OTP Extended Responses

1.0 ABSTRACT

This document provides a specification for a type of response to an
OTP [RFC 1938] challenge that carries explicit indication of the
response’s encoding. Codings for the two mandatory OTP data formats
using this new type of response are presented. This document also
provides a specification for a response that allows OTP generator to
request that a server re-initialize a sequence and change parameters
such as the secret pass phrase.

2.0 CONVENTIONS, TERMS, and NOTATION

This document specifies the data formats and software behaviors
needed to use OTP extended responses. The data formats are described
three ways: using an ad-hoc UNIX manual page style syntax, using
augmented BNF described in sections two and three of RFC
822[RFC822], and by examples. Should there be any conflict between
these descriptions, the augmented BNF takes precedence. The software
behaviors are described in words, and specific behavior compliance
requirements are itemized using the requirements terminology
described in section four of RFC 1938.
3.0 EXTENDED RESPONSES

This document builds on the protocol and terminology specified in RFC 1938 and assumes that you have already read this document and understand its contents.

An extended response is a single line of printable text terminated by a new line sequence appropriate for the context of its use (e.g., ASCII CR followed by ASCII LF). It contains two or more tokens that are separated with a single colon (':') character. The first token contains a type specifier that indicates the format of the rest of the response. The tokens that follow are argument data for the OTP extended response. At least one token of data MUST be present.

Syntax

In UNIX manual page like syntax, the general form of an extended response could be described as:

    <type-specifier>[:<arg1>[[:<arg2>[:...]]]

In augmented BNF syntax, the syntax of the general form of an extended response is:

    extended-response = type 1*(":" argument) newline
    type = token
    argument = token
    token = 1*<any CHAR except ":" and CTLs>
    newline = CRLF / CR / LF / <other appropriate new line sequence>

An example of the extended response using a mythical type named "foo" is:

    foo:some data:some more data:12345

Requirements

A server compliant with this specification:

1. MUST be able to receive and parse the general form of an extended response
2. MUST process the type field in a case-insensitive manner
3. MUST reject any authentication attempt using an extended response if it does not support that type of response
4. SHOULD provide an appropriate indication to the generator if the response was rejected because of (3)
5. MUST limit the length of the input reasonably
6. MUST accept otherwise arbitrary amounts of whitespace wherever a response allows it
7. MUST be able to receive and correctly process standard OTP responses
A generator compliant with this specification:

1. SHOULD have an option that selects whether standard or extended responses are generated
2. SHOULD make (1) easily available to the end user
3. SHOULD be configurable on a per-server or per-seed basis
4. MUST be able to generate standard OTP responses
5. SHOULD initially default to using standard responses
6. MUST generate the type field in lower case

4.0 THE "HEX" AND "WORD" RESPONSES

There exists a very rare case in which a standard OTP response could be a valid coding in both the hexadecimal and six-word formats. An example of this is the response "ABE ACE ADA ADD BAD A." The solution to this problem mandated by the OTP specification is that compliant servers MUST attempt to parse and verify a standard response in both hexadecimal and six-word formats and must consider the authentication successful if either succeeds.

This problem can be solved easily using extended responses. The "hex" response and the "word" response are two response types that encode an OTP in an extended response that explicitly describes the encoding. These responses start with a type label of "hex" for a hexadecimal OTP and "word" for a six-word coded OTP. These responses contain one argument field that contains a standard OTP response coded in the indicated format.

Syntax

In UNIX manual page like syntax, the format of these responses could be described as:

hex:<hexadecimal number>
word:<six dictionary words>

In augmented BNF syntax and with the definitions already provided, the syntax of these responses is:

hex-response = "hex:" hex-64bit newline
hex-64bit    = 16(hex-char *LWS-char)
hex-char     = ("A" / "B" / "C" / "D" / "E" / "F" /
                "a" / "b" / "c" / "d" / "e" / "f" /
                "0" / "1" / "2" / "3" / "4" / "5" /
                "6" / "7" / "8" / "9")

word-response = "word:" word-64bit newline
word-64bit    = 6(otp-word 1*LWS-char)
otp-word      = <any valid word in the standard OTP coding dictionary>
Examples of these responses are:

```
hex:8720 33d4 6202 9172      word:VAST SAUL TAKE SODA SUCH BOLT
```

Requirements

A server compliant with this specification:

1. MUST treat all arguments in a case-insensitive manner

A generator compliant with this specification:

1. MUST generate otp-word tokens in upper case with single spaces separating them
2. MUST generate hexadecimal numbers using only lower case for letters

5.0 THE "INIT" AND "INIT-WORD" RESPONSES

The OTP specification requires that implementations provide a means for a client to re-initialize or change its OTP information with a server but does not require any specific protocol for doing it. Implementations that support the OTP extended responses described in this document MUST support the response with the "init" and "init-word" type specifiers, which provide a standard way for a client to re-initialize its OTP information with a server. This response is intended to be used only by automated clients. Because of this, the primary form of this response uses the hexadecimal encoding for binary data. It is possible for a user to type in an "init" or "init-word" response. However, there is enough data that would need to be typed that the six-word coding does not make typing this response much easier.

Syntax

In UNIX manual page like syntax, the format of these responses could be described as:

```
init:<old-OTP>[:<new-params>]:<new-OTP>[:<check-update>:<check-value>]
init-word:<old-OTP>[:<new-params>]:<new-OTP>[:<check-update>:<check-value>]
```

In augmented BNF syntax and with the definitions already provided, the syntax of the "init" response is:

```
init-response   = "init:" old-OTP ":" new-params "::" new-OTP
                 1*0("::" check-update ":" check-value) newline
old-OTP         = hex-64bit
new-OTP         = hex-64bit
```
check-update = hex-64bit
check-value = hex-64bit

new-params = algorithm SPACE sequence-number SPACE seed
algorithm = "md4" / "md5" / "sha1"
sequence-number = 4*3DIGIT
seed = 16*1(ALPHA / DIGIT)

In augmented BNF syntax and with the definitions already provided, the syntax of the "init-word" response is:

init-word-response = "init-word:" old-OTP ":" new-params ":" new-OTP
1*0(":" check-update ":" check-value) newline

old-OTP = word-64bit
new-OTP = word-64bit
check-update = word-64bit
check-value = word-64bit

new-params = algorithm SPACE sequence-number SPACE seed
algorithm = "md4" / "md5" / "sha1"
sequence-number = 4*3DIGIT
seed = 16*1(ALPHA / DIGIT)

Note that all appropriate fields for the "init" response MUST be hexadecimally coded and that all appropriate fields for the "init-word" response MUST be six-word coded.

Examples of these responses are:

init:f6bd 6b33 89b8 7203:md5 499 ke6118:23d1 b253 5ae0 2b7e
init:c9b2 12bb 6425 5a0f:md5 499 ke0986:fd17 cef1 b4df 093e:
6ele faa6 b7d1 a43f:4c72 33b7 101a 7e62

init-word:MOOD SOFT POP COMB BOLO LIFE:md5 499 ke1235:
ARTY WEAR TAD RUG HALO GIVE
init-word:END KERN BALM NICK EROS WAVY:md5 499 ke1235:
BABY FAIN OILY NIL TIDY DADE:DIG DIVE SUNG HORN SWAG GAP:
GUT RODE CAKE ROY DATA GOER

(Note that all of these responses are one line. Due to their length, they had to be split into multiple lines in order to be included here. These responses MUST NOT span more than one line in actual use)

Description of Fields

The old-OTP field contains a hexadecimally coded response to the OTP challenge. The new-params field contains the parameters for the client’s new requested challenge and the new-OTP field contains a hexadecimally coded response to that challenge. If the re-
initialization is successful, a server MUST store the new OTP in its
database as the last successful OTP received and the sequence number
in the next challenge presented by the server MUST be one less than
the sequence number specified in the new-params field.

The check-update and check-value fields provide a simple defense
against active attacks. This is not intended to provide a high level
of security. It is intended to make it significantly more difficult
for an active attacker to interfere with the OTP re-initialization
process and thus set the OTP sequence and/or secret pass phrase to a
value chosen by the attacker. The check fields are optional. They
require that a server store data that is secret for purposes of the
re-initialization. This is unacceptable in some circumstances. Also,
some sites may already employ other security solutions that already
eliminate active attacks (e.g., IP Security [RFC1825]).

These fields use a "check key" that is derived from a seed and the
secret pass phrase. The OTP system prepares them for iteration by
concatenating them and running them through a hash function. The
check key is generated the same way, except that the order of the
secret pass phrase and seed is reversed. If the secret pass phrase
and seed are the same, the hash result could be used to generate OTP
responses. Therefore, servers and generators MUST specifically
disallow the secret pass phrase to be the same as the seed. For re-
initialization methods that only supply the server with an OTP
result, this can be accomplished by generating an OTP using the same
sequence number and seed as the generator used and setting the
secret pass phrase equal to the seed. If the resulting OTP is the
same as the OTP supplied by the generator, then the server MUST
reject it. For re-initialization on the console or with another
secure channel that allows the user to supply the server with the
secret pass phrase as cleartext, this check can be made by a simple
string comparison.

The check-update field contains the exclusive-OR (XOR) of the check
key generated from the current seed and secret pass phrase with the
check key generated from the new seed and secret pass phrase. This
allows any party that knows the current check key -- which is
considered secret data -- to derive the new check key.

The check-value field contains the result of a standard envelope
message authentication code using the OTP folded hash function.
This is the result of running the folded hash function over the
concatenation of:

1. current check key
2. current-OTP
3. new-params (string)
4. new-OTP
5. check-update
6. current check key
The new-params field is hashed as a string the same way that a seed or secret pass phrase would be. All other field values are hashed in their uncoded binary forms, in network byte order and without any padding.

Requirements

A server compliant with this specification:

1. MAY refuse to accept any init responses that do not carry the active attack protection values
2. MAY refuse to accept any init responses for a user if it does not have a check value in its database
3. MUST remove the init response check value from its database if the secret pass phrase or seed change using any mechanism that does not implement check values exactly the same way as the init response
4. MUST NOT allow a user to use the same value for their seed and secret pass phrase
5. MUST make the above check for all (re-)initialization methods they support
6. MUST disable all OTP access to any principal whose sequence number would be less than one
7. MUST handle the case where the old-OTP is valid but the active attack protection check fails by acting as if the old-OTP were provided as a standard response. That is, the next challenge will contain the current seed and one less than the current sequence number.
8. SHOULD allow a user to use the simple active attack protection provided by the check-update and check-value fields
9. SHOULD require that this protection be used if it has a check value in its database for the current secret pass phrase and seed
10. MUST perform the verification steps for the active attack protection if it has a check value in its database for the old secret pass phrase and seed and active attack protection information is provided in the response
11. MUST NOT re-initialize the sequence if (10) fails, even if the use of active attack protection is not mandatory

A generator compliant with this specification:

1. MUST NOT allow a user to use the same value for their seed and secret pass phrase
2. MUST take specific steps to prevent infinite loops of re-initialization attempts in case of failure
3. SHOULD provide the user with some indication that the re-initialization is taking place
4. SHOULD support the simple active attack protection
5. SHOULD NOT do a re-initialization without the user’s permission, either for that specific instance or as a configuration option
6. SHOULD NOT retry a failed re-initialization without a user’s permission
7. MUST refuse to generate OTPs with a sequence number below one

6.0 SECURITY CONSIDERATIONS

All of the security considerations for the OTP system also apply to the OTP system with extended responses.

The re-initialization response provides a means of protection against active attacks. It is not meant to defeat well-funded and well-skilled adversaries. This protection is not a substitute for stronger measures such as IP Security.

The active attack protection requires that shared secret information be stored on the server. It is the responsibility of the server to keep that information secret. Disclosure of the secret data reduces the security of the re-initialization response with active attack protection to that of the response without it (the user’s secret pass phrase is not compromised, but an attacker could substitute a different secret pass phrase).

7.0 ACKNOWLEDGEMENTS

Like rfc 1938, the protocol described in this document was created by contributors in the IETF OTP working group. Specific contributions were made by Neil Haller, who provided input on the overall design requirements of a re-initialization protocol, Denis Pinkas, who suggested an active attack defense mechanism that is used here with minor modifications, and Phil Servita, who opened the debate with the first real protocol proposal and provided lots of specific input on the design of this and earlier protocols.

Randall Atkinson and Ted T’so also contributed their views to discussions about details of the protocol extensions in this document.

8.0 REFERENCES


9.0 Author’s Address

Craig Metz
Kaman Sciences Corporation
For NRL Code 5544
4555 Overlook Avenue, S.W.
Washington, DC, 20375-5337, USA

Email: cmetz@itd.nrl.navy.mil
Appendix - Reference Responses

The following responses were generated by the One-Time Passwords in Everything (OPIE) 2.3 implementation of the extended responses described here.

All of these are responses to the challenge:

```plaintext
otp-md5 499 ke1234
```

Note that the re-initialization responses use the same secret pass phrase for new and current and a new seed of "ke1235". Also, these responses have been split for formatting purposes into multiple lines; they MUST NOT be multiple lines in actual use.

The secret pass phrase for these responses is:

```
This is a test.
```

The OTP standard hexadecimal response is:

```
5bf0 75d9 959d 036d
```

The OTP standard six-word response is:

```
BOND FOGY DRAB NE RISE MART
```

The OTP extended "hex" response is:

```
hex:5bf0 75d9 959d 036f
```

The OTP extended "word" response is:

```
word:BOND FOGY DRAB NE RISE MART
```

The OTP extended "init" response without active attack protection is:

```
init:5BF0 75D9 959D 036F:md5 499 ke1235:3712 DCB4 AA53 16C1
```

The OTP extended "init-word" response without active attack protection is:

```
init-word:BOND FOGY DRAB NE RISE MART:md5 499 ke1235:
RED HERD NOW BEAN PA BURG
```

The OTP extended "init" response with active attack protection is:

```
init:5BF0 75D9 959D 036F:md5 499 ke1235:3712 DCB4 AA53 16C1:
6E00 18AF 5582 73DE:E69E 1812 2A85 9DB6
```

C. Metz
The OTP extended "init-word" response with active attack protection is:

init-word:BOND FOGY DRAB NE RISE MART:md5 499 ke1235:
RED HERD NOW BEAN PA BURG:CURL ADD NIB LATE MAO TRAY:
SWUM TUFT AVE BEAU BLUE SICK