

Aspirational
Internet-Draft
Intended status: Informational
Expires: 24 September 2021

B. Campbell
Ping Identity
23 March 2021

Client-Cert HTTP Header: Conveying Client Certificate Information from
TLS Terminating Reverse Proxies to Origin Server Applications
draft-bdc-something-something-certificate-05

Abstract

This document defines the HTTP header field "Client-Cert" that allows a TLS terminating reverse proxy to convey the client certificate of a mutually-authenticated TLS connection to the origin server in a common and predictable manner.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 24 September 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Requirements Notation and Conventions	3
1.2. Terminology	3
2. HTTP Header Field and Processing Rules	4
2.1. Encoding	4
2.2. Client-Cert HTTP Header Field	4
2.3. Processing Rules	5
3. Security Considerations	6
4. IANA Considerations	7
5. Normative References	7
6. Informative References	7
Appendix A. Example	9
Appendix B. Considerations Considered	10
B.1. Header Injection	10
B.2. The Forwarded HTTP Extension	10
B.3. The Whole Certificate and Only the Whole Certificate . .	11
Appendix C. Acknowledgements	12
Appendix D. Document History	12
Author's Address	13

1. Introduction

A fairly common deployment pattern for HTTPS applications is to have the origin HTTP application servers sit behind a reverse proxy that terminates TLS connections from clients. The proxy is accessible to the internet and dispatches client requests to the appropriate origin server within a private or protected network. The origin servers are not directly accessible by clients and are only reachable through the reverse proxy. The backend details of this type of deployment are typically opaque to clients who make requests to the proxy server and see responses as though they originated from the proxy server itself. Although HTTPS is also usually employed between the proxy and the origin server, the TLS connection that the client establishes for HTTPS is only between itself and the reverse proxy server.

The deployment pattern is found in a number of varieties such as n-tier architectures, content delivery networks, application load balancing services, and ingress controllers.

Although not exceedingly prevalent, TLS client certificate authentication is sometimes employed and in such cases the origin server often requires information about the client certificate for its application logic. Such logic might include access control decisions, audit logging, and binding issued tokens or cookies to a certificate, and the respective validation of such bindings. The specific details from the certificate needed also vary with the

application requirements. In order for these types of application deployments to work in practice, the reverse proxy needs to convey information about the client certificate to the origin application server. A common way this information is conveyed in practice today is by using non-standard headers to carry the certificate (in some encoding) or individual parts thereof in the HTTP request that is dispatched to the origin server. This solution works but interoperability between independently developed components can be cumbersome or even impossible depending on the implementation choices respectively made (like what header names are used or are configurable, which parts of the certificate are exposed, or how the certificate is encoded). A well-known predictable approach to this commonly occurring functionality could improve and simplify interoperability between independent implementations.

This document aspires to standardize an HTTP header field named "Client-Cert" that a TLS terminating reverse proxy (TTRP) adds to requests that it sends to the backend origin servers. The header value contains the client certificate from the mutually-authenticated TLS connection between the originating client and the TTRP. This enables the backend origin server to utilize the client certificate information in its application logic. While there may be additional proxies or hops between the TTRP and the origin server (potentially even with mutually-authenticated TLS connections between them), the scope of the "Client-Cert" header is intentionally limited to exposing to the origin server the certificate that was presented by the originating client in its connection to the TTRP.

1.1. Requirements Notation and 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.

1.2. Terminology

Phrases like TLS client certificate authentication or mutually-authenticated TLS are used throughout this document to refer to the process whereby, in addition to the normal TLS server authentication with a certificate, a client presents its X.509 certificate [RFC5280] and proves possession of the corresponding private key to a server when negotiating a TLS connection or the resumption of such a connection. In contemporary versions of TLS [RFC8446] [RFC5246] this requires that the client send the Certificate and CertificateVerify messages during the handshake and for the server to verify the CertificateVerify and Finished messages.

[[HTTP2 forbids TLS renegotiation and post-handshake authentication but it's possible with HTTP1.1 and maybe needs to be discussed explicitly here or somewhere in this document? Naively I'd say that the "Client-Cert" header will be sent with the data of the most recent client cert anytime after renegotiation or post-handshake auth. And only for requests that are fully covered by the cert but that in practice making the determination of where exactly in the application data the cert messages arrived is hard to impossible so it'll be a best effort kind of thing.]]

2. HTTP Header Field and Processing Rules

2.1. Encoding

The field-values of the HTTP header defined herein utilize the following encoded form.

A certificate is represented in text as an "EncodedCertificate", which is the base64-encoded (Section 4 of [RFC4648]) DER [ITU.X690] PKIX certificate. The encoded value MUST NOT include any line breaks, whitespace, or other additional characters. ABNF [RFC5234] syntax for "EncodedCertificate" is shown in the figure below.

EncodedCertificate = 1*(DIGIT / ALPHA / "+" / "/") 0*2"="

DIGIT = <Defined in Section B.1 of [RFC5234]> ; A-Z / a-z

ALPHA = <Defined in Section B.1 of [RFC5234]> ; 0-9

2.2. Client-Cert HTTP Header Field

In the context of a TLS terminating reverse proxy (TTRP) deployment, the TTRP makes the TLS client certificate available to the backend application with the following header field.

Client-Cert The end-entity client certificate as an "EncodedCertificate" value.

The "Client-Cert" header field defined herein is only for use in HTTP requests and MUST NOT be used in HTTP responses. It is a single HTTP header field-value as defined in Section 3.2 of [RFC7230], which MUST NOT have a list of values or occur multiple times in a request.

2.3. Processing Rules

This section outlines the applicable processing rules for a TLS terminating reverse proxy (TTRP) that has negotiated a mutually-authenticated TLS connection to convey the client certificate from that connection to the backend origin servers. Use of the technique is to be a configuration or deployment option and the processing rules described herein are for servers operating with that option enabled.

A TTRP negotiates the use of a mutually-authenticated TLS connection with the client, such as is described in [RFC8446] or [RFC5246], and validates the client certificate per its policy and trusted certificate authorities. Each HTTP request on the underlying TLS connection are dispatched to the origin server with the following modifications:

1. The client certificate is be placed in the "Client-Cert" header field of the dispatched request as defined in Section 2.2.
2. Any occurrence of the "Client-Cert" header in the original incoming request MUST be removed or overwritten before forwarding the request. An incoming request that has a "Client-Cert" header MAY be rejected with an HTTP 400 response.

Requests made over a TLS connection where the use of client certificate authentication was not negotiated MUST be sanitized by removing any and all occurrences "Client-Cert" header field prior to dispatching the request to the backend server.

Backend origin servers may then use the "Client-Cert" header of the request to determine if the connection from the client to the TTRP was mutually-authenticated and, if so, the certificate thereby presented by the client.

Forward proxies and other intermediaries MUST NOT add the "Client-Cert" header to requests, or modify an existing "Client-Cert" header. Similarly, clients MUST NOT employ the "Client-Cert" header in requests.

A server that receives a request with a "Client-Cert" header value that it considers to be too large can respond with an HTTP 431 status code per Section 5 of [RFC6585].

3. Security Considerations

The header described herein enable a TTRP and backend or origin server to function together as though, from the client's perspective, they are a single logical server side deployment of HTTPS over a mutually-authenticated TLS connection. Use of the "Client-Cert" header outside that intended use case, however, may undermine the protections afforded by TLS client certificate authentication. Therefore steps MUST be taken to prevent unintended use, both in sending the header and in relying on its value.

Producing and consuming the "Client-Cert" header SHOULD be a configurable option, respectively, in a TTRP and backend server (or individual application in that server). The default configuration for both should be to not use the "Client-Cert" header thus requiring an "opt-in" to the functionality.

In order to prevent header injection, backend servers MUST only accept the "Client-Cert" header from a trusted TTRP (or other proxy in a trusted path from the TTRP). A TTRP MUST sanitize the incoming request before forwarding it on by removing or overwriting any existing instances of the header. Otherwise arbitrary clients can control the header value as seen and used by the backend server. It is important to note that neglecting to prevent header injection does not "fail safe" in that the nominal functionality will still work as expected even when malicious actions are possible. As such, extra care is recommended in ensuring that proper header sanitation is in place.

The communication between a TTRP and backend server needs to be secured against eavesdropping and modification by unintended parties.

The configuration options and request sanitization are necessarily functionally of the respective servers. The other requirements can be met in a number of ways, which will vary based on specific deployments. The communication between a TTRP and backend or origin server, for example, might be authenticated in some way with the insertion and consumption of the "Client-Cert" header occurring only on that connection. Alternatively the network topology might dictate a private network such that the backend application is only able to accept requests from the TTRP and the proxy can only make requests to that server. Other deployments that meet the requirements set forth herein are also possible.

4. IANA Considerations

[[TBD if this draft progresses, register the "Client-Cert" HTTP header field in the "Permanent Message Header Field Names" registry (<https://www.iana.org/assignments/message-headers/message-headers.xhtml>) defined in [RFC3864]]]

5. Normative References

- [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>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.
- [ITU.X690] International Telecommunications Union, "Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", August 2015.
- [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>>.
- [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>>.

6. Informative References

- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC6585] Nottingham, M. and R. Fielding, "Additional HTTP Status Codes", RFC 6585, DOI 10.17487/RFC6585, April 2012, <<https://www.rfc-editor.org/info/rfc6585>>.

- [RFC8705] Campbell, B., Bradley, J., Sakimura, N., and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens", RFC 8705, DOI 10.17487/RFC8705, February 2020, <<https://www.rfc-editor.org/info/rfc8705>>.
- [RFC7239] Petersson, A. and M. Nilsson, "Forwarded HTTP Extension", RFC 7239, DOI 10.17487/RFC7239, June 2014, <<https://www.rfc-editor.org/info/rfc7239>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [I-D.ietf-httpbis-header-structure] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", Work in Progress, Internet-Draft, draft-ietf-httpbis-header-structure-19, 3 June 2020, <<https://tools.ietf.org/html/draft-ietf-httpbis-header-structure-19>>.
- [RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, DOI 10.17487/RFC3864, September 2004, <<https://www.rfc-editor.org/info/rfc3864>>.
- [RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <<https://www.rfc-editor.org/info/rfc7250>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.

Appendix A. Example

In a hypothetical example where a TLS client presents the client and intermediate certificate from Figure 1 when establishing a mutually-authenticated TLS connection with the TTRP, the proxy would send the "Client-Cert" header shown in {#example-header} to the backend. Note that line breaks and whitespace have been added to the value of the header field in Figure 2 for display and formatting purposes only.

```
-----BEGIN CERTIFICATE-----
MIIBqDCCAUG6AwIBAgIBBzAKBggqhkhjOPQQDAjA6MRswGQYDVQQKDBJMZXQncyBB
dXR0ZW50aWNhdGUxGzAZBgNVBAMMEkxBIEludGVybWVkaWw0ZSBDQTAeFw0yMDAx
MTQyMjU1MzNaFw0yMTAxMjU1MzNaMA0xCzAJBgNVBAMMAkxJDMFkwEwYHkoZiZj
0CAQYIKoZIzj0DAQcDQgAE8YnXXfaUgmnMtOXU/IncWalRhebrXmckC8vdgJlp
5Be5F/3YC8OthxM4+k1M6aEAEFcGzkJiNy6J84y7uzo9M6NyMHAwCQYDVROTBAlw
ADAFBgNVHSMEGDAWgBRm3WjLa38lbEYCuicPct0ZaSED2DAOBgNVHQ8BAf8EBAMC
BsAwEwYDVR0lBAwwCgYIKwYBBQUHAWIwHQYDVRORAQH/BBMwEYEPYmRjQGV4YW1w
bGUuYy29tMAoGCCqGSM49BAMCA0gAMEUCIBHda/r1vaL6G3VliL4/Di6YK0Q6bmJe
SkC3dFCOOB8TAiEAX/kHSB4urmiZ0NX5r5XarmPk0wmuydBVoU4hBVZ1yHk=
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIB5jCCAYugAwIBAgIBFjAKBggqhkhjOPQQDAjBWMQswCQYDVQQGEwJVUzEhMBkG
A1UECgwSTGV0J3MgQXV0aGVudG1jYXR1MSowKAYDVQQDDCFMZXMzQncyBBdXR0ZW50
aWNhdGUgUm9vdCBBDXRob3JpdHkwHhcnMjAwMTE0MjEzMTU1MzNaFw0yMDAxMTQy
MjU1MzNaMA0xCzAJBgNVBAMMAkxJDMFkwEwYHkoZiZj0CAQYIKoZIzj0DAQcDQgA
IEludGVybWVkaWw0ZSBDQTBZMBMGBYqGSM49AgEGCCqGSM49AwEHA0IABJf+aA54
RC5pyLAR5yfXVYmNpgd+CGUTDp2KOGhc0gK91zxhHesEYkdXkps2UN8Kati+yHtW
CV3kKhCngGyv7RqjzjBkMB0GA1UdDgQWBm3WjLa38lbEYCuicPct0ZaSED2DAF
BgNVHSMEGDAWgBTEA2Q6eecKu9g9yb5g1bkhhVINGDASBgNVHRMBAf8ECDAGAQH/
AgEAMA4GA1UdDwEB/wQEAwIBhjAKBggqhkhjOPQQDAgNJADBGAiEA5pLvaFwRRkxo
mIAtdIw9D7gC1xzxBl4r28EzmSO1pcCIQCJUSHPsXO9HDIQMUgH69fNDEMhXD3R
RX5gP7kuu2KGMg==
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIICBjCCAaygAwIBAgIJAKS0yiqKtlhoMAoGCCqGSM49BAMCMFYxCzAJBgNVBAYT
AlVTMRswGQYDVQQKDBJMZXQncyBBdXR0ZW50aWNhdGUxKjAoBgNVBAMMIUxldCdz
IEF1dGhlbnRpY2F0ZSBSb290IEF1dGhvcml0eTAeFw0yMDAxMTQyMTU1NDVaFw00
MDAxMDkYMTU1NDVaMFYxCzAJBgNVBAYTA1VTMRswGQYDVQQKDBJMZXQncyBBdXR0
ZW50aWNhdGUxKjAoBgNVBAMMIUxldCdzIEF1dGhlbnRpY2F0ZSBSb290IEF1dGhv
cml0eTBZMBMGBYqGSM49AgEGCCqGSM49AwEHA0IABFoaHU+Z5bPKmGz1YXtCf+E6
HYj62fORaHDOrt+yyh3H/rTcs7ynFfGn+gyFsrSP3Ez88rajv+U2NfD0o0uZ4Pmj
YzBhMB0GA1UdDgQWBm3WjLa38lbEYCuicPct0ZaSED2DAFBgNVHSMEGDAWgBTE
A2Q6eecKu9g9yb5g1bkhhVINGDAPBgNVHRMBAf8EBTADAQH/MA4GA1UdDwEB/wQE
AwIBhjAKBggqhkhjOPQQDAgNIADBFAiEAmAeg1ycKHriqHnaD4M/UDBPQRPkmdcRF
YGMg1Qyrkx4CIB4ivz3wQcQkGhcsUZ1SOImd/lq1Q0FLf09rGfLQPWDC
-----END CERTIFICATE-----
```

Figure 1: Certificate Chain (with client certificate first)

```
Client-Cert: MIIBqDCCAU6gAwIBAgIBBzAKBggqhkJOPQQDAjA6MRswGQYDVQQKDBJM
ZXQncyBBdXRozW50aWNhdGUxGzAZBgNVBAMMEkxBIEludGVybWVkaWF0ZSBDQTAEFw0y
MDAxMTQyMjU1MzNaFw0yMTAxMjMyMjU1MzNaMA0xCzAJBgNVBAMMAkJDMFkwEwYHKOZI
zj0CAQYIKoZIzj0DAQcDQgAE8YnXXfaUgmnMtOXU/IncWalRhebrXmckC8vdgJlp5Be5
F/3YC8OthxM4+k1M6aEAEFcGzkJiNy6J84y7uzo9M6NyMHawCQYDVR0TBAlwADAfBgNV
HSMGDAWgBRm3WjLa38lbEYCuiCPct0ZaSED2DAOBgNVHQ8BAf8EBAMCBsAwEwYDVR0l
BAwwCgYIKwYBBQUHAWIwHQYDVR0RAQH/BBMwEYEPYmRjQGV4YW1wbGUuY29tMAoGCCqG
SM49BAMCA0gAMEUCIBHda/r1vaL6G3VliL4/Di6YK0Q6bMjeSkC3dFCOOB8TAiEax/kH
SB4urmiZ0NX5r5XarmPk0wmuydBVoU4hBVZ1lyhk=
```

Figure 2: Header in HTTP Request to Origin Server

Appendix B. Considerations Considered

B.1. Header Injection

This draft requires that the TTRP sanitize the headers of the incoming request by removing or overwriting any existing instances of the "Client-Cert" header before dispatching that request to the backend application. Otherwise, a client could inject its own "Client-Cert" header that would appear to the backend to have come from the TTRP. Although numerous other methods of detecting/preventing header injection are possible; such as the use of a unique secret value as part of the header name or value or the application of a signature, HMAC, or AEAD, there is no common general standardized mechanism. The potential problem of client header injection is not at all unique to the functionality of this draft and it would therefore be inappropriate for this draft to define a one-off solution. In the absence of a generic standardized solution existing currently, stripping/sanitizing the headers is the de facto means of protecting against header injection in practice today. Sanitizing the headers is sufficient when properly implemented and is normative requirement of Section 3.

B.2. The Forwarded HTTP Extension

The "Forwarded" HTTP header field defined in [RFC7239] allows proxy components to disclose information lost in the proxying process. The TLS client certificate information of concern to this draft could have been communicated with an extension parameter to the "Forwarded" header field, however, doing so would have had some disadvantages that this draft endeavored to avoid. The "Forwarded" header syntax allows for information about a full chain of proxied HTTP requests, whereas the "Client-Cert" header of this document is concerned only with conveying information about the certificate presented by the originating client on the TLS connection to the TTRP (which appears as the server from that client's perspective) to backend applications. The multi-hop syntax of the "Forwarded" header is

expressive but also more complicated, which would make processing it more cumbersome, and more importantly, make properly sanitizing its content as required by Section 3 to prevent header injection considerably more difficult and error prone. Thus, this draft opted for the flatter and more straightforward structure of a single "Client-Cert" header.

B.3. The Whole Certificate and Only the Whole Certificate

Different applications will have varying requirements about what information from the client certificate is needed, such as the subject and/or issuer distinguished name, subject alternative name(s), serial number, subject public key info, fingerprint, etc.. Furthermore some applications, such as "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens" [RFC8705], make use of the entire certificate. In order to accommodate the latter and ensure wide applicability by not trying to cherry-pick particular certificate information, this draft opted to pass the full encoded certificate as the value of the "Client-Cert" header.

The handshake and validation of the client certificate (chain) of the mutually-authenticated TLS connection is performed by the TTRP. With the responsibility of certificate validation falling on the TTRP, only the end-entity certificate is passed to the backend - the root Certificate Authority is not included nor are any intermediates.

[[It has been suggested that more information about the certificate chain might be needed/wanted by the backend application (to independently evaluate the cert chain, for example, although that seems like it would be terribly inefficient) and that any intermediates as well as the root should also be somehow conveyed, which is an area for further discussion should this draft progress. One potential approach suggested by a few folks is to allow some configurability in what is sent along with maybe a prefix token to indicate what's being sent - something like "Client-Cert: FULL <cert> <intermediate> <anchor>" or "Client-Cert: EE <cert>" as the strawman. Or a perhaps a parameter or other construct of [I-D.ietf-httpbis-header-structure] to indicate what's being sent. It's also been suggested that the end-entity certificate by itself might sometimes be too big (esp. e.g., with some post-quantum signature schemes). Hard to account for it both being too much data and not enough data at the same time. But potentially opening up configuration options to send only specific attribute(s) from the client certificate is a possibility for that. In the author's humble opinion the end-entity certificate by itself strikes a good balance for the vast majority of needs and avoids optionality. But, again, this is an area for further discussion should this draft progress.]]

[[It has also been suggested that maybe considerations for [RFC7250] Raw Public Keys is maybe worth considering. This too is this is an area for further discussion and consideration should this draft progress.]]

Appendix C. Acknowledgements

The author would like to thank the following individuals who've contributed in various ways ranging from just being generally supportive of bringing forth the draft to providing specific feedback or content: Evan Anderson, Annabelle Backman, Mike Bishop, Rory Hewitt, Fredrik Jeansson, Benjamin Kaduk, Torsten Lodderstedt, Kathleen Moriarty, Mark Nottingham, Mike Ounsworth, Matt Peterson, Eric Rescorla, Justin Richer, Michael Richardson, Joe Salowey, Rich Salz, Mohit Sethi, Rifaat Shekh-Yusef, Travis Spencer, Nick Sullivan, Peter Wu, and Hans Zandbelt.

[[Please let me know if you've been erroneously omitted or if you prefer not to be named]]

Appendix D. Document History

[[To be removed by the RFC Editor before publication as an RFC (should that come to pass)]]

draft-bdc-something-something-certificate-05

- * Change intended status of the draft to Informational
- * Editorial updates and (hopefully) clarifications

draft-bdc-something-something-certificate-04

- * Update reference from draft-ietf-oauth-mtls to RFC8705

draft-bdc-something-something-certificate-03

- * Expanded [[further discussion notes]] to capture some of the feedback in and around the presentation of the draft in SECDISPATCH at IETF 107 and add those who've provided such feedback to the acknowledgements

draft-bdc-something-something-certificate-02

- * Editorial tweaks + [[further discussion notes]]

draft-bdc-something-something-certificate-01

- * Use the RFC v3 Format or die trying

draft-bdc-something-something-certificate-00

- * Initial draft after a time constrained and rushed secdispatch presentation (<https://datatracker.ietf.org/meeting/106/materials/slides-106-secdispatch-securing-protocols-between-proxies-and-backend-http-servers-00>) at IETF 106 in Singapore with the recommendation to write up a draft (at the end of the minutes (<https://datatracker.ietf.org/meeting/106/materials/minutes-106-secdispatch>)) and some folks expressing interest despite the rather poor presentation

Author's Address

Brian Campbell
Ping Identity

Email: bcampbell@pingidentity.com

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: 16 July 2022

K. Paine
Splunk Inc.
O. Whitehouse
NCC Group
J. Sellwood
Twilio
A. Shaw
UK National Cyber Security Centre
12 January 2022

Indicators of Compromise (IoCs) and Their Role in Attack Defence
draft-paine-smart-indicators-of-compromise-04

Abstract

Cyber defenders frequently rely on Indicators of Compromise (IoCs) to identify, trace, and block malicious activity in networks or on endpoints. This draft reviews the fundamentals, opportunities, operational limitations, and best practices of IoC use. It highlights the need for IoCs to be detectable in implementations of Internet protocols, tools, and technologies - both for the IoCs' initial discovery and their use in detection - and provides a foundation for new approaches to operational challenges in network security.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 16 July 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

This document may not be modified, and derivative works of it may not be created, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	3
1.1. Requirements Language	3
2. Terminology	3
3. IoC Fundamentals	4
3.1. IoC Types and the Pyramid of Pain	4
3.2. IoC Lifecycle	8
3.2.1. Discovery	8
3.2.2. Assessment	9
3.2.3. Sharing	9
3.2.4. Deployment	10
3.2.5. Detection	10
3.2.6. Reaction	10
3.2.7. End of Life	10
4. Using IoCs Effectively	10
4.1. Opportunities	11
4.1.1. IoCs underpin and enable multiple layers of the modern defence-in-depth strategy	11
4.1.2. IoCs can be used even with limited resources	12
4.1.3. IoCs have a multiplier effect on attack defence effort	12
4.1.4. IoCs are easily shared	13
4.1.5. IoCs can provide significant time savings	13
4.1.6. IoCs allow for discovery of historic attacks	14
4.1.7. IoCs can be attributed to specific threats	14
4.2. Case Studies	14
4.2.1. Introduction	14
4.2.2. Cobalt Strike	15
4.2.2.1. Overall TTP	15
4.2.2.2. IoCs	15
4.2.3. APT33	16
4.2.3.1. Overall TTP	16
4.2.3.2. IoCs	17
5. Operational Limitations	17

5.1. Time and Effort	17
5.1.1. Fragility	17
5.1.2. Discoverability	18
5.2. Precision	19
5.2.1. Specificity	19
5.2.2. Dual and Compromised Use	20
5.3. Privacy	20
5.4. Automation	21
6. Best Practice	22
6.1. Comprehensive Coverage and Defence-in-Depth	22
6.2. Security Considerations	24
7. Conclusions	24
8. IANA Considerations	25
9. Acknowledgements	25
10. Informative References	25
Authors' Addresses	27

1. Introduction

This draft describes the various types of Indicator of Compromise (IoC) and how they are used effectively in attack defence (often called cyber defence). It introduces concepts such as the Pyramid of Pain [PoP] and the IoC lifecycle to highlight how IoCs may be used to provide a broad range of defences. This draft provides best practice for implementers of controls based on IoCs, as well as potential operational limitations. Two case studies which demonstrate the usefulness of IoCs for detecting and defending against real world attacks are included. One case study involves an intrusion set (a collection of indicators for a specific attack) known as APT33 and the other an attack tool called Cobalt Strike. This document is not a comprehensive report of APT33 or Cobalt Strike and is intended to be read alongside publicly published reports (referred to as open source material among intelligence practitioners) on these threats (for example, [Symantec] and [NCCGroup], respectively).

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Terminology

Attack defence: the activity of providing cyber security to an environment through the prevention, detection and response to attempted and successful cyber intrusions. Successful defence is achieved through the blocking, monitoring and response to adversarial activity at a network, endpoint or application levels.

Command and control (C2) server: an attacker-controlled server used to communicate with, send commands to and receive data from compromised machines. Communication between a C2 server and compromised hosts is called command and control traffic.

Domain Generation Algorithm (DGA): used in malware strains to generate domain names periodically. Adversaries may use DGAs to dynamically identify a destination for C2 traffic, rather than relying on a list of static IP addresses or domains that can be blocked more easily.

Kill chain: a model for conceptually breaking down a cyber intrusion to allow defenders to think about, discuss, plan for, and implement controls to defend discrete phases of an attacker's activity [KillChain].

Tactics, Techniques, and Procedures (TTPs): the way an adversary undertakes activities in the kill chain - the choices made, methods followed, tools and infrastructure used, protocols employed, and commands executed. If they are distinct enough, aspects of an attacker's TTPs can form specific Indicators of Compromise (IoCs), as if they were a fingerprint.

3. IoC Fundamentals

3.1. IoC Types and the Pyramid of Pain

Indicators of Compromise (IoCs) are observable artefacts relating to an attacker or their activities, such as their tactics, techniques, procedures, and associated tooling and infrastructure. These indicators can be observed at network or endpoint (host) levels and can, with varying degrees of confidence, help network defenders (blue teams) to pro-actively block malicious traffic or code execution, determine a cyber intrusion occurred, or associate discovered activity to a known intrusion set and thereby potentially identify additional avenues for investigation. Examples of protocol-related IoCs can include:

- * IPv4 and IPv6 addresses in network traffic.
- * DNS domain names in network traffic, resolver caches or logs.
- * TLS Server Name Indication values in network traffic.
- * Code signing certificates in binaries or TLS certificate information (such as SHA256 hashes) in network traffic.

- * Cryptographic hashes (e.g. MD5, SHA1 or SHA256) of malicious binaries or scripts when calculated from network traffic or file system artefacts.
- * Attack tools (such as Mimikatz [Mimikatz]) and their code structure and execution characteristics.
- * Attack techniques, such as Kerberos golden tickets [GoldenTicket] which can be observed in network traffic or system artefacts.

The common types of IoC form a 'Pyramid of Pain' [PoP] that informs prevention, detection, and mitigation strategies. Each IoC type's place in the pyramid represents how much 'pain' a typical adversary experiences as part of changing the activity that produces that artefact. The greater pain an adversary experiences (towards the top) the less likely they are to change those aspects of their activity and the longer the IoC is likely to reflect the attacker's intrusion set - i.e., the less fragile those IoCs will be from a defender's perspective. The layers of the PoP commonly range from hashes up to TTPs, with the pain ranging from simply recompiling code to creating a whole new attack strategy. Other types of IoC do exist and could be included in an extended version of the PoP should that assist the defender to understand and discuss intrusion sets most relevant to them.

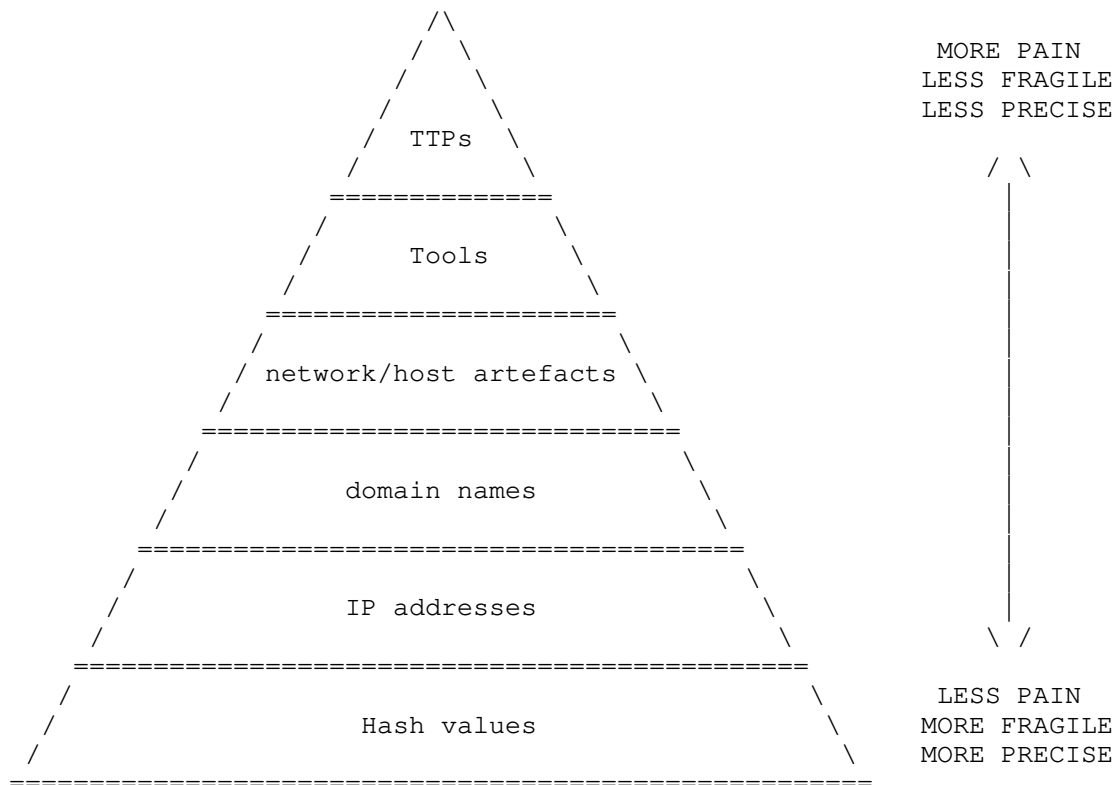


Figure 1

On the lowest (and least painful) level are hashes of malicious files. These are easy for a defender to gather and can be deployed to firewalls or endpoint protection to block malicious downloads or prevent code execution. While IoCs aren't the only way for defenders to do this kind of blocking, they are a quick, convenient, and unintrusive method. Hashes are precise detections for individual files based on their binary content. To subvert this defence, however, an adversary need only recompile code, or otherwise modify the file content with some trivial changes, to modify the hash value.

The next two levels are IP addresses and domain names. Interactions with these may be blocked, with varying false positive rates (misidentifying non-malicious traffic as malicious, see Section 5), and often cause more pain to an adversary to subvert than file hashes. The adversary may have to change IP ranges, find a new provider, and change their code (e.g., if the IP address is hard-coded, rather than resolved). Domain names are more specific than IP addresses (as multiple domain names may be associated with a single IP address) and are more painful for an adversary to change.

Network and endpoint artefacts, such as a malware's beaconing pattern on the network or the modified timestamps of files touched on an endpoint, are harder still to change as they relate specifically to the attack taking place and, in some cases, may not be under the direct control of the attacker. However, more sophisticated attackers use TTPs or tooling that provide flexibility at this level (such as Cobalt Strike's malleable command and control [COBALT]) or a means by which some artefacts can be masked (see [Timestamp]).

Tools and TTPs form the top two levels of the pyramid; these levels describe a threat actor's methodology – the way they perform the attack. The tools level refers specifically to the software (and less frequently hardware) used to conduct the attack, whereas the TTPs level picks up on all the other aspects of the attack strategy. IoCs at these levels are more complicated and complex – for example they can include the details of how an attacker deploys malicious code to perform reconnaissance of a victim's network, that pivots laterally to a valuable endpoint, and then downloads a ransomware payload. TTPs and tools take intensive effort to diagnose on the part of the defender, but they are fundamental to the attacker and campaign and hence incredibly painful for the adversary to change.

The variation in discoverability of IoCs is indicated by the numbers of IoCs in the open threat intelligence community Alienvault [ALIENVAULT]. As of June 2021, Alienvault contained:

- * Groups (i.e., combinations of TTPs): 441
- * Malware families (i.e., tools): ~24,000
- * URL: 1,976,224
- * Domain names: 34,959,787
- * IPv4 addresses: 4,305,036
- * SHA256 hash values: 4,767,891

The number of domain names appears out of sync with the other counts, which reduce on the way up the PoP. This discrepancy warrants further research; however, a contributing factor may be the fact that threat actors use domain names to masquerade as legitimate organisations and so have added incentive for creating new domain names as they are identified and confiscated.

3.2. IoC Lifecycle

To be of use to defenders, IoCs must first be discovered, assessed, shared, and deployed. When a logged activity is identified and correlated to an IoC this detection triggers a reaction by the defender which may include an investigation, potentially leading to more IoCs being discovered, assessed, shared, and deployed. This cycle continues until such time that the IoC is determined to no longer be relevant, at which point it is removed from the control space.

3.2.1. Discovery

IoCs are often discovered initially through manual investigation or automated analysis. They can be discovered in a range of sources, including in networks and at endpoints. They must either be extracted from logs monitoring protocol runs, code execution or system operations (in the case of hashes, IP addresses, domain names, and network or endpoint artefacts), or be determined through analysis of attack activity or tooling. In some cases, discovery may be a reactive process, where IoCs from past or current attacks are identified from the traces left behind. However, discovery may also result from proactive hunting for potential future IoCs extrapolated from knowledge of past events (such as from identifying attacker infrastructure by monitoring domain name registration patterns).

Crucially, for an IoC to be discovered, the indicator must be extractable from the internet protocol, tool, or technology it is associated with. Identifying a particular protocol run related to an attack is of limited benefit if indicators cannot be extracted and subsequently associated with a later related run of the same, or a different, protocol. If it is not possible to tell the source or destination of malicious attack traffic, it will not be possible to identify and block subsequent attack traffic either.

3.2.2. Assessment

Defenders may treat different IoCs differently, depending on the IoCs' quality and the defender's needs and capabilities. Defenders may, for example, place differing trust in IoCs depending on their source, freshness, confidence level, or the associated threat. These decisions rely on associated contextual information recovered at the point of discovery or provided when the IoC was shared.

An IoC without context is not much use for network defence. On the other hand, an IoC delivered with context (for example the threat actor it relates to, its role in an attack, the last time it was seen in use, its expected lifetime, or other related IoCs) allows a network defender to make an informed choice on how to use it to protect their network - for example, whether to simply log it, actively monitor it, or out-right block it.

3.2.3. Sharing

Once discovered and assessed, IoCs are most helpful when then shared at scale so many individuals and organisations can defend themselves. An IoC may be shared individually (with appropriate context) in an unstructured manner or may be packaged alongside many other IoCs in a standardised format, such as Structured Threat Information Expression [STIX], for distribution via a structured feed, such as one implementing Trusted Automated Exchange of Intelligence Information [TAXII], or through a Malware Information Sharing Platform [MISP].

While some security companies and some membership-based groups (often dubbed Information Sharing and Analysis Centres (ISACs)) provide paid intel feeds containing IoCs, there are various free IoC sources available from individual security researchers up through small trust groups to national governmental cyber security organisations and international Computer Emergency Response Teams (CERTs). Whomever they are, sharers commonly indicate the extent to which receivers may further distribute IoCs using the Traffic Light Protocol [TLP]. At its simplest, this indicates that the receiver may share with anyone (TLP WHITE), share within the defined sharing community (TLP GREEN), share within their organisation (TLP AMBER), or not share with anyone outside the original specific IoC exchange (TLP RED).

3.2.4. Deployment

For IoCs to provide defence-in-depth (see Section 6.1), which is one of their key strengths, and so cope with different points of failure, they should be deployed in controls monitoring networks and endpoints through solutions that have sufficient privilege to act on them. Wherever IoCs exist they need to be made available to security controls and associated apparatus to ensure they can be deployed quickly and widely. While IoCs may be manually assessed after discovery or receipt, significant advantage may be gained by automatically ingesting, processing, assessing, and deploying IoCs from logs or intel feeds to the appropriate security controls.

3.2.5. Detection

Security controls with deployed IoCs monitor their relevant control space and trigger a generic or specific reaction upon detection of the IoC in monitored logs.

3.2.6. Reaction

The reaction to an IoC's detection may differ depending on factors such as the capabilities and configuration of the control it is deployed in, the assessment of the IoC, and the properties of the log source in which it was detected. For example, a connection to a known botnet C2 server may indicate a problem but does not guarantee it, particularly if the server is a compromised host still performing some other legitimate functions. Common reactions include event logging, triggering alerts, and blocking or terminating the source of the activity.

3.2.7. End of Life

How long an IoC remains useful varies and is dependent on factors including initial confidence level, fragility, and precision of the IoC (discussed further in Section 5). In some cases, IoCs may be automatically 'aged' based on their initial characteristics and so will reach end of life at a predetermined time. In other cases, IoCs may become invalidated due to a shift in the threat actor's TTPs (e.g., resulting from a new development or their discovery) or due to remediation action taken by a defender. End of life may also come about due to an activity unrelated to attack or defence, such as when a third-party service used by the attacker changes or goes offline. Whatever the cause, IoCs should be removed from detection at the end of their life to reduce the likelihood of false positives.

4. Using IoCs Effectively

4.1. Opportunities

IoCs offer a variety of opportunities to cyber defenders as part of a modern defence-in-depth strategy. No matter the size of an organisation, IoCs can provide an effective, scalable, and efficient defence mechanism against classes of attack from the latest threats or specific intrusion sets which may have struck in the past.

4.1.1. IoCs underpin and enable multiple layers of the modern defence-in-depth strategy

Firewalls, Intrusion Detection Systems (IDS), and Intrusion Prevention Systems (IPS) all employ IoCs to identify and mitigate threats across networks. Anti-Virus (AV) and Endpoint Detection and Response (EDR) products deploy IoCs via catalogues or libraries to all supported client endpoints. Security Incident Event Management (SIEM) platforms compare IoCs against aggregated logs from various sources - network, endpoint, and application. Of course, IoCs do not address all attack defence challenges - but they form a vital tier of any organisation's layered defence. Some types of IoC may be present across all those controls while others may be deployed only in certain layers. Further, IoCs relevant to a specific kill chain may only reflect activity performed during a certain phase and so need to be combined with other IoCs or mechanisms for complete coverage of the kill chain as part of an intrusion set.

As an example, open source malware can be deployed by many different actors, each using their own TTPs and infrastructure. However, if the actors use the same executable, the hash remains the same and this IoC can be deployed in endpoint protection to block execution regardless of individual actor, infrastructure, or other TTPs. Should this defence fail in a specific case, for example if an actor recompiles the executable binary producing a unique hash, other defences can prevent them progressing further through their attack - for instance, by blocking known malicious domain name look-ups and thereby preventing the malware calling out to its C2 infrastructure.

Alternatively, another malicious actor may regularly change their tools and infrastructure (and thus the indicator intrusion set) deployed across different campaigns, but their access vectors may remain consistent and well-known. In this case, this access TTP can be recognised and proactively defended against even while there is uncertainty of the intended subsequent activity. For example, if their access vector consistently exploits a vulnerability in software, regular and estate-wide patching can prevent the attack from taking place. Should these pre-emptive measures fail however, other IoCs observed across multiple campaigns may be able to prevent the attack at later stages in the kill chain.

4.1.2. IoCs can be used even with limited resources

IoCs are inexpensive, scalable, and easy to deploy, making their use particularly beneficial for smaller entities, especially where they are exposed to a significant threat. For example, a small manufacturing subcontractor in a supply chain producing a critical, highly specialised component may represent an attractive target because there would be disproportionate impact on both the supply chain and the prime contractor if it were compromised. It may be reasonable to assume that this small manufacturer will have only basic security (whether internal or outsourced) and while it is likely to have comparatively less resources to manage the risks it faces compared to larger partners, it can still leverage IoCs to great effect. Small entities like this can deploy IoCs to give a baseline protection against known threats without having access to a well-resourced, mature defensive team and the threat intelligence relationships necessary to perform resource-intensive investigations. One reason for this is that use of IoCs does not require the same intensive training as needed for more subjective controls, such as those based on manual analysis of tipped machine learning events. In this way, a major part of the appeal of IoCs is that they can afford some level of protection to organisations across spectrums of resource capability, maturity, and sophistication.

4.1.3. IoCs have a multiplier effect on attack defence effort

Individual IoCs can provide widespread protection that scales effectively for defenders. Within a single organisation, simply blocking one IoC may protect thousands of users and that blocking may be performed (depending on the IoC type) across multiple security controls monitoring numerous different types of activity within networks, endpoints, and applications. While discovering one IoC can be intensive, once shared via well-established routes (as discussed in Section 3.2.2) that individual IoC may, further, protect thousands of organisations and so all of their users. The prime contractor from our earlier example can supply IoCs to the small subcontractor and so further uplift that smaller entity's defensive capability and at the same time protect itself and its interests.

Not only may multiple organisations benefit through directly receiving shared IoCs, but they may also benefit through the IoCs' application in services they utilise. In the case of an ongoing email phishing campaign, IoCs can be monitored, discovered, and deployed quickly and easily by individual organisations. However, if they are deployed quickly via a mechanism such as a protective DNS filtering service, they can be more effective still - an email campaign may be mitigated before some organisations' recipients ever click the link or before some malicious payloads can call out for instructions. Through such approaches other parties can be protected without additional effort.

4.1.4. IoCs are easily shared

There is significant benefit to be had from the sharing of IoCs and they can be easily shared for two main reasons: firstly, indicators are easy to distribute as they are textual and so in small numbers are frequently exchanged in emails, blog posts, or technical reports; secondly, standards such as MISP Core [MISPCORE], OpenIOC [OPENIOC], and STIX [STIX] provide well-defined formats for sharing large collections or regular sets of IoC along with all the associated context. Quick and easy sharing of IoCs gives blanket coverage for organisations and allows widespread mitigation in a timely fashion - they can be shared with systems administrators, from small to large organisations and from large teams to single individuals, allowing them all to implement defences on their networks.

4.1.5. IoCs can provide significant time savings

Not only are there time savings from sharing IoCs, saving duplication of investigation effort, but deploying them automatically at scale is seamless for many enterprises. Where automatic deployment of IoCs is working well, organisations and users get blanket protection with minimal human intervention and minimal effort, a key goal of attack defence. The ability to do this at scale and at pace is often vital when responding to agile threat actors that may change their intrusion set frequently and so the relevant IoCs also change. Conversely, protecting a complex network without automatic deployment of IoCs could mean manually updating every single endpoint or network device consistently and reliably to the same security state. The work this entails (including locating assets and devices, polling for logs and system information, and manually checking patch levels) introduces complexity and a need for skilled analysts and engineers. While it is still necessary to invest effort to eliminate false positives when widely deploying IoCs, the cost and effort involved can be far smaller than the work entailed in reliably manually updating all endpoint and network devices - for example, particularly on legacy systems that may be particularly complicated, or even

impossible, to update.

4.1.6. IoCs allow for discovery of historic attacks

A network defender can use recently acquired IoCs in conjunction with historic data, such as logged DNS queries or email attachment hashes, to hunt for signs of past compromise. Not only can this technique help to build up a clear picture of past attacks, but it also allows for retrospective mitigation of the effects of any previous intrusion. This opportunity is reliant on historic data not having been compromised itself, by a technique such as Timestomp [Timestomp], and not being incomplete due to data retention policies, but is nonetheless valuable for detecting and remediating past attacks.

4.1.7. IoCs can be attributed to specific threats

Deployment of various modern security controls, such as firewall filtering or EDR, come with an inherent trade-off between breadth of protection and various costs, including the risk of false positives (see Section 5.2), staff time, and pure financial costs. Organisations can use threat modelling and information assurance to assess and prioritise risk from identified threats and to determine how they will mitigate or accept each of them. Contextual information tying IoCs to specific threats or actors and shared alongside the IoCs enables organisations to focus their defences against particular risks and so allows them the technical freedom and capability to choose their risk posture and defence methods. Producing this contextual information before sharing IoCs can take intensive analytical effort as well as specialist tools and training. At its simplest it can involve documenting sets of IoCs from multiple instances of the same attack campaign, say from multiple unique payloads (and therefore with distinct file hashes) from the same source and connecting to the same C2 server. A more complicated approach is to cluster similar combinations of TTPs seen across multiple campaigns over a period of time. This can be used alongside detailed malware reverse engineering and target profiling, overlaid on a geopolitical and criminal backdrop, to infer attribution to a single threat actor.

4.2. Case Studies

4.2.1. Introduction

The following two case studies illustrate how IoCs may be identified in relation to threat actor tooling (in the first) and a threat actor campaign (in the second). The case studies further highlight how these IoCs may be used by cyber defenders.

4.2.2. Cobalt Strike

Cobalt Strike [COBALT] is a commercial attack framework that consists of an implant framework (beacon), network protocol, and a C2 server. The beacon and network protocol are highly malleable, meaning the protocol representation 'on the wire' can be easily changed by an attacker to blend in with legitimate traffic. The proprietary beacon supports TLS encryption overlaid with a custom encryption scheme based on a public-private keypair. The product also supports other techniques, such as domain fronting [DFRONT], in attempt to avoid obvious passive detection by static network signatures.

4.2.2.1. Overall TTP

A beacon configuration describes how the implant should operate and communicate with its C2 server. This configuration also provides ancillary information such as the Cobalt Strike user's licence watermark.

4.2.2.2. IoCs

Tradecraft has been developed that allows the fingerprinting of C2 servers based on their responses to specific requests. This allows the servers to be identified and then their beacon configurations to be downloaded and the associated infrastructure addresses extracted as IoCs.

The resulting mass IoCs for Cobalt Strike are:

- * IP addresses of the C2 servers
- * domain names used

Whilst these IoCs need to be refreshed regularly (due to the ease of which they can be changed), the authors' experience of protecting public sector organisations show these IoCs are effective for disrupting threat actor operations that use Cobalt Strike.

These IoCs can be used to check historical data for evidence of past compromise, as well as deployed to detect or block future infection in a timely manner, thereby contributing to preventing the loss of user and system data.

4.2.3. APT33

In contrast to the first case study, this describes a current campaign by the threat actor APT33, also known as Elfin and Refined Kitten (see [Symantec]). APT33 has been assessed by industry to be a state-sponsored group [FireEye2], yet in this case study, IoCs still gave defenders an effective tool against such a powerful adversary. The group has been active since at least 2015 and is known to target a range of sectors including petrochemical, government, engineering, and manufacturing. Activity has been seen in countries across the globe, but predominantly in the USA and Saudi Arabia.

4.2.3.1. Overall TTP

The techniques employed by this actor exhibit a relatively low level of sophistication considering it is a state-sponsored group; typically, APT33 performs spear phishing (sending targeted malicious emails to a limited number of pre-selected recipients) with document lures that imitate legitimate publications. User interaction with these lures executes the initial payload and enables APT33 to gain initial access. Once inside a target network, APT33 attempts to pivot to other machines to gather documents and gain access to administrative credentials. In some cases, users are tricked into providing credentials that are then used with RULER, a freely available tool that allows exploitation of an email client. The attacker, in possession of a target's password, uses RULER to access the target's mail account and embeds a malicious script which will be triggered when the mail client is next opened, resulting in the execution of malicious code (often additional malware retrieved from the Internet) (see [FireEye]).

APT33 sometimes deploys a destructive tool which overwrites the master boot record (MBR) of the hard drives in as many PCs as possible. This type of tool, known as a wiper, results in data loss and renders devices unusable until the operating system is reinstalled. In some cases, the actor uses administrator credentials to invoke execution across a large swathe of a company's IT estate at once; where this isn't possible the actor may attempt to spread the wiper first manually or by using worm-like capabilities against unpatched vulnerabilities on the networked computers.

4.2.3.2. IoCs

As a result of investigations by a partnership of industry and the UK's National Cyber Security Centre (NCSC), a set of IoCs were compiled and shared with both public and private sector organisations so network defenders could search for them in their networks. Detection of these IoCs is likely indicative of APT33 targeting and could indicate potential compromise and subsequent use of destructive malware. Network defenders could also initiate processes to block these IoCs to foil future attacks. This set of IoCs comprised:

- * 9 hashes and email subject lines
- * 5 IP addresses
- * 7 domain names

5. Operational Limitations

The different IoC types inherently embody a set of trade-offs for defenders between the risk of false positives (misidentifying non-malicious traffic as malicious) and the risk of failing to identify attacks. The attacker's relative pain of modifying attacks to subvert known IoCs, as discussed using the Pyramid of Pain (PoP) in Section 3.1, inversely correlates with the fragility of the IoC and with the precision with which the IoC identifies an attack. Research is needed to elucidate the exact nature of these trade-offs between pain, fragility, and precision.

5.1. Time and Effort

5.1.1. Fragility

As alluded to in Section 3.1, the Pyramid of Pain can be thought of in terms of fragility for the defender as well as pain for the attacker. The less painful it is for the attacker to change an IoC, the more fragile that IoC is as a defence tool. It is relatively simple to determine the hash value for various malicious file attachments observed as lures in a phishing campaign and to deploy these through AV or an email gateway security control. However, those hashes are fragile and can (and often will) be changed between campaigns. Malicious IP addresses and domain names can also be changed between campaigns, but this happens less frequently due to the greater pain of managing infrastructure compared to altering files, and so IP addresses and domain names provide a less fragile detection capability.

This does not mean the more fragile IoC types are worthless. Firstly, there is no guarantee a fragile IoC will change, and if a known IoC isn't changed by the attacker but wasn't blocked then the defender missed an opportunity to halt an attack in its tracks. Secondly, even within one IoC type, there is variation in the fragility depending on the context of the IoC. The file hash of a phishing lure document (with a particular theme and containing a specific staging server link) may be more fragile than the file hash of a remote access trojan payload the attacker uses after initial access. That in turn may be more fragile than the file hash of an attacker-controlled post-exploitation reconnaissance tool that doesn't connect directly to the attacker's infrastructure. Thirdly, some threats and actors are more capable or inclined to change than others, and so the fragility of an IoC for one may be very different to an IoC of the same type for another actor.

Ultimately, fragility is a defender's concern that impacts the ongoing efficacy of each IoC and will factor into decisions about end of life. However, it should not prevent adoption of individual IoCs unless there are significantly strict resource constraints that demand down-selection of IoCs for deployment. More usually, defenders researching threats will attempt to identify IoCs of varying fragilities for a particular kill chain to provide the greatest chances of ongoing detection given available investigative effort (see Section 5.1.2) and while still maintaining precision (see Section 5.2).

Finally, it is worth noting that fragility can apply to an entire class of IoCs for a range of reasons; for example, IPv4 addresses are becoming increasingly fragile due to addresses growing scarce, widespread use of cloud services, and the ease with which domain names can be moved from one hosting provider to another (thus changing IP range).

5.1.2. Discoverability

To be used in attack defence, IoCs must first be discovered through proactive hunting or reactive investigation. As noted in Section 3.1, IoCs in the tools and TTPs levels of the PoP require intensive effort and research to discover. However, it is not just an IoC's type that impacts its discoverability. The sophistication of the actor, their TTPs, and their tooling play a significant role, as does whether the IoC is retrieved from logs after the attack or extracted from samples or infected systems earlier.

For example, on an infected endpoint it may be possible to identify a malicious payload and then extract relevant IoCs, such as the file hash and its C2 server address. If the attacker used the same static

payload throughout the attack this single file hash value will cover all instances. If, however, the attacker diversified their payloads, that hash can be more fragile and other hashes may need to be discovered from other samples used on other infected endpoints. Concurrently, the attacker may have simply hard-coded configuration data into the payload, in which case the C2 server address can be easy to recover. Alternatively, the address can be stored in an obfuscated persistent configuration either within the payload (e.g., within its source code or associated resource) or the infected endpoint's filesystem (e.g., using alternative data streams [ADS]) and thus requiring more effort to discover. Further, the attacker may be storing the configuration in memory only or relying on a domain generation algorithm (DGA) to generate C2 server addresses on demand. In this case, extracting the C2 server address can require a memory dump or the execution or reverse engineering of the DGA, all of which increase the effort still further.

If the malicious payload has already communicated with its C2 server, then it may be possible to discover that C2 server address IoC from network traffic logs more easily. However, once again multiple factors can make discoverability more challenging, such as the increasing adoption of HTTPS for malicious traffic - meaning C2 communications blend in with legitimate traffic, and can be complicated to identify. Further, some malwares obfuscate their intended destinations by using alternative DNS resolution services (e.g., OpenNIC [OPENNIC]) or by performing transformation operations on resolved IP addresses to determine the real C2 server address encoded in the DNS response [LAZARUS].

5.2. Precision

5.2.1. Specificity

Alongside pain and fragility, the PoP's levels can also be considered in terms of how precise the defence can be, with the false positive rate usually increasing as we move up the pyramid to less specific IoCs. A hash value identifies a particular file, such as an executable binary, and given a suitable cryptographic hash function the false positives are effectively nil; by suitable we mean one with preimage resistance and strong collision resistance. In comparison, IoCs in the upper levels (such as some network artefacts or tool fingerprints) may apply to various malicious binaries, and even benign software may share the same identifying characteristics. For example, threat actor tools making web requests may be identified by the user-agent string specified in the request header. However, this value may be the same as used by legitimate software, either by the attacker's choice or through use of a common library.

It should come as no surprise that the more specific an IoC the more fragile it is – as things change, they move outside of that specific focus. While less fragile IoCs may be desirable for their robustness and longevity, this must be balanced with the increased chance of false positives from their broadness. One way in which this balance is achieved is by grouping indicators and using them in combination. While two low-specificity IoCs for a particular attack may each have chances of false positives, when observed together they may provide greater confidence of an accurate detection of the relevant kill chain.

5.2.2. Dual and Compromised Use

As noted in Section 3.2.2, the context of an IoC, such as the way in which the attacker uses it, may equally impact the precision with which that IoC detects an attack. An IP address representing an attacker's staging server, from which their attack chain downloads subsequent payloads, offers a precise IP address for attacker-owned infrastructure. However, it will be less precise if that IP address is associated with a cloud hosting provider and it is regularly reassigned from one user to another; and it will be less precise still if the attacker compromised a legitimate web server and is abusing the IP address alongside the ongoing legitimate use.

In a similar manner, a file hash representing an attacker's custom remote access trojan will be very precise; however, a file hash representing a common enterprise remote administration tool will be less precise depending on whether the defender organisation usually uses that tool for legitimate systems administration or not. Notably, such dual use indicators are context specific both in whether they are usually used legitimately and in the way they are used in a particular circumstance. Use of the remote administration tool may be legitimate for support staff during working hours, but not generally by non-support staff, particularly if observed outside of that employee's usual working hours.

It is reasons such as these that context is so important when sharing and using IoCs.

5.3. Privacy

As noted in Section 3.2.2, context is critical to effective detection using IoCs. However, at times, defenders may feel there are privacy concerns with how much to share about a cyber intrusion, and with whom. For example, defenders may generalise the IoCs' description of the attack, by removing context to facilitate sharing. This generalisation can result in an incomplete set of IoCs being shared or IoCs being shared without clear indication of what they represent

and how they are involved in an attack. The sharer will consider the privacy trade-off when generalising the IoC, and should bear in mind that the loss of context can greatly reduce the utility of the IoC for those they share with.

Self-censoring by sharers appears more prevalent and more extensive when sharing IoCs into groups with more members, into groups with a broader range of perceived member expertise (particularly the further the lower bound extends below the sharer's perceived own expertise), and into groups that do not maintain strong intermember trust. Trust within such groups appears often strongest where members: interact regularly; have common backgrounds, expertise, or challenges; conform to behavioural expectations (such as by following defined handling requirements and not misrepresenting material they share); and reciprocate the sharing and support they receive. Research opportunities exist to determine how IoC sharing groups' requirements for trust and members' interaction strategies vary and whether sharing can be optimised or incentivised, such as by using game theoretic approaches.

5.4. Automation

While IoCs can be effectively utilised by organisations of various sizes and resource constraints, as discussed in Section 4.1.2, automation of IoC ingestion, processing, assessment, and deployment is critical for managing them at scale. Manual oversight and investigation may be necessary intermittently, but a reliance on manual processing and searching only works at small scale or for occasional cases.

The adoption of automation can also enable faster and easier correlation of IoC detections across log sources, time, and space. Thereby, the response can be tailored to reflect the number and overlap of detections from a particular intrusion set, and the necessary context can be presented alongside the detection when generating any alerts for defender review. While manual processing and searching may be no less accurate (although IoC transcription errors are a common problem during busy incidents), the correlation and cross-referencing necessary to provide the same degree of situational awareness is much more time consuming.

A third important consideration when performing manual processing is the longer phase monitoring and adjustment necessary to effectively age out IoCs as they become irrelevant or, more crucially, inaccurate. Manual implementations must often simply include or exclude an IoC, as anything more granular is time consuming and complicated to manage. In contrast, automations can support a gradual reduction in confidence scoring enabling IoCs to contribute but not individually disrupt a detection as their specificity reduces.

6. Best Practice

6.1. Comprehensive Coverage and Defence-in-Depth

IoCs provide the defender with a range of options across the Pyramid of Pain's (PoP) layers, enabling them to balance precision and fragility to give high confidence detections that are practical and useful. Broad coverage of the PoP is important as it allows the defender to cycle between high precision but high fragility options and more robust but less precise indicators. As fragile indicators are changed, the more robust IoCs allow for continued detection and faster rediscovery. For this reason, it's important to collect as many IoCs as possible across the whole PoP.

At the top of the PoP, TTPs identified through anomaly detection and machine learning are more likely to have false positives, which gives lower confidence and, vitally, requires better trained analysts to understand and implement the defences. However, these are very painful for attackers to change and so when tuned appropriately provide a robust detection. Hashes, at the bottom, are precise and easy to deploy but are fragile and easily changed within and across campaigns by malicious actors.

Endpoint Detection and Response (EDR) or Anti-Virus (AV) are often the first port of call for protection from intrusion but endpoint solutions aren't a panacea. One issue is that there are many environments where it is not possible to keep them updated, or in some cases, deploy them at all. For example, the Owari botnet, a Mirai variant [Owari], exploited Internet of Things (IoT) devices where such solutions could not be deployed. It is because of such gaps, where endpoint solutions can't be relied on (see [EVOLVE]), that a defence-in-depth approach is commonly advocated, using a blended approach that includes both network and endpoint defences.

If an attack happens, then you hope an endpoint solution will pick it up. If it doesn't, it could be for many good reasons: the endpoint solution could be quite conservative and aim for a low false-positive rate; it might not have ubiquitous coverage; or it might only be able

to defend the initial step of the kill chain [KillChain]. In the worst cases, the attack specifically disables the endpoint solution or the malware is brand new and so won't be recognised.

In the middle of the pyramid, IoCs related to network information (such as domains and IP addresses) can be particularly useful. They allow for broad coverage, without requiring each and every endpoint security solution to be updated, as they may be detected and enforced in a more centralised manner at network choke points (such as proxies and gateways). This makes them particularly useful in contexts where ensuring endpoint security isn't possible such as "Bring Your Own Device" (BYOD), Internet of Things (IoT) and legacy environments. It's important to note that these network-level IoCs can also protect against compromised endpoints when these IoCs used to detect the attack in network traffic, even if the compromise passes unnoticed. For example, in a BYOD environment, enforcing security policies on the device can be difficult, so non-endpoint IoCs and solutions are needed to allow detection of compromise even with no endpoint coverage.

One example of how IoCs provide a layer of a defence-in-depth solution is Protective DNS (PDNS), a free and voluntary DNS filtering service provided by the UK NCSC for UK public sector organisations [PDNS]. In 2018, this service blocked access to 57.4 million DNS queries for 118,527 unique reasons (out of 68.7 billion total queries) for the organisations signed up to the service [ACD2019]. 28 million of them were for domain generation algorithms (DGAs) [DGAs], including 15 known DGAs which are a type of TTP.

IoCs such as malicious domains can be put on PDNS straight away and can then be used to prevent access to those known malicious domains across the entire estate of over 460 separate public sector entities that use NCSC's PDNS [Annual2019]. Coverage can be patchy with endpoints, as the roll-out of protections isn't uniform or necessarily fast - but if the IoC is on PDNS, a consistent defence is maintained. This offers protection, regardless of whether the context is a BYOD environment or a managed enterprise system. Other IoCs, like Server Name Indicator values in TLS or the server certificate information, also provide IoC protections.

Similar to the AV scenario, large scale services face risk decisions around balancing threat against business impact from false positives. Organisations need to be able to retain the ability to be more conservative with their own defences, while still benefiting from them. For instance, a commercial DNS filtering service is intended for broad deployment, so will have a risk tolerance similar to AV products; whereas DNS filtering intended for government users (e.g. PDNS) can be more conservative, but will still have a relatively

broad deployment if intended for the whole of government. A government department or specific company, on the other hand, might accept the risk of disruption and arrange firewalls or other network protection devices to completely block anything related to particular threats, regardless of the confidence, but rely on a DNS filtering service for everything else.

Other network defences can make use of this blanket coverage from IoCs, like middlebox mitigation, proxy defences, and application layer firewalls, but are out of scope for this draft. Note too that DNS goes through firewalls, proxies and possibly to a DNS filtering service; it doesn't have to be unencrypted, but these appliances must be able to decrypt it to do anything useful with it, like blocking queries for known bad URIs.

Covering a broad range of IoCs gives defenders a wide range of benefits: they are easy to deploy; they provide a high enough confidence to be effective; at least some will be painful for attackers to change; their distribution around the infrastructure allows for different points of failure, and so overall they enable the defenders to disrupt bad actors. The combination of these factors cements IoCs as a particularly valuable tool for defenders with limited resources.

6.2. Security Considerations

This draft is all about system security. However, when poorly deployed, IoCs can lead to over-blocking which may present an availability concern for some systems. While IoCs preserve privacy on a macro scale (by preventing data breaches), research could be done to investigate the impact on privacy from sharing IoCs, and improvements could be made to minimise any impact found. The creation of a privacy-preserving IoC sharing method, that still allows both network and endpoint defences to provide security and layered defences, would be an interesting proposal.

7. Conclusions

IoCs are versatile and powerful. IoCs underpin and enable multiple layers of the modern defence-in-depth strategy. IoCs are easy to share, providing a multiplier effect on attack defence effort and they save vital time. Network-level IoCs offer protection, especially valuable when an endpoint-only solution isn't sufficient. These properties, along with their ease of use, make IoCs a key component of any attack defence strategy and particularly valuable for defenders with limited resources.

For IoCs to be useful, they don't have to be unencrypted or visible in networks - but crucially they do need to be made available, along with their context, to entities that need them. It is also important that this availability and eventual usage copes with multiple points of failure, as per the defence-in-depth strategy, of which IoCs are a key part.

8. IANA Considerations

This draft does not require any IANA action.

9. Acknowledgements

Thanks to all those who have been involved with improving cyber defence in the IETF and IRTF communities.

10. Informative References

- [ACD2019] Levy, I. and M. S, "Active Cyber Defence - The Second Year", 2019, <<https://www.ncsc.gov.uk/report/active-cyber-defence-report-2019>>.
- [ADS] Microsoft, "File Streams (Local File Systems)", 2018, <<https://docs.microsoft.com/en-us/windows/win32/fileio/file-streams>>.
- [ALIENVAULT] AlienVault, "AlienVault", 2021, <<https://otx.alienvault.com/>>.
- [Annual2019] NCSC, "Annual Review 2019", 2019, <https://www.ncsc.gov.uk/annual-review/2019/ncsc/docs/ncsc_2019-annual-review.pdf>.
- [COBALT] Cobalt Strike, "OVERRULED: Containing a Potentially Destructive Adversary", 2021, <<https://www.cobaltstrike.com/>>.
- [DFRONT] InfoSec Resources, "Domain Fronting", 2017, <<https://resources.infosecinstitute.com/topic/domain-fronting/>>.
- [DGAs] MITRE, "Dynamic Resolution: Domain Generation Algorithms", 2020, <<https://attack.mitre.org/techniques/T1483/>>.

- [EVOLVE] McFadden, M., "Evolution of Endpoint Security - An Operational Perspective", 2021, <<https://datatracker.ietf.org/doc/draft-mcfadden-opsec-endp-evolve/>>.
- [FireEye] O'Leary, J., Kimble, J., Vanderlee, K., and N. Fraser, "Insights into Iranian Cyber Espionage: APT33 Targets Aerospace and Energy Sectors and has Ties to Destructive Malware", 2017, <<https://www.fireeye.com/blog/threat-research/2017/09/apt33-insights-into-iranian-cyber-espionage.html>>.
- [FireEye2] FireEye, "OVERRULED: Containing a Potentially Destructive Adversary", 2018, <<https://www.fireeye.com/blog/threat-research/2018/12/overruled-containing-a-potentially-destructive-adversary.html>>.
- [GoldenTicket] Soria-Machado, M., Abolins, D., Boldea, C., and K. Socha, "Kerberos Golden Ticket Protection", 2014, <https://cert.europa.eu/static/WhitePapers/UPDATED - CERT-EU_Security_Whitepaper_2014-007_Kerberos_Golden_Ticket_Protection_v1_4.pdf>.
- [KillChain] Lockheed Martin, "The Cyber Kill Chain", 2020, <<https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html>>.
- [LAZARUS] Kaspersky Lab, "Lazarus Under The Hood", 2018, <https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/07180244/Lazarus_Under_The_Hood_PDF_final.pdf>.
- [Mimikatz] Mulder, J., "Mimikatz Overview, Defenses and Detection", 2016, <<https://www.sans.org/reading-room/whitepapers/detection/mimikatz-overview-defenses-detection-36780>>.
- [MISP] MISP, "MISP", 2019, <<https://www.misp-project.org/>>.
- [MISPCORE] MISP, "MISP Core", 2020, <<https://github.com/MISP/misp-rfc/blob/master/misp-core-format/raw.md.txt>>.
- [NCCGroup] Jansen, W., "Abusing cloud services to fly under the radar", 2021, <<https://research.nccgroup.com/2021/01/12/abusing-cloud-services-to-fly-under-the-radar/>>.

- [OPENIOC] Gibb, W., "OpenIOC: Back to the Basics", 2013,
<<https://www.fireeye.com/blog/threat-research/2013/10/openioc-basics.html>>.
- [OPENNIC] OpenNIC Project, "OpenNIC Project", 2021,
<<https://www.opennic.org/>>.
- [Owari] NCSC, "Owari botnet own-goal takeover", 2018,
<<https://www.ncsc.gov.uk/report/weekly-threat-report-8th-june-2018>>.
- [PDNS] NCSC, "Protective DNS", 2019,
<<https://www.ncsc.gov.uk/information/pdns>>.
- [PoP] Bianco, D.J., "The Pyramid of Pain", 2014,
<<https://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html>>.
- [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>>.
- [STIX] OASIS Cyber Threat Intelligence, "STIX", 2019,
<<https://oasis-open.github.io/cti-documentation/stix/intro>>.
- [Symantec] Symantec, "Elfin: Relentless", 2019,
<<https://www.symantec.com/blogs/threat-intelligence/elfin-apt33-espionage>>.
- [TAXII] OASIS Cyber Threat Intelligence, "TAXII", 2021,
<<https://oasis-open.github.io/cti-documentation/taxii/intro.html>>.
- [Timestomp] OASIS Cyber Threat Intelligence, "Timestomp", 2019,
<<https://attack.mitre.org/techniques/T1099/>>.
- [TLP] FIRST, "Traffic Light Protocol", 2021,
<<https://www.first.org/tlp/>>.

Authors' Addresses

Kirsty Paine
Splunk Inc.

Email: kirsty.ietf@gmail.com

Ollie Whitehouse
NCC Group

Email: ollie.whitehouse@nccgroup.com

James Sellwood
Twilio

Email: jsellwood@twilio.com

Andrew Shaw
UK National Cyber Security Centre

Email: andrew.s2@ncsc.gov.uk

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 5, 2020

R. Van Rein
ARPA2.net
March 4, 2020

HTTP Authentication with SASL
draft-vanrein-httpauth-sasl-04

Abstract

Most application-level protocols standardise their authentication exchanges under the SASL framework. HTTP has taken another course, and often ends up replicating the work to allow individual mechanisms. This specification adopts full SASL authentication into HTTP.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 5, 2020.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Embedding SASL in HTTP	3
2.1. HTTP Request and Response Messages	4
2.2. Authentication Field Definitions	5
2.3. Caching Authentication Results	6
3. Server-Side User Name	6
4. Authentication Session Example	7
5. Security Considerations	9
6. IANA Considerations	10
7. References	11
7.1. Normative References	11
7.2. Informative References	12
Appendix A. HTTP Server Environment Variables	13
Appendix B. Acknowledgements	14
Author's Address	14

1. Introduction

HTTP has historically followed its own path for client authentication, while many other end-user protocols standardised on SASL; examples of SASL protocols include SMTP, IMAP, POP, XMPP, LDAP, AMQP and MQTT. This specification introduces SASL to HTTP, so it may share in past and future work done for SASL in general.

Among the work that could be shared is backend authentication integration, which is possible due to protocol-independent SASL exchanges for any given method, making it easy to take them out of one protocol and inserting them into another. Although HTTP has adopted several SASL-compatible authentication methods, it uses various notations and so it still needs method-specific support at the HTTP level to translate them to a SASL backend.

In front-ends, a similar situation has arisen. The varying syntaxes for authentication methods have made it difficult to rely on support in most or all HTTP clients. When such clients could externalise their SASL handling to generic software such as a SASL library, then any extension to a library automatically spills over into the HTTP sphere. It is common for developers of web clients to also produce email clients, so a shared code base (and credential store) is not difficult to imagine.

Sharing of authentication mechanisms is beneficial in both directions. HTTP benefits by being able to use anything from strong password mechanisms [RFC5802] without explicit support [RFC7804] in applications, up to GS2 mechanisms [RFC5801] with channel binding [RFC5056] [RFC5554] to TLS [RFC5929] based on pinning either the

certificate for the TLS server or even a unique part of the individual TLS connection; for instance Kerberos5 [RFC4120] currently uses Negotiate authentication [RFC4559] which is not as secure as GS2-KRB5-PLUS over SASL.

SASL also benefits; had it been the norm for HTTP, then the work to pass SAML over it [RFC6595] would probably have been done immediately. In fact, HTTP can still benefit from receiving standardised SAML20 inquiries over SASL, because it resolves the need for configuration of initiation paths and practices. Also, it removes authentication data from URIs, where they are not ideally placed.

In terms of security for HTTP applications, it appears beneficial to have very good authentication capabilities in the layers below the application; this is specifically true for applications developed in HTML and JavaScript, which tend to load code from various places, including code that is not always in the end user's interest; since it already is a concern what identity information passes through these applications, it is not advisable to use credentials in those places. The HTTP layer is in a better position to take control over these assets, at the protocol levels of HTTP and TLS, and conceal credentials and possibly also identity from applications running on top. Inasfar as tokens are needed, they can be derived from session keys using generally accepted key derivation schemes, but the session keys can be isolated from dynamic layers above HTTP.

2. Embedding SASL in HTTP

This specification integrates the SASL framework [RFC4422] into mainstream HTTP [RFC7231], [RFC7232]. The SASL Authentication scheme follows the general structure for HTTP Authentication [RFC7235]. It uses the WWW-Authenticate and Proxy-Authenticate headers in responses from web servers and web proxies, respectively, and correspondingly the Authorization and Proxy-Authorization request header to answer to requests.

The SASL service name for the following embedding of SASL is HTTP; contrary to most other service names, it is spelled in uppercase, in line with what has become general practice in Kerberos and GSSAPI.

Since SASL prescribes channel binding to occur relative to TLS instead of to the application protocol, we can add that when the HTTPS transport is used. Whether channel binding is used SHOULD remain a configuration choice in HTTP software, as it might interfere with intentional HTTPS proxying. Unintended proxying on the other hand, might lead to tapping of credentials under certain SASL mechanisms, and it may be considered helpful to prevent such

situations by relying on channel binding for at least those mechanisms.

2.1. HTTP Request and Response Messages

This section defines a few names for HTTP request and response messages, to be used in the remainder of this specification.

Initial Responses are HTTP responses that normally set a status code 401 or 407, and that are sent when the HTTP server decides to initiate an authentication exchange. In addition, the server MAY send Initial Responses in other responses, to indicate to the client that it MAY try again to achieve better results [Section 4.1 of [RFC7235]].

Initial Requests are those HTTP requests that a client sends to initiate a fresh SASL authentication. The identity SHOULD be selected by the user independently from the URI; prior settings MAY however be remembered by a client for the combination of resource authority (scheme, host and possibly a separately communicated resource user name) with the server-sent realm string. The server can support a mixture of client identities for various roles or access levels through variation of realm strings. There is no current practice of server-side resource names in HTTP, but the generic URI schema presents this logic and it is easy to imagine an HTTP User header that a client could support.

Intermediate Responses are HTTP responses to SASL authentication, with a status code set to 401 or 407. Intermediate Requests are those HTTP requests that a client sends to continue a SASL authentication after an Intermediate Response.

Positive Responses set a 200 status code to depict success. Information in this response is provided in an Authentication-Info or Proxy-Authentication-Info header [RFC7615] instead of the headers used in Initial Responses and Intermediate Responses [RFC7235]. Proper interpretation of a Positive Response requires client state indicating that SASL authentication was used, or else the optional fields are not completely reliable information sources; cryptographic markers in the c2c field MAY be used to overcome this in a manner that defies abuse by rogue servers.

Negative Responses also set a 401 or 407 status code and will often return the client to an earlier state that it recognises as one it has tried before. These responses should therefore offer authentication to start again. In contrast to the Initial Response, there is now a c2c field that helps the client evaluate the request.

The following fields, defined in upcoming sections, MUST and MAY be present in HTTP authentication exchanges for SASL:

Request or Response	MUST have fields	MAY have fields
Initial Response	s2s, mech	realm
Initial Request	c2c, s2s, mech	c2s, realm
Intermediate Response	c2c, s2s	s2c
Intermediate Request	c2c, s2s	c2s
Positive Response	c2c	s2s
Negative Response	c2c, s2s, mech	realm

2.2. Authentication Field Definitions

Data for SASL is transported in the following fields:

- c2s holds SASL token data from client to server. This field is transmitted with base64 encoding. The field is absent when the SASL client sends no token.
- s2c holds SASL token data from server to client. This field is transmitted with base64 encoding. The field is absent when the SASL server sends no token.
- s2s holds opaque server data which the client MUST reflect in Intermediate Requests. This is a necessity for a stateless HTTP Authentication framework [Section 5.1.2 of [RFC7235]]. It MAY be used in a Positive Response to pass a cacheable Section 2.3 authentication token.
- c2c holds opaque client data which the server MUST reflect in Intermediate, Positive and Negative Responses. This can help to also make the client stateless.

The following fields support SASL within the HTTP Authentication Framework:

- realm optionally names a scope of authorisation under the combination of scheme, server host name and possibly a HTTP user to implement the semantics of the generic URI username for resource selection. The realm does not necessarily match a domain name, which is used elsewhere as a realm notation.
- mech In an Initial Response, the field is filled with a space-separated list of SASL mechanism names; In an Initial Request, the client chooses one SASL mechanism name.

2.3. Caching Authentication Results

When an HTTP server sends a Positive Response, it MAY include an "s2s" field. If it does this, then it should be prepared to accept the field value for authentication in an Initial Request. However, credentials can expire or fall in disgrace for other reasons, so the server MAY still choose to reject the provided field.

When an HTTP client receives a Positive Response with an "s2s" field, it MAY memorise the fields for future reuse in an Initial Request, either with or without preceding Initial Response from the server. The HTTP client MUST use the realm as part of the decision which cached result to use, but it MAY extrapolate the results from one resource retrieval in an attempt to authenticate another.

When cached fields result in a Negative Response then the HTTP client SHOULD remove the failing cache entry, and it SHOULD try again by going through a full SASL authentication cycle. The stateless nature of HTTP authentication is helpful in the sense that a new Initial Request can be sent to an older Initial Response.

3. Server-Side User Name

HTTP does not define a mechanism to specifically select the user as an authoritative resource name space on the server. Local syntax conventions exist, but lack universally reliable semantics. Basic authentication has been used to this effect, but this conflates the client identity with the server-side name space, which is not necessarily the same.

To allow HTTP servers to zoom in on user-specific information, the User header is hereby introduced. Its syntax matches the userinfo part of a URI, up to but excluding any colons in it:

```
User = *( unreserved / pct-encoded / sub-delims )
```

The value of the header MUST be percent-decoded before the server can use it to identify a local user.

The User header MAY be sent by clients, and HTTP servers MAY ignore it for any reason, including local user identities that do not comply to a more restrictive local user name syntax.

When an HTTP server makes use of the User header, it MUST include a Vary header in its response, with either a single "*" in it or the name "User". This informs caches that the response must be considered specific to the User header value in the matching request.

The User header may be used with or without any form of authentication. When used with authentication, the value of the percent-decoded header is considered part of the authority component of the resource, and therefore of the naming scope for the realm. Clients can use this refined notion of realm to select an authentication identity; when the value is known early enough, this may even help to select an X.509 client certificate. Note that the User header might be used together with the aforementioned practice of Basic authentication, but it can also replace it with an even simpler mechanism to free up the authentication exchange for HTTP SASL.

The distinction of a client-side user from a server-side user can benefit the use of credential schemes that are not tied to the HTTP server. A specific example of this is the current work on realm crossover with GS2-SXOVER-PLUS. The use of such a mechanism may offload security concerns from the application layer.

4. Authentication Session Example

This section is non-normative.

When an HTTP server receives a request for a protected page, it will send an Initial Response to ask for authentication with a special status code 401; for proxy access that would be 407, and header names change accordingly. Stripped down to the bare essentials, the server sends (this section adds whitespace for clarity)

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: SASL
    realm="members only"
    mech="SCRAM-SHA-256 SCRAM-SHA-256-PLUS
        SCRAM-SHA-1 SCRAM-SHA-1-PLUS
        GS2-KRB5-PLUS GS2-KRB5",
    s2s=[xxxxxx]
```

The server offers SCRAM-* and GS2-KRB5 mechanisms. The variants with -PLUS provide additional channel binding, to ensure that authentication is specific to the current HTTPS connection, thus avoiding replay of the session across connections. Clients aware of HTTP connections may use connection-specific channel binding (tls-unique) while those that abstract from the connections must resort to weaker name-based channel binding (tls-server-end-point).

The server might have additionally offered the ANONYMOUS mechanism to allow the client to select "guest mode" access; the interaction would continue as authenticated, but presumably with limited access to HTTP resources and continued WWW-Authenticate headers to continue to offer

authentication to improve resource information content. The server might have offered EXTERNAL to allow the client to incorporate a TLS credential for authentication and possibly change to an authorization identity. The server might have offered GS2-SXOVER-PLUS if it is willing to connect to the client's home realm over Diameter, and thereby support realm crossover of SASL credentials.

The client initiates the SCRAM-SHA-256-PLUS mechanism, and to that end sends an Initial Request (this section shows square brackets instead of base64-encoding)

Authorization: SASL

```
realm="members only"
mech="SCRAM-SHA-256-PLUS",
c2s=[n,,n=user,r=rOprNGfwEbeRWgbNEkqO],
s2s=[xxxxx],
c2c=[qqqqq]
```

This mechanism is initiated by the client, hence the inclusion of the c2s token in the Initial Request. The contents of this field are specific to the selected mechanism, so SCRAM-SHA-256-PLUS in this case.

The SCRAM mechanism implementation is now initiated with the c2s token, and the server produces a followup challenge in a s2c token. To be able to validate future client messages against server-side state, it includes such state in an s2s token. This token is presumably protected from abuse with a signature and/or encryption, and it would likely identify the selected mechanism to validate during later rounds. The server packs all this in an Intermediate Response

HTTP/1.1 401 Unauthorized

WWW-Authentication: SASL

```
s2c=[r=rOprNGfwEbeRWgbNEkqO%hvYDpWUa2RaTCAfuxF
    Ilj)hNlF$k0,s=W22ZaJ0SNY7soEsUEjb6gQ==,
    i=4096]
s2s=[yyyyy],
c2c=[qqqqq]
```

Given that all server state is contained in this message, the client is free at any time to give up authentication and perhaps try another method. Normally however, it would proceed with the ongoing transaction. The client bounces state through the server in the c2c token, though it could be empty if a client manages state locally. Complex services however, would prefer the added signing and/or encryption of c2c in return for the benefit of decoupling the request/response state from the network connection.

The SCRAM mechanism continues with another round. The client engages in the prescribed cryptographic computations and packs an Intermediate Request along with updated state in the new c2c token

Authorization: SASL

```
c2s=[c=biws,r=rOprNGfwEbeRWgbNEkq0%hvYDpWUa2RaTCAfuxFIlj)hN
    1F$k0,p=dHzbZapWIk4jUhN+Ute9ytag9zjfMHgsqmmiz7AndVQ=]
s2s=[yyyyy],
c2c=[rrrrrr]
```

When the client has performed authentication properly, as determined by a server-side check of the c2s response token with the prior state in the s2s token, it can send a Positive Response along with the requested resource

HTTP/1.1 200 OK

WWW-Authentication: SASL

```
s2c=[v=6rrriTRBi23WpRR/wtup+mMhUZUn/dB5nLTJRsjl95G4=]
s2s=[zzzzzz],
c2c=[rrrrrr]
```

The s2s token in a Positive Response is an optional extension. It is presented by the server to allow the client to speed up authentication in future requests. The client may send it whenever the server asks for the same realm string under the same scheme and authority; the client may make proactive assumptions about the realm string for new requests. Authentication must never be reused in another context than bound by channel binding. When used, the client immediately sends an Intermediate Response holding

Authorization: SASL

```
realm="members only"
s2s=[zzzzzz],
c2c=[ssssss]
```

The server always has an option to refuse repeated authentication and forcing the client into a new authentication round. One reason for this could be that a session timed out. Another might be that the client is trying to use a credential outside a scope set by channel binding.

5. Security Considerations

It is not generally safe for SASL mechanisms to exchange c2s and s2c messages over unprotected transports. Furthermore, the SASL exchange may be at risk of tampering when the sequence of HTTP messages is not secured to form one stream. This means that a secure transport layer

must be used, like TLS. The termination of such a secure layer MUST also terminate any ongoing SASL handshakes.

The c2c and s2s fields MUST be protected against tampering by rogue peers, and such protection also protects against tampering by rogue intermediates when using an unprotected transport. In addition, but dependent on the mechanism used, the c2c and s2s fields may also need encryption to conceal their data from peers and intermediates.

SASL EXTERNAL can be a very efficient mechanism to combine with a secure transport layer if that includes authentication. This may be the case for TLS, especially when client-side authentication is deployed. Mechanisms other than EXTERNAL should take into account that a relation may exist between identities negotiated in the protective layer and the SASL exchange over HTTP. For example, a login account may be exchanged for an alias or group identity.

Channel binding is available in some SASL mechanisms. When used with HTTP SASL over TLS, it binds to the TLS channel, by default using the type tls-unique [Section 3 of [RFC5929]]. When doing so, it is vital that either there be no renegotiation of the TLS handshake, or both secure renegotiation [RFC5746] and the extended master secret [RFC7627] are used.

The User header field as defined herein is orthogonal to issues of authentication and authorisation, and adds no security concerns.

6. IANA Considerations

This specification extends the "Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry" with an "Authentication Scheme Name" SASL, referencing this specification.

This specification defines an additional entry in the registry "Generic Security Service Application Program Interface (GSSAPI)/Kerberos/Simple Authentication and Security Layer (SASL) Service Names" namely:

Service Name: HTTP

Usage: Web authentication using the SASL framework

Reference: TBD:this specification

The capitalisation of the service name has historic origins and is now the preferred spelling for reasons of compatibility.

Please add the following entry to the Message Headers registry:

Header Field Name	Template	Protocol	Status	Reference
-----	-----	-----	-----	-----
User		http	TBD	TBD:THIS_SPEC

7. References

7.1. Normative References

- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, DOI 10.17487/RFC4120, July 2005, <<https://www.rfc-editor.org/info/rfc4120>>.
- [RFC4422] Melnikov, A., Ed. and K. Zeilenga, Ed., "Simple Authentication and Security Layer (SASL)", RFC 4422, DOI 10.17487/RFC4422, June 2006, <<https://www.rfc-editor.org/info/rfc4422>>.
- [RFC4559] Jaganathan, K., Zhu, L., and J. Brezak, "SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows", RFC 4559, DOI 10.17487/RFC4559, June 2006, <<https://www.rfc-editor.org/info/rfc4559>>.
- [RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, DOI 10.17487/RFC5056, November 2007, <<https://www.rfc-editor.org/info/rfc5056>>.
- [RFC5554] Williams, N., "Clarifications and Extensions to the Generic Security Service Application Program Interface (GSS-API) for the Use of Channel Bindings", RFC 5554, DOI 10.17487/RFC5554, May 2009, <<https://www.rfc-editor.org/info/rfc5554>>.
- [RFC5746] Rescorla, E., Ray, M., Dispensa, S., and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension", RFC 5746, DOI 10.17487/RFC5746, February 2010, <<https://www.rfc-editor.org/info/rfc5746>>.
- [RFC5801] Josefsson, S. and N. Williams, "Using Generic Security Service Application Program Interface (GSS-API) Mechanisms in Simple Authentication and Security Layer (SASL): The GS2 Mechanism Family", RFC 5801, DOI 10.17487/RFC5801, July 2010, <<https://www.rfc-editor.org/info/rfc5801>>.

- [RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, DOI 10.17487/RFC5929, July 2010, <<https://www.rfc-editor.org/info/rfc5929>>.
- [RFC6595] Wierenga, K., Lear, E., and S. Josefsson, "A Simple Authentication and Security Layer (SASL) and GSS-API Mechanism for the Security Assertion Markup Language (SAML)", RFC 6595, DOI 10.17487/RFC6595, April 2012, <<https://www.rfc-editor.org/info/rfc6595>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.
- [RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <<https://www.rfc-editor.org/info/rfc7232>>.
- [RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <<https://www.rfc-editor.org/info/rfc7235>>.
- [RFC7615] Reschke, J., "HTTP Authentication-Info and Proxy-Authentication-Info Response Header Fields", RFC 7615, DOI 10.17487/RFC7615, September 2015, <<https://www.rfc-editor.org/info/rfc7615>>.
- [RFC7627] Bhargavan, K., Ed., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", RFC 7627, DOI 10.17487/RFC7627, September 2015, <<https://www.rfc-editor.org/info/rfc7627>>.

7.2. Informative References

- [I-D.vanrein-dnstxt-krbl] Rein, R., "Declaring Kerberos Realm Names in DNS (_kerberos TXT)", draft-vanrein-dnstxt-krbl-09 (work in progress), October 2016.
- [RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, DOI 10.17487/RFC2617, June 1999, <<https://www.rfc-editor.org/info/rfc2617>>.

- [RFC4505] Zeilenga, K., "Anonymous Simple Authentication and Security Layer (SASL) Mechanism", RFC 4505, DOI 10.17487/RFC4505, June 2006, <<https://www.rfc-editor.org/info/rfc4505>>.
- [RFC5802] Newman, C., Menon-Sen, A., Melnikov, A., and N. Williams, "Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms", RFC 5802, DOI 10.17487/RFC5802, July 2010, <<https://www.rfc-editor.org/info/rfc5802>>.
- [RFC7804] Melnikov, A., "Salted Challenge Response HTTP Authentication Mechanism", RFC 7804, DOI 10.17487/RFC7804, March 2016, <<https://www.rfc-editor.org/info/rfc7804>>.

Appendix A. HTTP Server Environment Variables

We define a number of variables that SHOULD be passed from an HTTP SASL stack (and from User header processing) to applications run on top of it. The intention of defining these is to obtain maximum interoperability between these layers of software.

The following variables MUST NOT be available until SASL authentication is successful; it would be available when the server could send a 200 OK response:

SASL_SECURE is only "yes" (without the quotes) when a client is authenticated to the current resource. It never has another value; it is simply undefined when not secured by SASL.

SASL_REALM is the realm for which the secure exchange succeeded. A realm is not always used, because sites only need it when there are more than one in the same name space. When undefined in the SASL flow, this variable will not be set.

REMOTE_USER is the client identity as confirmed through SASL authentication. Its content is formatted like an email address, and includes a domain name. That domain need not be related to the web server; it is possible for a web server to welcome foreign clients.

SASL_MECH indicates the mechanism used, and is one of the standardised SASL mechanism names. It may be used to detect the level of security.

SASL_S2S holds the accepted s2s field, and could be used as a random session identifier. It would normally be encrypted information.

SASL_S2S_ is a prefix for extra information that the server may extract from the s2s field in the HTTP SASL protocol flow. This depends on the authentication stack used in the web server.

The following variable SHOULD be available while processing a request with a User header with locally acceptable syntax:

LOCAL_USER gives the HTTP User header value after syntax checking and percent-decoding. If used at all, it MUST be treated as a resource name space selector. This header does not describe the authenticated client identity, which is usually passed in a variable REMOTE_USER.

Appendix B. Acknowledgements

Thanks to Henri Manson for making the first implementation of this specification and for feedback on the header formats. The specification also benefited from input by Daniel Stenberg.

Author's Address

Rick van Rein
ARPA2.net
Haarlebrink 5
Enschede, Overijssel 7544 WP
The Netherlands

Email: rick@openfortress.nl

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 7 August 2022

R. Van Rein
ARPA2.net
3 February 2022

HTTP Authentication with SASL
draft-vanrein-httpauth-sasl-06

Abstract

Most application-level protocols standardise their authentication exchanges under the SASL framework. HTTP has taken another course, and often ends up replicating the work to allow individual mechanisms. This specification adopts full SASL authentication into HTTP.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 7 August 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	2
2. Embedding SASL in HTTP	3
2.1. HTTP Request and Response Messages	4
2.2. Authentication Field Definitions	5
2.3. Caching Authentication Results	6
3. Server-Side User Name	6
4. Authentication Session Example	7
5. Security Considerations	9
6. IANA Considerations	10
7. References	10
7.1. Normative References	11
7.2. Informative References	12
Appendix A. HTTP Server Environment Variables	13
Appendix B. Acknowledgements	14
Author's Address	14

1. Introduction

HTTP has historically followed its own path for client authentication, while many other end-user protocols standardised on SASL; examples of SASL protocols include SMTP, IMAP, POP, XMPP, LDAP, AMQP and MQTT. This specification introduces SASL to HTTP, so it may share in past and future work done for SASL in general.

Among the work that could be shared is backend authentication integration, which is possible due to protocol-independent SASL exchanges for any given method, making it easy to take them out of one protocol and inserting them into another. Although HTTP has adopted several SASL-compatible authentication methods, it uses various notations and so it still needs method-specific support at the HTTP level to translate them to a SASL backend.

In front-ends, a similar situation has arisen. The varying syntaxes for authentication methods have made it difficult to rely on support in most or all HTTP clients. When such clients could externalise their SASL handling to generic software such as a SASL library, then any extension to a library automatically spills over into the HTTP sphere. It is common for developers of web clients to also produce email clients, so a shared code base (and credential store) is not difficult to imagine.

Sharing of authentication mechanisms is beneficial in both directions. HTTP benefits by being able to use anything from strong password mechanisms [RFC5802] without explicit support [RFC7804] in applications, up to GS2 mechanisms [RFC5801] with channel binding [RFC5056] [RFC5554] to TLS [RFC5929] based on pinning either the

certificate for the TLS server or even a unique part of the individual TLS connection; for instance Kerberos5 [RFC4120] currently uses Negotiate authentication [RFC4559] which is not as secure as GS2-KRB5-PLUS over SASL.

SASL also benefits; had it been the norm for HTTP, then the work to pass SAML over it [RFC6595] would probably have been done immediately. In fact, HTTP can still benefit from receiving standardised SAML20 inquiries over SASL, because it resolves the need for configuration of initiation paths and practices. Also, it removes authentication data from URIs, where they are not ideally placed.

In terms of security for HTTP applications, it appears beneficial to have very good authentication capabilities in the layers below the application; this is specifically true for applications developed in HTML and JavaScript, which tend to load code from various places, including code that is not always in the end user's interest; since it already is a concern what identity information passes through these applications, it is not advisable to use credentials in those places. The HTTP layer is in a better position to take control over these assets, at the protocol levels of HTTP and TLS, and conceal credentials and possibly also identity from applications running on top. Inasfar as tokens are needed, they can be derived from session keys using generally accepted key derivation schemes, but the session keys can be isolated from dynamic layers above HTTP.

2. Embedding SASL in HTTP

This specification integrates the SASL framework [RFC4422] into mainstream HTTP [RFC7231], [RFC7232]. The SASL Authentication scheme follows the general structure for HTTP Authentication [RFC7235]. It uses the WWW-Authenticate and Proxy-Authenticate headers in responses from web servers and web proxies, respectively, and correspondingly the Authorization and Proxy-Authorization request header to answer to requests.

The SASL service name for the following embedding of SASL is HTTP; contrary to most other service names, it is spelled in uppercase, in line with what has become general practice in Kerberos and GSSAPI.

Since SASL prescribes channel binding to occur relative to TLS instead of to the application protocol, we can add that when the HTTPS transport is used. Whether channel binding is used SHOULD remain a configuration choice in HTTP software, as it might interfere with intentional HTTPS proxying. Unintended proxying on the other hand, might lead to tapping of credentials under certain SASL mechanisms, and it may be considered helpful to prevent such

situations by requiring channel binding for those situations. HTTP in general allows a user session to hop between connections, and browsers are likely to do this; to support this, the support of tls-server-end-point channel binding [Section 4 of [RFC5929]] is RECOMMENDED. Specific HTTP clients may exercise more control over connections to achieve stronger security; for those use cases, tls-unique channel binding [Section 3 of [RFC5929]] is RECOMMENDED. Generic web servers SHOULD support both forms of channel binding.

2.1. HTTP Request and Response Messages

This section defines a few names for HTTP request and response messages, to be used in the remainder of this specification.

Initial Responses are HTTP responses that normally set a status code 401 or 407, and that are sent when the HTTP server decides to initiate an authentication exchange. In addition, the server MAY send Initial Responses in other responses, to indicate to the client that it MAY try again to achieve better results [Section 4.1 of [RFC7235]].

Initial Requests are those HTTP requests that a client sends to initiate a fresh SASL authentication. The identity SHOULD be selected by the user independently from the URI; prior settings MAY however be remembered by a client for the combination of resource authority (scheme, host and possibly a separately communicated resource user name) with the server-sent realm string. The server can support a mixture of client identities for various roles or access levels through variation of realm strings. There is no current practice of server-side resource names in HTTP, but the generic URI schema presents this logic and it is easy to imagine an HTTP User header that a client could support.

Intermediate Responses are HTTP responses to SASL authentication, with a status code set to 401 or 407. Intermediate Requests are those HTTP requests that a client sends to continue a SASL authentication after an Intermediate Response.

Positive Responses set a 200 status code to depict success. Information in this response is provided in an Authentication-Info or Proxy-Authentication-Info header [RFC7615] instead of the headers used in Initial Responses and Intermediate Responses [RFC7235]. Proper interpretation of a Positive Response requires client state indicating that SASL authentication was used, or else the optional fields are not completely reliable information sources; cryptographic markers in the c2c field MAY be used to overcome this in a manner that defies abuse by rogue servers.

Negative Responses also set a 401 or 407 status code and will often return the client to an earlier state that it recognises as one it has tried before. These responses should therefore offer authentication to start again. In contrast to the Initial Response, there is now a c2c field that helps the client evaluate the request.

The following fields, defined in upcoming sections, MUST and MAY be present in HTTP authentication exchanges for SASL:

Request or Response	MUST have fields	MAY have fields
Initial Response	s2s, mech	realm
Initial Request	mech	c2s, realm, s2s
Intermediate Response	s2s	s2c
Intermediate Request	s2s	c2s
Positive Response		s2s
Negative Response	s2s, mech	realm

2.2. Authentication Field Definitions

Data for SASL is transported in the following fields:

- c2s holds SASL token data from client to server. This field is transmitted with base64 encoding. The field is absent when the SASL client sends no token.
- s2c holds SASL token data from server to client. This field is transmitted with base64 encoding. The field is absent when the SASL server sends no token.
- s2s holds opaque server data which the client MUST reflect in Intermediate Requests and, when responding to an Initial Response, in the Initial Request. This is a necessity for a stateless HTTP Authentication framework [Section 5.1.2 of [RFC7235]]. It MAY be used in a Positive Response to pass a cacheable Section 2.3 authentication token in a future Initial Request.

The following fields support SASL within the HTTP Authentication Framework:

realm optionally names a scope of authorisation under the combination of scheme, server host name and possibly a HTTP user to implement the semantics of the generic URI username for resource selection. The realm does not necessarily match a domain name, which is used elsewhere as a realm notation.

mech In an Initial Response, the field is filled with a space-

separated list of SASL mechanism names; In an Initial Request, the client chooses one SASL mechanism name.

2.3. Caching Authentication Results

When an HTTP server sends a Positive Response, it MAY include an "s2s" field. If it does this, then it should be prepared to accept the field value for authentication in an Initial Request. However, credentials can expire or fall in disgrace for other reasons, so the server MAY still choose to reject the provided field.

When an HTTP client receives a Positive Response with an "s2s" field, it MAY memorise the fields for future reuse in an Initial Request, either with or without preceding Initial Response from the server. The HTTP client MUST use the realm as part of the decision which cached result to use, but it MAY extrapolate the results from one resource retrieval in an attempt to authenticate another.

When cached fields result in a Negative Response then the HTTP client SHOULD remove the failing cache entry, and it SHOULD try again by going through a full SASL authentication cycle. The stateless nature of HTTP authentication is helpful in the sense that a new Initial Request can be sent to an older Initial Response.

3. Server-Side User Name

HTTP does not define a mechanism to specifically select the user as an authoritative resource name space on the server. Local syntax conventions exist, but lack universally reliable semantics. Basic authentication has been used to this effect, but this conflates the client identity with the server-side name space, which is not necessarily the same.

To allow HTTP servers to zoom in on user-specific information, the User header is hereby introduced. Its syntax matches the userinfo part of a URI, up to but excluding any colons in it:

User = *(unreserved / pct-encoded / sub-delims)

The value of the header MUST be percent-decoded before the server can use it to identify a local user.

The User header MAY be sent by clients, and HTTP servers MAY ignore it for any reason, including local user identities that do not comply to a more restrictive local user name syntax.

When an HTTP server makes use of the User header, it MUST include a Vary header in its response, with either a single "*" in it or the name "User". This informs caches that the response must be considered specific to the User header value in the matching request.

The User header may be used with or without any form of authentication. When used with authentication, the value of the percent-decoded header is considered part of the authority component of the resource, and therefore of the naming scope for the realm. Clients can use this refined notion of realm to select an authentication identity; when the value is known early enough, this may even help to select an X.509 client certificate. Note that the User header might be used together with the aforementioned practice of Basic authentication, but it can also replace it with an even simpler mechanism to free up the authentication exchange for HTTP SASL.

The distinction of a client-side user from a server-side user can benefit the use of credential schemes that are not tied to the HTTP server. A specific example of this is the current work on realm crossover with GS2-SXOVER-PLUS. The use of such a mechanism may offload security concerns from the application layer.

4. Authentication Session Example

This section is non-normative.

When an HTTP server receives a request for a protected page, it will send an Initial Response to ask for authentication with a special status code 401; for proxy access that would be 407, and header names change accordingly. Stripped down to the bare essentials, the server sends (this section adds whitespace for clarity)

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: SASL
    realm="members only"
    mech="SCRAM-SHA-256 SCRAM-SHA-256-PLUS
        SCRAM-SHA-1 SCRAM-SHA-1-PLUS
        GS2-KRB5-PLUS GS2-KRB5",
    s2s="[xxxxx]"
```

The server offers SCRAM-* and GS2-KRB5 mechanisms. The variants with -PLUS provide additional channel binding, to ensure that authentication is specific to the current HTTPS connection, thus avoiding replay of the session across connections. Clients aware of HTTP connections may use connection-specific channel binding (tls-unique) while those that abstract from the connections must resort to weaker name-based channel binding (tls-server-end-point).

The server might have additionally offered the ANONYMOUS mechanism to allow the client to select "guest mode" access; the interaction would continue as authenticated, but presumably with limited access to HTTP resources and continued WWW-Authenticate headers to continue to offer authentication to improve resource information content. The server might have offered EXTERNAL to allow the client to incorporate a TLS credential for authentication and possibly change to an authorization identity. The server might have offered GS2-SXOVER-PLUS if it is willing to connect to the client's home realm over Diameter, and thereby support realm crossover of SASL credentials.

The client initiates the SCRAM-SHA-256-PLUS mechanism, and to that end sends an Initial Request (this section shows square brackets around text that is transmitted with base64-encoding)

Authorization: SASL

```
realm="members only"
mech="SCRAM-SHA-256-PLUS",
c2s="[n,,n=user,r=rOprNGfwEbeRWgbNEkqO]",
s2s="[xxxxx]"
```

This mechanism is initiated by the client, hence the inclusion of the c2s token in the Initial Request. The contents of this field are specific to the selected mechanism, so SCRAM-SHA-256-PLUS in this case.

The SCRAM mechanism implementation is now initiated with the c2s token, and the server produces a followup challenge in a s2c token. To be able to validate future client messages against server-side state, it includes such state in an s2s token. This token is presumably protected from abuse with a signature and/or encryption, and it would likely identify the selected mechanism to validate during later rounds. The server packs all this in an Intermediate Response

HTTP/1.1 401 Unauthorized

WWW-Authentication: SASL

```
s2c="[r=rOprNGfwEbeRWgbNEkqO%hvYDpWUa2RaTCAfuxF
I1j)hN1F$K0,s=W22ZaJ0SNY7soEsUEjb6gQ==,
i=4096]"
s2s="[yyyyy]"
```

Given that all server state is contained in this message, the client is free at any time to give up authentication and perhaps try another method. Normally however, it would proceed with the ongoing transaction.

The SCRAM mechanism continues with another round. The client engages in the prescribed cryptographic computations and packs an Intermediate Request along with updated state in the new c2s token

Authorization: SASL

```
c2s="[c=biws,r=rOprNGfwEbeRWgbNEkqO%hvYDpWUa2RaTCAfuxFILj)hN
    1F$k0,p=dHzbZapWIk4jUhN+Ute9ytag9zjfMHgsqmmiz7AndVQ=]"
s2s="[yyyyy]"
```

When the client has performed authentication properly, as determined by a server-side check of the c2s response token with the prior state in the s2s token, it can send a Positive Response along with the requested resource

HTTP/1.1 200 OK

WWW-Authentication: SASL

```
s2c="[v=6rrriTRBi23WpRR/wtup+mMhUZUn/dB5nLTJRsjl95G4=]"
s2s="[zzzzz]"
```

The s2s token in a Positive Response is an optional extension. It is presented by the server to allow the client to speed up authentication in future requests. The client may send it whenever the server asks for the same realm string under the same scheme and authority; the client may make proactive assumptions about the realm string for new requests. Authentication must never be reused in another context than bound by channel binding. When used, the client immediately sends an Intermediate Response holding

Authorization: SASL

```
realm="members only"
s2s="[zzzzz]"
```

The server always has an option to refuse repeated authentication and forcing the client into a new authentication round. One reason for this could be that a session timed out. Another might be that the client is trying to use a credential outside a scope set by channel binding.

5. Security Considerations

It is not generally safe for SASL mechanisms to exchange c2s and s2c messages over unprotected transports. Furthermore, the SASL exchange may be at risk of tampering when the sequence of HTTP messages is not secured to form one stream. This means that a secure transport layer must be used, like TLS. The termination of such a secure layer MUST also terminate any ongoing SASL handshakes.

The s2s field MUST be protected against tampering by rogue peers, and such protection also protects against tampering by rogue intermediates when using an unprotected transport. In addition, but dependent on the mechanism used, the s2s field may also need encryption to conceal their data from peers and intermediates.

SASL EXTERNAL can be a very efficient mechanism to combine with a secure transport layer if that includes authentication. This may be the case for TLS, especially when client-side authentication is deployed. Mechanisms other than EXTERNAL should take into account that a relation may exist between identities negotiated in the protective layer and the SASL exchange over HTTP. For example, a login account may be exchanged for an alias or group identity.

Channel binding is available in some SASL mechanisms. When used with HTTP SASL over TLS, it binds to the TLS channel. When doing so, it is vital that either there be no renegotiation of the TLS handshake, or both secure renegotiation [RFC5746] and the extended master secret [RFC7627] are used.

The User header field as defined herein is orthogonal to issues of authentication and authorisation, and adds no security concerns.

6. IANA Considerations

This specification extends the "Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry" with an "Authentication Scheme Name" SASL, referencing this specification.

This specification defines an additional entry in the registry "Generic Security Service Application Program Interface (GSSAPI)/Kerberos/Simple Authentication and Security Layer (SASL) Service Names" namely:

Service Name: HTTP

Usage: Web authentication using the SASL framework

Reference: TBD:this specification

The capitalisation of the service name has historic origins and is now the preferred spelling for reasons of compatibility.

Please add the following entry to the Message Headers registry:

Header Field Name	Template	Protocol	Status	Reference
User		http	TBD	TBD:THIS_SPEC

7. References

7.1. Normative References

- [RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, DOI 10.17487/RFC4120, July 2005, <<https://www.rfc-editor.org/info/rfc4120>>.
- [RFC4559] Jaganathan, K., Zhu, L., and J. Brezak, "SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows", RFC 4559, DOI 10.17487/RFC4559, June 2006, <<https://www.rfc-editor.org/info/rfc4559>>.
- [RFC4422] Melnikov, A., Ed. and K. Zeilenga, Ed., "Simple Authentication and Security Layer (SASL)", RFC 4422, DOI 10.17487/RFC4422, June 2006, <<https://www.rfc-editor.org/info/rfc4422>>.
- [RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, DOI 10.17487/RFC5056, November 2007, <<https://www.rfc-editor.org/info/rfc5056>>.
- [RFC5554] Williams, N., "Clarifications and Extensions to the Generic Security Service Application Program Interface (GSS-API) for the Use of Channel Bindings", RFC 5554, DOI 10.17487/RFC5554, May 2009, <<https://www.rfc-editor.org/info/rfc5554>>.
- [RFC5746] Rescorla, E., Ray, M., Dispensa, S., and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension", RFC 5746, DOI 10.17487/RFC5746, February 2010, <<https://www.rfc-editor.org/info/rfc5746>>.
- [RFC5801] Josefsson, S. and N. Williams, "Using Generic Security Service Application Program Interface (GSS-API) Mechanisms in Simple Authentication and Security Layer (SASL): The GS2 Mechanism Family", RFC 5801, DOI 10.17487/RFC5801, July 2010, <<https://www.rfc-editor.org/info/rfc5801>>.
- [RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, DOI 10.17487/RFC5929, July 2010, <<https://www.rfc-editor.org/info/rfc5929>>.
- [RFC6595] Wierenga, K., Lear, E., and S. Josefsson, "A Simple Authentication and Security Layer (SASL) and GSS-API Mechanism for the Security Assertion Markup Language (SAML)", RFC 6595, DOI 10.17487/RFC6595, April 2012, <<https://www.rfc-editor.org/info/rfc6595>>.

- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.
- [RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <<https://www.rfc-editor.org/info/rfc7232>>.
- [RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <<https://www.rfc-editor.org/info/rfc7235>>.
- [RFC7615] Reschke, J., "HTTP Authentication-Info and Proxy-Authentication-Info Response Header Fields", RFC 7615, DOI 10.17487/RFC7615, September 2015, <<https://www.rfc-editor.org/info/rfc7615>>.
- [RFC7627] Bhargavan, K., Ed., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", RFC 7627, DOI 10.17487/RFC7627, September 2015, <<https://www.rfc-editor.org/info/rfc7627>>.

7.2. Informative References

- [RFC5802] Newman, C., Menon-Sen, A., Melnikov, A., and N. Williams, "Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms", RFC 5802, DOI 10.17487/RFC5802, July 2010, <<https://www.rfc-editor.org/info/rfc5802>>.
- [RFC7804] Melnikov, A., "Salted Challenge Response HTTP Authentication Mechanism", RFC 7804, DOI 10.17487/RFC7804, March 2016, <<https://www.rfc-editor.org/info/rfc7804>>.
- [I-D.vanrein-dnstxt-krbl]
Rein, R., "Declaring Kerberos Realm Names in DNS (_kerberos TXT)", Work in Progress, Internet-Draft, draft-vanrein-dnstxt-krbl-09, 24 October 2016, <<http://www.ietf.org/internet-drafts/draft-vanrein-dnstxt-krbl-09.txt>>.

Appendix A. HTTP Server Environment Variables

We define a number of variables that SHOULD be passed from an HTTP SASL stack (and from User header processing) to applications run on top of it. The intention of defining these is to obtain maximum interoperability between these layers of software.

The following variables MUST NOT be available until SASL authentication is successful; it would be available when the server could send a 200 OK response:

`SASL_SECURE` is only "yes" (without the quotes) when a client is authenticated to the current resource. It never has another value; it is simply undefined when not secured by SASL.

`SASL_REALM` is the realm for which the secure exchange succeeded. A realm is not always used, because sites only need it when there are more than one in the same name space. When undefined in the SASL flow, this variable will not be set.

`REMOTE_USER` is the client identity as confirmed through SASL authentication. Its content is formatted like an email address, and includes a domain name. That domain need not be related to the web server; it is possible for a web server to welcome foreign clients.

`SASL_MECH` indicates the mechanism used, and is one of the standardised SASL mechanism names. It may be used to detect the level of security.

`SASL_S2S` holds the accepted s2s field, and could be used as a random session identifier. It would normally be encrypted information.

`SASL_S2S_` is a prefix for extra information that the server may extract from the s2s field in the HTTP SASL protocol flow. This depends on the authentication stack used in the web server.

The following variable SHOULD be available while processing a request with a User header with locally acceptable syntax:

`LOCAL_USER` gives the HTTP User header value after syntax checking and percent-decoding. If used at all, it MUST be treated as a resource name space selector. This header does not describe the authenticated client identity, which is usually passed in a variable `REMOTE_USER`.

Appendix B. Acknowledgements

Thanks to Henri Manson for making the first implementation of this specification and for feedback on the header formats. The specification also benefited from input by Daniel Stenberg.

This work was supported with an open source development fund from NLNet.

Author's Address

Rick van Rein
ARPA2.net
Haarlebrink 5
Enschede

Email: rick@openfortress.nl