Workgroup: Network Working Group
Internet-Draft: draft-zubov-snif-00
Published: 31 January 2022
Intended Status: Experimental
Expires: 4 August 2022
Authors: J. Zubov
VESvault Corp

Deploying Publicly Trusted TLS Servers on IoT Devices Using SNI-based End-to-End TLS Forwarding (SNIF)

Abstract

This document proposes a solution, referred as SNIF, that provides the means for any Internet connected device to:

* allocate a globally unique anonymous hostname
* obtain and maintain a publicly trusted X.509 certificate [RFC5280] issued for the allocated hostname
* accept incoming TLS connections on specific TCP ports of the allocated hostname from any TLS clients that are capable of sending Server Name Indication [RFC6066]

The private key associated with the X.509 certificate is securely stored on the TLS terminating device, and is never exposed to any other party at any step of the process.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-zubov-snif-00 .

Information can be found at https://snif.host .

Source for this draft and an issue tracker can be found at https://github.com/vesvault/snif-i-d .

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

A typical Internet-of-Things (IoT) device connects to the Internet using a dynamic IP address, and is usually unable to accept incoming connections to TCP ports. A dedicated trusted relay is needed to facilitate the communications between the IoT device and its intended users. While all communications are recommended to be TLS encrypted, the trusted relay will terminate each TLS connection and therefore have access to unencrypted traffic between IoT devices and user clients, which may pose undesirable security risk.

Designing a dedicated relay that works in end-to-end encrypted mode, where the TLS tunnel is established between the IoT device and the client, and is passed by the relay in an encrypted form, raises additional challenges. Clients expect to be able to verify the authenticity of the TLS certificate presented by the IoT device they are connecting to. Public certificate authorities requite to validate the ownership of the hostname the certificate is being requested for, using certain challenge mechanisms. Therefore, the IoT device needs to allocate a unique hostname, and to be able to complete the CA challenge in order to acquire a trusted certificate.

Alternatively, the client may decide to use a different certificate trust scheme, not based on publicly trusted root CAs. In this case, the client is limited to specifically built software with custom trust rules, or the system trust root on the client device needs to be customized.

This document proposes a solution, referred as SNIF, that allows any common TLS client with standard root CAs, such as a web browser, to establish a trusted end-to-end TLS connection with an IoT device using the unique hostname permanently allocated to the device, via a dedicated relay.

While this document focuses on IoT devices, SNIF is applicable to any physical or virtual device or software that can benefit from accepting trusted TLS connections to an anonymous hostname.

1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Overview

SNIF CA Proxy is a combination of web-based services and background processes that run on a publicly accessible server, normally on the same physical server as SNIF Relay. SNIF CA Proxy allocates hostnames for SNIF Connectors and facilitates issuing and renewing X.509 certificates without having access to the Connectors' private keys. The functions of SNIF CA Proxy are described in Section 3.

SNIF Relay is a process that runs on a publicly accessible server, normally on the same physical server as SNIF CA Proxy. SNIF Relay facilitates end-to-end TLS connections between SNIF Clients with SNIF Connectors. The functions of SNIF Relay are described in Section 4.

SNIF Connector is a software process that runs on an IoT device, or on other type of device that intends to provide TLS-based services that can be accessed by general purpose TLS clients using SNIF Relay. SNIF Connector can be implemented as a standalone process that communicates with the TLS server processes over local filesystem and sockets, or as an integral part of a TLS server process.

SNIF Client is any common TLS-compatible client with SNI capability, such as a web browser or an email client, that connects to a SNIF hostname provided by a specific SNIF Connector. SNIF Client does not need any awareness of SNIF, or of any protocols described in this document.

Certificate Authority (CA) is a service that issues public trusted TLS Certificates to specific hostnames when requested by the hostname owner, upon validating the ownership of the hostname. CA does not need any awareness of SNIF, except for a working relationship with the SNIF CA Proxy that requests certificates using protocols supported by the CA.

SNIF Peripheral Process is any kind of additional service that extends or supplements functions of SNIF, in a way not defined within the scope of this document.

3. SNIF CA Proxy Protocol

SNIF CA Proxy Protocol is designed for securely acquiring and maintaining a publicly trusted TLS/SSL X.509 certificate issued by a Certificate Authority to a uniquely allocated hostname, by an agent that has no direct control over that hostname, or over a server the hostname is pointing to.

3.1. Protocol Summary

SNIF CA Proxy accepts requests from SNIF Connectors via HTTP / HTTPS.

SNIF CA Proxy interacts with the CA using protocols supported by the CA, such as ACME [RFC8555], not covered by this document.

Each SNIF Connector is configured with a specific initiation URL ({initUrl}), which is specific to the SNIF CA Proxy server the Connector intends to work with. Depending on the CA Proxy rules, {initUrl} might be unique for each Connector, or common for multiple Connectors.

3.2. Protocol Flow

Upon the initial start or after a hard reset, the Connector MUST generate a Private Key, which needs to be securely permanently stored by the Connector. Any key algorithm acceptable by the CA can be used, generally RSA-4096 is recommended.

The Connector MUST send a CN Allocation Request using the {initUrl}.

Having the {cn}, the Connector MUST generate a CSR [RFC2986] using the Private Key, the subject containing the {cn}. The CSR subject may or may not have other fields besides {cn}, according to the specific requirements of the CA.

The Connector MUST issue a CSR Submission Request to send the CSR to the CA Proxy.

Once the CSR is submitted, the Connector MUST permanently store the {cn} by some means - to minimize the storage compartments it might be practical to generate and store a dummy self-signed certificate with the {cn} in the subject until it gets replaced with a trusted certificate issued by the CA.

A this point, the Connector will normally know the SNIF hostname it will be using with the SNIF Relay - it matches the {cn} in case of a single host CN, or is a one sub-level down from a wildcard {cn}, the name being derived by the Connector in a way that is not deterministically derivable from the {cn} and the public key, e.g. a hash of the Private Key. The Connector SHOULD communicate the hostname by some means to the SNIF Clients that will be accessing the Connector. The means of such communication is not covered by this document.

The Connector can now send a Certificate Download Request, and SHOULD verify the returned Certificate. If the Certificate is valid - the Connector MUST permanently store it.

If the Certificate Download Request fails - the Connector should repeat the request after certain delay. In case if the response was 401 and the {authUrl} is returned in a header, and the Connector has the means of communicating with the device user - the Connector also

SHOULD alert the user and bring {authUrl} to their attention by some means, so the user can complete the required authorization steps. If the Connector has no means of alerting the user, which is often the case with IoT devices - the user should be provided with some external means of authorizing with the CA Proxy, not covered by this domcument.

Once the Certificate is stored, the Connector is capable of terminating SNIF connections, and may proceed launching a SNIF Control Connection (Section 4.2).

The Connector SHOULD watch for the expiration of the stored Certificate. If the Certificate is about to expire in 7 days or less, or has already expired - the Connector SHOULD send a Certificate Download Requests, and repeat with appropriate delays until the renewed Certificate is successfully downloaded and verified.

At any stage of the flow, if the Connector receives unexpected volume of rejections or inconsistent responses from the CA Proxy, the Connector MAY decide to hard reset the storage and start the flow over from the beginning. In such case, the Connector will have to re-send its new SNIF hostname to any concerned SNIF Clients, the means of such communication is not covered by this document.

3.3. CN Allocation Request

Connection from: SNIF Connector

Connection to: SNIF CA Proxy

Protocol: https or http

GET {initUrl}

Response 200: CN is successfully allocated. The response headers MUST include X-SNIF-CN: with the value of the allocated {cn}, either a wildcard starting with "*.", or a single hostname, depending on the CA Proxy rules. The response content type SHOULD be "text/plain", the response body SHOULD include the copy of the allocated {cn}, optionally padded with newlines or spaces on the right.

Any other response: Error, try again later.

3.4. CSR Submission Request

Connection from: SNIF Connector

Connection to: SNIF CA Proxy

Protocol:
http

PUT http://{cn_host}/snif-cert/{cn_host}.csr
Content-Type: application/pkcs10

{cn_host} is a hostname derived from the {cn} - it is identical to {cn} in case of a single-host CN, or is the {cn} with truncated initial "*." in case of a wildcard CN.

The request body MUST contain a PEM encoded PKCS#10 CSR [RFC5967], the newlines are either <CR><LF> or <LF>, the length of the body SHOULD NOT exceed 16384 bytes.

Note that a CSR for the specific allocated CN can be submitted to the CA Proxy once in a lifetime. In case of an incorrect submission the Connector should hard reset the storage and restart the flow from the beginning, including allocating a new CN.

Response 201: the CSR is successfully submitted. The response headers MAY include X-SNIF-AuthUrl: with the value of an {authUrl}, that SHOULD, if possible, be communicated to the user to authorize the certificate issuance.

Response 403: the CSR for this CN has already been submitted, or is denied by the CA Proxy rules. If the Connector receives 403, is SHOULD hard reset the storage and restart the CA Proxy flow from the beginning.

Response 404: the CN was not allocated.

Any other response: Error, try again later.

3.5. Certificate Download Request

Connection from: SNIF Connector

Connection to: SNIF CA Proxy

Protocol: http

GET http://{cn_host}/snif-cert/{cn_host}.crt

{cn_host} is a hostname derived from the {cn} - it is identical to {cn} in case of a single-host CN, or is the {cn} with truncated initial "*." in case of a wildcard CN.

The CA Proxy SHOULD check for a cached previously generated Certificate chain for the {cn}. If the cached Certificate chain is found and if it expires in more that 10 days in the future - the cached Certificate chain SHOULD be returned with status 200. Otherwise, if the {cn} has a valid CSR and a proper authorization to issue a certificate - the CA Proxy SHOULD return status 503 and SHOULD launch a background process that communicates with the CA to issue or renew the certificate, and caches the issued Certificate chain for subsequent Certificate Download Requests.

Response 200: the Certificate chain is returned. The Content-Type of such response SHOULD be "application/x-x509-ca-cert". The response body MUST be a PEM encoded X.509 certificate chain, the issued certificate being the first member, the newlines are either <CR><LF> or <LF>, the length of the body SHOULD NOT exceed 65535 bytes.

Response 503: the Certificate is being issued, try later.

Response 401: Certificate issuance authorization is required. The response headers MAY include X-SNIF-AuthUrl: with the value of an {authUrl}, that SHOULD, if possible, be communicated to the user to authorize the certificate issuance. If the Connector cannot communicate with the user, the CA Proxy should include external means of the authorization, not covered by this document.

Response 404: the CN was not allocated, or the CSR was not submitted.

Any other response: Error, try again later.

4. SNIF Relay Protocol Suite

Except for SNIF Client Connection, all protocols mentioned below involve sending and receiving asynchronous SNIF Messages over a specific type of stream connection.

SNIF Control Connection Protocol defines communications between SNIF Relay and SNIF Connector that runs on an IoT device, or other type of device that provides TLS-based services through SNIF.

SNIF Service Connection Protocol defines secondary communications between SNIF Relay and SNIF Connector that include end-to-end TLS traffic forwarded by the Relay.

SNIF Client Connection Protocol defines TLS communications between SNIF Relay and a Client, where the Relay acts as a transparent end-to-end forwarder.

SNIF IPC FIFO Protocol defines communications between nodes of a SNIF Relay cluster, and/or between SNIF Relay and SNIF Peripheral Processes.

4.1. SNIF Messages

A SNIF Message consists of a 1 or more ASCII characters excluding special characters, terminated by <CR><LF>.

The total length of a SNIF Message, including the terminal <CR><LF>, SHOULD NOT exceed 4096 bytes.

8-bit characters are discouraged. If 8-bit characters are used, they should comply to UTF-8 [RFC3629].

The receiving party SHOULD silently ignore any invalid or malformed SNIF message.

4.2. SNIF Control Connection Protocol

Protocol name: snif

Default port: TCP 7123

Connection from: SNIF Connector

Connection to: SNIF Relay

To be able to open a SNIF Control Connection, the SNIF Connector MUST have a valid trusted TLS/SSL certificate, the CN hostname DNS pointing to the SNIF Relay or a wildcard CN having a sub-host DNS pointing to the SNIF Relay, and a Private Key that matches the Certificate. Normally, the SNIF Connector will generate the Private Key and use SNIF CA Proxy Protocol (Section 3) to obtain and maintain the Certificate, although other means can be used.

To initiate the Control Connection, the SNIF Connector opens a TCP connection to the hostname matching the Certificate's CN, that points to the Relay.

Upon accepting the incoming TCP connection, the SNIF Relay MUST initiate a reversed TLS session as a client peer.

The SNIF Connector MUST initiate the TLS as a server peer, using the Certificate and the Private Key.

Upon successful TLS negotiation, the SNIF Relay MUST validate the SNIF Connector's certificate. If the certificate is not trusted, the SNIF Relay MUST shut down the TLS session and the TCP socket immediately.

If the certificate is accepted, both SNIF Relay and SNIF Connector are ready to accept SNIF Messages from each other over the TLS connection, as following.

SNIF LISTEN {hostname}

Sent by: SNIF Connector

The SNIF LISTEN message informs the Relay that the Connector is ready to accept incoming TLS connections to {hostname} through the Relay.

{hostname} MUST specify a single host (no wildcards), and MUST match the CN of the Connector's TLS certificate - either match a wildcard CN, or exactly match a single host CN.

The SNIF LISTEN message SHOULD be send only once per the Control Connection. The Relay SHOULD ignore any invalid or subsequent SNIF LISTEN messages.

SNIF CONNECT {conn_id} {dst_host}:{dst_port} {fwd_host}:{fwd_port} {c

Sent by: SNIF Relay

The SNIF CONNECT message informs the Connector of an incoming TLS connection from a Client to the Connector's {dst_host}, TCP port {dst_port}.

{conn_id} is a unique alphanumeric connection identifier assigned by the Relay, {cln_addr}:{cln_port} are the Client's remote IPv4/IPv6 address and TCP port, {cln_addr} is supplied in "[" brackets "]".

The Relay sends the SNIF CONNECT message to Connectors with {dst_host} matching the {hostname} the Connector is listening to. The Connector doesn't need to verify {dst_host}.

If the Connector decides to accept the connection - it MUST launch a SNIF Service Connection to {fwd_host}:{fwd_port}. It also SHOULD send any SNIF message back to the Relay over the Control Connection to update the keep-alive timer, a copy of the SNIF ACCEPT message that is sent over the Service Connection can be used.

In case of a rejection - the Connector SHOULD send SNIF CLOSE with matching {conn_id}.

SNIF CLOSE {conn_id}

Sent by: SNIF Connector

The SNIF CLOSE message instructs the Relay to terminate the Client connection with matching {conn_id}.

For SNIF CLOSE received from a Connector, the Relay MUST validate that the connection was targeted at the Connector's {hostname}, otherwise ignore the message.

SNIF ABUSE {conn_id} {abuse_score}

Sent by: SNIF Connector

The SNIF ABUSE message instructs the Relay to increase the DoS protection abuse counter for the Client that initiated the connection {conn_id} by {abuse score}.

{abuse score} SHOULD be an integer from 1 to 255, 1 is the score for a normal non-abusive connection.

For SNIF ABUSE received from a Connector, the Relay MUST validate that the connection was targeted at the Connector's {hostname}, otherwise ignore the message.

SNIF MSG {hostname} {content}

Sent by: SNIF Connector or SNIF Relay

The SNIF MSG message is relayed between the Connector and the SNIF Peripheral Processes attached to the Relay.

{content} SHOULD NOT contain whitespaces or special characters. Its semantics is specific to the targeted Peripheral Process, and is not covered by this document.

For SNIF MSG received by the Relay from a Connector, the Relay MUST verify that the {hostname} matches the one associated with the Connector, forward the message to all IPC FIFOs if matched, ignore otherwise.

For SNIF MSG received by the Relay from an IPC FIFO, the Relay SHOULD forward the message to the Connector(s) with the matching {hostname}, ignore the message if none are found.

Note that in certain uncommon circumstances a SNIF MSG send by a Connector might come back to the Connector through a different Control Connection. The Connector SHOULD be aware of this fact to avoid a potential message storm.

NOOP

Sent by: SNIF Connector or SNIF Relay

The NOOP message is not associated with any explicit action, except that the Relay receiving NOOP from the connector SHOULD promply send

NOOP or any other message back to the Connector. Therefore, the Connector may use NOOP as a keep-alive ping.

4.3. SNIF Service Connection Protocol

Protocol name: snif-srv

Default port: TCP 7120 (unofficial)

Connection from: SNIF Connector

Connection to: SNIF Relay

The SNIF Connector opens a TCP connection to the {fwd_host}:{fwd_port} in response to a SNIF CONNECT message received from the Relay over the Control Connection.

The Connector MUST immediately send a SNIF ACCEPT message over the Service Connection as a plain TCP:

SNIF ACCEPT {conn_id}

The {conn_id} is the one that was received in the SNIF CONNECT message over the Control Connection.

Upon sending the SNIF ACCEPT message, the Connector MUST immediately assign further control and bi-directional traffic of the SNIF Service Connection to the matching TLS server process.

If the Relay decides to reject the connection, either because of invalid message or {conn_id}, or because of reaching the abuse threshold - the Relay SHOULD terminate the TCP connection immediately.

Otherwise, the Relay SHOULD link the Service Connection to the matched Client Connection, forward to the Service Connection all buffered TLS data previously received from the Client, and start bi-directional forwarding between the Client Connection and the Service Connection.

When either Client or Service Connection is shut down, or an inactivity timeout is reached, the Relay SHOULD shut down both the Client Connection and the Service Connection.

Once the Relay has linked the Client Connection matching the {conn_id} to the Service Connection, any further SNIF ACCEPT messages with the same {conn_id} on other Service Connections MUST be rejected.

4.4. SNIF Client Connection Protocol

Protocol name:
snif-cln

Default port: N/A

Connection from: Any TLS enabled software, such as a web browser or an email client

Connection to: SNIF Relay

From the Client's perspective, a SNIF Client Connection functions as a direct TLS connection to the IoT Device.

The ports the Relay is listening to, can be any well-known ports for services with persistent TLS, such as https or imaps, or can be any custom ports agreed among the Relay, the Connectors and the Clients.

The Relay accepts an incoming TCP connection, receives and buffers the incoming initial data from the client, and attempts to interpret the received data as a TLS handshake.

If the received data is not recognized as a TLS handshake, does not contain an SNI record in a supported format, or the SNI hostname does not meet rules defined for the Relay - the Relay SHOULD immediately reject the TLS session with an appropriate error status, and shut down the Client Connection.

If the SNI hostname is found acceptable - the Relay allocates a unique {conn_id}, checks if there are current Control Connections that match the SNI hostname, and sends a SNIF CONNECT message over those connections.

If there are no active applicable Control Connections, or if the Relay doesn't receive a response from a SNIF Connector within a specified timeframe - the Relay SHOULD forward the same SNIF CONNECT message over IPC FIFOs (if any are open) to alert cluster peer Relays and Peripheral processes of the incoming Client Connection.

A Service Connection with a matching SNIF ACCEPT establishes an end-to-end TLS circuit with the Client Connection. Once established, the Relay bi-directionally forwards all traffic between the Client and the Service Connection until either of the connections is closed or is timed out due to inactivity.

Upon receiving a matching SNIF CLOSE - the Relay MUST terminate the Client Connection. If a Service Connection has already been linked it MUST be terminated too, otherwise the Relay SHOULD attempt to gracefully reject TLS on the Client Connection with an appropriate status prior to shutting down TCP.

4.5. SNIF IPC FIFO Protocol

Protocol name: snif-fifo

Default port: N/A

Connection from: SNIF Relay or SNIF Peripheral Service

Connection to: SNIF Relay or SNIF Peripheral Service

SNIF IPC FIFO is a permanent trusted connection between the SNIF Relay and a SNIF Peripheral Process, or between a pair of nodes in a SNIF Relay cluster. An IPC FIFO is usually unidirectional, but a bidirectional connection can serve as a pair of FIFOs. An IPC FIFO can be implemented as a Unix FIFO pipe, a TCP socket, an SSH tunnel or by other means. The mechanism of establishing and maintaining IPC FIFOs is implementation specific and is not covered by this document.

The following SNIF Messages are defined over an IPC FIFO from the perspective of a SNIF Relay:

SNIF CONNECT {conn_id} {dst_host}:{dst_port} {fwd_host}:{fwd_port} {c

Direction: Send or Receive

(see SNIF Control Connection, Section 4.2).

The SNIF CONNECT message is sent by a Relay over an IPC FIFO in case if the Relay failed to reach the respective Connector through Control Connections. When sent by a Relay, SNIF CONNECT must be followed up by one of SNIF CLEAR or SNIF CLOSE to inform the Peripheral Processes of the further outcome.

When a SNIF CONNECT message is received by a Relay, the Relay SHOULD forward it to any matching open Control Connections, or ignore it otherwise.

SNIF CLEAR {conn_id}

Direction: Send

The SNIF CLEAR message should be sent by a Relay only as a followup to SNIF CONNECT with a matching {conn_id}, in case if the Client Connection that triggered SNIF CONNECT was accepted by a Service Connection.

The purpose of SNIF CLEAR is to advice Peripheral Processes to cease further attempts of reaching the Connector by external means, not specified within this document.

SNIF CLOSE {conn_id}

Direction: Send or Receive

(see SNIF Control Connection, Section 4.2).

The SNIF CLOSE message should be sent by a Relay only as a followup to SNIF CONNECT with a matching {conn_id}, in case if the Client Connection that triggered SNIF CONNECT was closed without being accepted.

When the SNIF CLOSE is received by a Relay, the Relay SHOULD immediately close the matching Client and/or Service Connection if any found, ignore the message otherwise.

SNIF ABUSE {conn_id} {abuse_score}

Direction: Receive

(see SNIF Control Connection, Section 4.2).

SNIF MSG {hostname} {content}

Direction: Send or Receive

(see SNIF Control Connection, Section 4.2).

SNIF CTL {ctl_fd} {hostname} {remote_addr}:{remote_port}
SNIF CTL {ctl_fd}

Direction: Send

The SNIF CTL message is sent by a Relay to inform Peripheral Processes about Control Connections. The first version is sent for each opening Control Connection, and is followed up by the second version with the matching {ctl_fd} when the Control Connection is closed. {ctl_fd} is a numeric descriptor which is unique for open connections, but can be reused after a connection is closed.

4.6. Abuse Management

SNIF Relay SHOULD implement basic protection from denial of service. A separate abuse count SHOULD be assigned to each remote address, incremented by 1 on every incoming connection from the address, incremented by a specified score on every received SNIF ABUSE message, and periodically decremented or reset at regular time intervals.

If the abuse counter for a certain remote address reaches a specific threshold, the Relay SHOULD drop any further TCP connections from that address until the abuse counter goes below the threshold. The Relay MAY allow some grace above the threshold to incoming SNIF Service Connections, to minimize stalled Client Connections.

SNIF Connector MAY implement basic protection from denial of service by limiting the number of accepted connections per period of time and/or the total number of open connections, and reject connections over the limit.

5. Security Considerations

All information communicated to/from SNIF CA Proxy over plain unencrypted HTTP is safe to be exposed to third parties or to intruders without compromising any private information.

SNIF Control Connection Protocol communicates all sensitive information over a TLS connection with a trusted certificate.

SNIF Service Connection Protocol communicates a randomly generated {conn_id} over an unsecure TCP connection. Except if used over a trusted SNIF IPC FIFO, the {conn_id} can be used only once to accept the Client's TLS connection, which in turn can only be successfully negotiated by the targeted SNIF Connector. All further communications are comprised of end-to-end encrypted TLS traffic. The security of the TLS encrypted content between the Client and the Connector is specific to the protocols involved. The underlying protocol SHOULD require proper authentication specific to the protocol before communicating any sensitive information. Negotiation of the credentials for such authentification is not covered by this document.

SNIF Client Connection is a TLS session with a trusted certificate. The security of the TLS encrypted content between the Client and the Connector is specific to the protocols involved.

SNIF IPC FIFO connections should only be established between mutually trusted parties, and need to be secured by external means specific to the implementation, such as filesystem permissions, TLS or SSH tunnels etc. The security of such external means cannot be assessed within the scope of this document.

A compromised SNIF CA Proxy can potentially issue certificates to any hostnames allocated by the Relay, including a catch-all wildcard, using an alternative private key, and thus allow a man-in-the-middle attack on any SNIF Connectors associated with the Relay. This vulnerability can be mitigated by constant monitoring of public TLS Transparency logs, such as [RFC6962]. At least one independent party SHOULD continuosly monitor TLS Transparency logs for each deployed SNIF CA Proxy and Relay. Once any duplicate or overlapping certificates are detected - the corresponding SNIF Relay MUST be permanently deemed compromised.

6. IANA Considerations

Protocols "snif", "snif-srv", "snif-cln" and "snif-fifo" are registered with IANA.

TCP port 7123 is registered with IANA for protocol "snif".

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2986] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <https://www.rfc-editor.org/info/rfc2986>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

7.2. Informative References

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC5967] Turner, S., "The application/pkcs10 Media Type", RFC 5967, DOI 10.17487/RFC5967, August 2010, <https://www.rfc-editor.org/info/rfc5967>.

[RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", RFC 6962, DOI 10.17487/RFC6962, June 2013, <https://www.rfc-editor.org/info/rfc6962>.

[RFC8555] Barnes, R., Hoffman-Andrews, J., McCarney, D., and J. Kasten, "Automatic Certificate Management Environment (ACME)", RFC 8555, DOI 10.17487/RFC8555, March 2019, <https://www.rfc-editor.org/info/rfc8555>.

Author's Address

Jim Zubov
VESvault Corp

Email: jz@vesvault.com
URI: https://snif.host