attacker still needs the valid token.

File location Talaria-main/transport/socks5tls/server.go:1152-1168

```
func decodeCredentialsBytes(user, password []byte, token *
[coretransport.DataTokenSize]byte) (uint64, bool) {
    if token == nil {
        return 0, false
    }
    clear(token[:])
    clientID, err := strconv.ParseUint(string(user), 10, 64)
    if err != nil || clientID == 0 {
        return 0, false
    }
    raw := make([]byte, base64.RawURLEncoding.DecodedLen(len(password)))
    defer clear(raw)
    n, err := base64.RawURLEncoding.Decode(raw, password)
    if err != nil || n != len(token) {
        return 0, false
    }
    copy(token[:], raw[:n])
    return clientID, true
}
```

IMPACT

Different text can represent the same login value. This may bypass text-based filters, create mixed audit records, and make an investigation harder. It does not reveal the token or let a person log in without a valid token.

RECOMMENDATIONS

- Allow only ASCII digits in the client ID, limit its length, reject leading zeroes, and compare it with strconv.FormatUint.
- Require a 43-byte unpadded Base64URL token, reject CR and LF, use

base64.RawURLEncoding.Strict(), and compare the result with the original text.

- Return the same login error for every bad credential format or wrong credential.

COMMENTS

jiawen.zheng@vantagepoint.sg • 09/10/2026

Retest result: Fixed

The TaliaTCP SOCKS credential decoder now requires the exact unpadded Base64URL length, canonical decimal client-ID text, and strict Base64URL decoding. Retesting confirmed that canonical credentials remain accepted while leading-zero IDs, zero or overflowing IDs, CR/LF variants, non-zero unused bits, padding, and shortened tokens are rejected.

Fixed code: transport/socks5tls/server.go

```
if len(password) != base64.RawURLEncoding.EncodedLen(len(token)) {
    return 0, false
}
clientID, err := strconv.ParseUint(string(user), 10, 64)
if err != nil || clientID == 0 || string(user) != strconv.FormatUint(clientID, 10) {
    return 0, false
}
n, err := base64.RawURLEncoding.Strict().Decode(raw, password)
if err != nil || n != len(token) {
    return 0, false
}
```

Result: non-canonical SOCKS session credential encodings are no longer accepted.

REFERENCES

RFC 4648 Base-N Encodings and Canonical Encoding

<https://www.rfc-editor.org/rfc/rfc4648.html#section-3.5>

MITRE CWE-180 Validate Before Canonicalize

<https://cwe.mitre.org/data/definitions/180.html>

5. Appendix

5.1. Disclaimer

The material contained in this document is confidential and only for use by the company receiving this information from Vantage Point. The material will be held in the strictest confidence by the recipients and will not be used, in whole or in part, for any purpose other than the purpose for which it is provided without prior written consent by Vantage Point. The recipient assumes responsibility for further distribution of this document. In no event shall Vantage Point be liable to anyone for direct, special, incidental, collateral or consequential damages arising out of the use of this material, to the maximum extent permitted under law.

The security testing team made every effort to cover the systems in the test scope as effectively and completely as possible given the time budget available. There is however no guarantee that all existing vulnerabilities have been discovered. Furthermore, the security assessment applies to a snapshot of the current state at the examination time.

5.2. Risk Rating

COMMON VULNERABILITY SCORING SYSTEM

CVSS is an open standard designed to allow organizations to understand the criticality of security vulnerabilities and assess the priority given to security patching. It is platform and technology independent; in practice, CVSS scores can be used to indicate the severity of security vulnerabilities affecting products like operating systems, security products such as firewalls and databases. Vantage Point uses CVSS scoring to be more transparent and allow organizations to compare and process vulnerabilities found by the security consultants of Vantage Point with vulnerabilities from other sources. A unified vulnerability scoring system within an organization allows better understanding of vulnerability risks and consequently provides a better basis for decisions to improve information security.

VULNERABILITY RISK SCORE

All risk calculation and vulnerabilities found by Vantage Point will receive an individual risk rating performed with the official CVSS v4.0 risk calculator.

CRITICAL

CVSS score between 9.0 to 10.0

A CRITICAL finding requires immediate attention and should be given the highest priority by the business as it will impact business interest critically.

HIGH

CVSS score between 7.0 to 8.9

A HIGH finding requires immediate attention and should be given higher priority by the business.

MEDIUM

CVSS score between 4.0 to 6.9

A MEDIUM finding has the potential to present a serious risk to the business.

LOW

CVSS score between 0.1 to 3.9

A LOW finding contradicts security best practices and have minimal impact on the business.

OBSERVATIONAL

CVSS Score 0.0

An OBSERVATIONAL finding relates primarily to non-compliance issues or is considered an additional security feature that would increase the security stance of the environment. It does not have a CVSS Score.