Abstract

Client puzzles allow a TLS server to defend itself against asymmetric DDoS attacks. In particular, it allows a server to request clients perform a selected amount of computation prior to the server performing expensive cryptographic operations. This allows servers to employ a layered defense that represents an improvement over pure rate-limiting strategies.

Client puzzles are implemented as an extension to TLS 1.3 [I-D.ietf-tls-tls13] wherein a server can issue a HelloRetryRequest containing the puzzle as an extension. The client must then resend its ClientHello with the puzzle results in the extension.

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 http://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 January 3, 2016.

Copyright Notice

Copyright (c) 2015 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 (http://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. Overview and rationale ...................................... 2
2. Notational Conventions .................................... 3
3. Handshake Changes ........................................ 3
3.1. The ClientPuzzleExtension Message ..................... 5
4. Usage by Servers .......................................... 6
5. Proposed Client Puzzles ................................... 6
5.1. Cookie Client Puzzle Type ................................. 6
5.2. SHA-256 CPU Reverse Puzzle Type ......................... 6
5.3. SHA-512 CPU Reverse Puzzle Type ......................... 8
5.4. SHA-256 Memory Reverse Puzzle Type ...................... 8
6. IANA Considerations ....................................... 10
7. Security Considerations .................................... 10
8. Privacy Considerations ..................................... 11
9. Acknowledgments ........................................... 11
10. References ................................................ 11
10.1. Normative References .................................. 11
10.2. Informative References ................................ 12

1. Overview and rationale

Adversaries can exploit the design of the TLS protocol to craft powerful asymmetric DDOS attacks. Once an attacker has opened a TCP connection, the attacker can transmit effectively static content that causes the server to perform expensive cryptographic operations. Rate limiting offers one possible defense against this type of attack; however, pure rate limiting systems represent an incomplete solution:

1. Rate limiting systems work best when a small number of bots are attacking a single server. Rate limiting is much more difficult when a large number of bots are directing small amounts of traffic to each member of a large distributed pool of servers.

2. Rate limiting systems encounter problems where a mixture of "good" and "bad" clients are hidden behind a single NAT or Proxy IP address and thus are all stuck being treated on equal footing.

3. Rate limiting schemes often penalize well-behaved good clients (which try to complete handshakes and may limit their number of
Client puzzles are complementary to rate-limiting and give servers another option than just rejecting some fraction of requests. A server can provide a puzzle (of varying and server-selected complexity) to a client as part of a HelloRetryRequest extension. The client must choose to either abandon the connection or solve the puzzle and resend its ClientHello with a solution to the puzzle. Puzzles are designed to have asymmetric complexity such that it is much cheaper for the server to generate and validate puzzles than it is for clients to solve them.

Client puzzle systems may be inherently "unfair" to clients that run with limited resources (such as mobile devices with batteries and slow CPUs). However, client puzzle schemes will typically only be evoked when a server is under attack and would otherwise be rejecting some fraction of requests. The overwhelming majority of transactions will never involve a client puzzle. Indeed, if client puzzles are successful in forcing adversaries to use a new attack vector, the presence of client puzzles will be completely transparent to end users.

It is likely that not all clients will choose to support this extension. During attack scenarios, servers will still have the option to apply traditional rate limiting schemes (perhaps with different parameters) to clients not supporting this extension or using a version of TLS prior to 1.3.

2. Notational Conventions

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 [RFC2119].

Messages are formatted with the notation as described within [I-D.ietf-tls-tls13].

3. Handshake Changes

Client puzzles are implemented as a new ClientPuzzleExtension to TLS 1.3 [I-D.ietf-tls-tls13]. A client supporting the ClientPuzzleExtension MUST indicate support by sending a ClientPuzzleExtension along with their ClientHello containing a list of puzzle types supported, but with no puzzle response. When a server wishes to force the client to solve a puzzle, it MAY send a
HelloRetryRequest with a ClientPuzzleExtension containing a puzzle of a supported puzzle type and with associated parameters. To continue with the handshake, a client MUST resend their ClientHello with a ClientPuzzleExtension containing a response to the puzzle. The ClientHello must otherwise be identical to the initial ClientHello, other than for attributes that are defined by specification to not be identical.

Puzzles issued by the server contain a token that the client must include in their response. This allows a server to issue puzzles without retaining state, which is particularly useful when used in conjunction with DTLS.

If a puzzle would consume too many resources, a client MAY choose to abort the handshake with the new fatal alert "puzzle_too_hard" and terminate the connection.

A typical handshake when a puzzle is issued will look like:

```
Client                                               Server

ClientHello                                          ServerHello
+ ClientPuzzleExtension                            ServerKeyShare
+ ClientKeyShare                                    {EncryptedExtensions*}
  ---------->                                          {ServerConfiguration*}
  + Certificate*                                     {Certificate*}
  + CertificateRequest*                              {CertificateVerify*}
  + CertificateVerify*                               {Finished}

  + CertificateVerify*                               {Finished}
  + [Application Data]                               [Application Data]

---------->                                          ---------->
```

Figure 1. Message flow for a handshake with a client puzzle

* Indicates optional or situation-dependent messages that are not always sent.
() Indicates messages protected using keys derived from the ephemeral secret.

[] Indicates messages protected using keys derived from the master secret.

Note in particular that the major cryptographic operations (starting to use the ephemeral secret and generating the CertificateVerify) are performed _after_ the server has received and validated the ClientPuzzleExtension response from the client.

3.1. The ClientPuzzleExtension Message

The ClientPuzzleExtension message contains an indication of supported puzzle types during the initial ClientHello, a selected puzzle type and puzzle challenge during HelloRetryRequest, and the puzzle type and puzzle response in the retried ClientHello:

```
struct {
    ClientPuzzleType type<1..255>;
    opaque client_puzzle_challenge_response<0..2^16-1>;
} ClientPuzzleExtension;
```

```
enum {
    cookie (0),
    sha256_reverse_cpu (1),
    sha512_reverse_cpu (2),
    sha256_reverse_memory (3),
    (0xFFFF)
} ClientPuzzleType;
```

type During initial ClientHello, a vector of supported client puzzle types. During the HelloRetryRequest, a vector of exactly one element containing the proposed puzzle. During the retried ClientHello, a vector containing exactly one element with the type of the puzzle being responded to.

client_puzzle_challenge_response Data specific to the puzzle type, as defined in Section (#puzzles). In the initial ClientHello, this MUST be empty (zero-length). During HelloRetryRequest, this contains the challenge. During the retried ClientHello, this contains a response to the challenge. Puzzles containing a token may have it within this field.
4. Usage by Servers

Servers MAY send puzzles to clients when under duress, and the percentage of clients receiving puzzles and the complexity of the puzzles both MAY be selected as a function of the degree of duress.

Servers MAY also occasionally send puzzles to clients under normal operating circumstances to ensure that the extension works properly.

Servers MAY use additional factors, such as client IP reputation information, to determine when to send a puzzle as well as the complexity.

5. Proposed Client Puzzles

Having multiple client puzzle types allows good clients a choice to implement puzzles that match with their hardware capabilities (although this also applies to bad clients). It also allows "broken" puzzles to be phased out and retired, such as when cryptographic weaknesses are identified.

5.1. Cookie Client Puzzle Type

The "cookie" ClientPuzzleType is intended to be trivial. The client_puzzle_challenge_response data field is defined to be a token that the client must echo back.

During an initial ClientHello, this MUST be empty (zero-length). During HelloRetryRequest, the server MAY send a cookie challenge of zero or more bytes as client_puzzle_challenge_response. During the retried ClientHello, the client MUST respond by resending the identical cookie sent in the HelloRetryRequest.

5.2. SHA-256 CPU Reverse Puzzle Type

This puzzle forces the client to calculate a SHA-256 [RFC5754] multiple times. In particular, the server selects a random number and challenge includes both the maximum possible value that the random number could be as well as a salt bytestring. The server communicates the maximum possible value that the number could be, along with the salt and the result of performing a SHA-256 across a number, the salt, and a label. The client solves the puzzle by finding the number (within the range) where the SHA-256 matches the provided value.
struct {
    opaque token<0..2^16-1>
    uint64 challenge_max;
    uint8 salt<0..2^16-1>
    uint8 sha256_result<32>
} SHA256CPUReversePuzzleChallenge;

struct {
    opaque token<0..2^16-1>
    uint64 challenge_solution;
} SHA256CPUReversePuzzleResponse;

token  The token allows the server to encapsulate and drop state, and
also acts as a cookie for DTLS. During an initial ClientHello,
this MUST be empty (zero-length). During HelloRetryRequest, the
server MAY send a token challenge of zero or more bytes. During
the retried ClientHello, the client MUST respond by resending the
identical token sent in the HelloRetryRequest. Servers MAY
include an authenticated version of challenge_max, sha256_result,
and salt in this token if they wish to be stateless.

salt  A server selected variable-length bytestring.

sha256_result  The expected result of performing a SHA-256 across the
challenge_solution, salt, and label.

challenge_max  The upper bound of the range that challenge_solution
is within. This is selected by the server to select the hardness
of the puzzle. The computational work that a client will need to
expend is intended to be O(challenge_max).

challenge_solution  The solution response to the puzzle, as solved by
the client. The server guarantees that 0 <= challenge_solution <=
challenge_max.

To find the response, the client must find the numeric value of
challenge_solution where:

    sha256_result = SHA-256(challenge_solution + salt + label)

where "+" denotes concatenation and where label is the NUL-terminated
value "TLS SHA256CPUReversePuzzle" (including the NUL terminator).

Clients offering to support this puzzle type SHOULD support
challenge_max values of at least 500,000. [[TODO: is this a good
value? https://en.bitcoin.it/wiki/Non-specialized_hardware_comparison
has a comparison of SHA256 on various
hardware.]]
(_TODO / Open question: Should this be HMAC-SHA256(salt, challenge_solution + label) or similar?_)  

5.3. SHA-512 CPU Reverse Puzzle Type  

The SHA-512 CPU Reverse Puzzle Type is identical to the "SHA256 CPU Reverse Puzzle Type" except that the SHA-512 [RFC5754] hash function is used instead of SHA-256. The label used is the value "TLS SHA512CPUReversePuzzle" and SHA512CPUReversePuzzleChallenge is updated accordingly:

```c
struct {
    opaque token<0..2^16-1>;
    uint64 challenge_max;
    uint8 salt<0..2^16-1>;
    uint8 sha512_result<64>;
} SHA512CPUReversePuzzleChallenge;
```

Clients offering to support this puzzle type SHOULD support challenge_max values of at least 250,000.  [[TODO: is this a good value?]]

5.4. SHA-256 Memory Reverse Puzzle Type  

[[_TODO: This puzzle is a place-holder not intended to be implemented until a better and asymmetric memory-hard puzzle is proposed and specified. Another, and likely better, alternative for a symmetric memory-hard puzzle would be to leverage the scrypt KDF._ [I-D.josefsson-scrypt-kdf]]]

This puzzle is more memory-heavy than CPU-heavy which may be a good option for mobile clients. Unfortunately, the memory-hard aspect of this puzzle is not yet asymmetric.

This puzzle starts similar to SHA256CPUReversePuzzle but to fold in another bytestring from information provided by the server.
struct {
    opaque token<0..2^16-1>;
    uint64 challenge_max;
    uint8 salt<0..2^16-1>;
    uint8 seed<0..2^16-1>;
    uint64 mem;
    unit32 noff;
    uint8 sha256_result<32>;
} SHA256MemoryReversePuzzleChallenge;

struct {
    opaque token<0..2^16-1>;
    uint64 challenge_solution;
} SHA256MemoryReversePuzzleResponse;

Additional parameters to those in SHA256CPUReversePuzzleChallenge and SHA256CPUReversePuzzleResponse:

seed  the seed to a PRF (pseudorandom function) from which a bytestream is generated  
mem  the number of bytes to expand from the PRF into memory  
noff  the number of offsets into the expand to pull from the PRF

So in specific:

expanded = PRF(seed, 0...mem-1)
extraction = SHA-256(expanded[PRF(seed, mem)]
                  + expanded[PRF(seed, mem+1)]
                  + ...
                  + expanded[PRF(seed, mem+noff)])

(_TODO: Formalize the details of the PRF, unless this puzzle is eliminated or replaced._)

In this case, challenge_solution is searched for in:

sha256_result = SHA-256(challenge_solution + salt 
                       + label + extraction)

Where label is the NUL-terminated string "TLS SHA256MemoryReversePuzzle".

Resource usage by clients is then _either_:

- Memory of O(1) and CPU of O(challenge_max) + O(noff * mem)
6. IANA Considerations

The IANA will need to assign an extension codepoint value for ClientPuzzleExtension.

The IANA will need to assign an AlertDescription codepoint value for puzzle_too_hard.

The IANA will also need to maintain a registry of client puzzle types.

7. Security Considerations

A hostile server could cause a client to consume unbounded resources. Clients MUST bound the amount of resources (cpu/time and memory) they will spend on a puzzle.

A puzzle type with economic utility could be abused by servers, resulting in unnecessary resource usage by clients. In the worst case, this could open up a new class of attacks where clients might be directed to malicious servers to get delegated work. As such, any new puzzle types SHOULD NOT be ones with utility for other purposes (such as mining cryptocurrency or cracking password hashes). Including fixed labels in new puzzle definitions may help mitigate this risk.

Depending on the structure of the puzzles, it is possible that an attacker could send innocent clients to a hostile server and then use those clients to solve puzzles presented by another target server that the attacker wishes to attack. There may be ways to defend against this by including IP information in the puzzles (not currently proposed in this draft), although that introduces additional issues.

All extensions add complexity, which could expose additional attack surfaces on the client or the server. Using cryptographic primitives and patterns already in-use in TLS can help reduce (but certainly not eliminate) this complexity.

An attacker that can force a server into client puzzle mode could result in a denial of service to clients not supporting puzzles or not having the resources to complete the puzzles. This is not necessarily worse than if the server was overloaded and forced to
deny service to all clients or to a random selection of clients. By using client puzzles, clients willing to rate-limit themselves to the rate at which they can solve puzzles should still be able to obtain service while the server is able to stay available for these clients.

It is inevitable that attackers will build hardware optimized to solve particular puzzles. Using common cryptographic primitives (such as SHA-256) also means that commonly deployed clients may have hardware assistance, although this also benefits legitimate clients.

8. Privacy Considerations

Measuring the response time of clients to puzzles gives an indication of the relative capabilities of clients. This could be used as an input for client fingerprinting.

Client’s support for this extension, as well as which puzzles they support, could also be used as an input for client fingerprinting.

9. Acknowledgments

Some of this was inspired by work done by Kyle Rose in 2001, as well as a 2001 paper by Drew Dean (Xerox PARC) and Adam Stubblefield (Rice) [SEC2001.DEAN]. Discussions with Eric Rescorla, Yoav Nir, Richard Willey, Samuel Erb, Rich Salz, Kyle Rose, Brian Sniffen, and others on the TLS working group have heavily influenced this proposal and contributed to its content. An alternate approach was proposed in [I-D.nir-tls-puzzles]. Some similar mechanisms for protecting IKE are discussed in [I-D.ietf-ipsecme-ddos-protection].

10. References

10.1. Normative References

[I-D.ietf-tls-tls13]


10.2. Informative References

[I-D.ietf-ipsecme-ddos-protection]
Nir, Y. and V. Smyslov, "Protecting Internet Key Exchange (IKE) Implementations from Distributed Denial of Service Attacks", draft-ietf-ipsecme-ddos-protection-01 (work in progress), March 2015.

[I-D.josefsson-scrypt-kdf]

[I-D.nir-tls-puzzles]
Nir, Y., "Using Client Puzzles to Protect TLS Servers From Denial of Service Attacks", draft-nir-tls-puzzles-00 (work in progress), April 2014.

[SEC2001.DEAN]

Author’s Address

Erik Nygren
Akamai Technologies

EMail: erik+ietf@nygren.org
URI: http://erik.nygren.org/