Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 1, 2017

E. Nygren
S. Erb
Akamai Technologies
A. Biryukov
D. Khovratovich
University of Luxembourg
A. Juels
Cornell University

December 28, 2016

TLS Client Puzzles Extension
draft-nygren-tls-client-puzzles-02

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.

Table of Contents

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 retries) much more heavily than they penalize attacking bad clients (which may try to disguise themselves as good clients, but which otherwise are not constrained to behave in any particular way).

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:

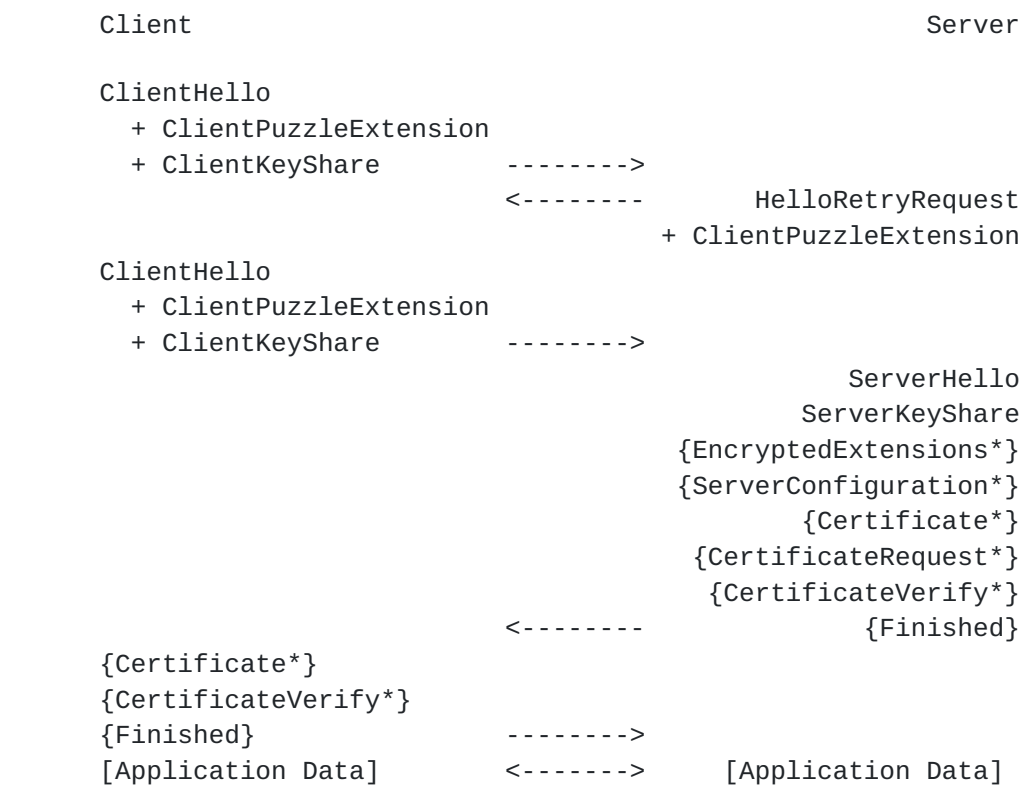

Figure 1. Message flow for a handshake with a client puzzle

* 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_cpu (1),
   sha512_cpu (2),
   birthday_puzzle (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 Puzzle Type

This puzzle forces the client to calculate a SHA-256 [RFC5754] multiple times. In particular, the server selects a difficulty and a random salt. The client solves the puzzle by finding any nonce where a SHA-256 hash across the nonce, the salt and a label contains difficulty leading zero bits.

```
struct {
    opaque token<0..2^16-1>;
    uint16 difficulty;
    uint8 salt<0..2^16-1>;
} SHA256CPUPuzzleChallenge;

struct {
    opaque token<0..2^16-1>;
    uint64 challenge_solution;
} SHA256CPUPuzzleResponse;
```

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 included an authenticated version of difficulty and salt in this token if they wish to be stateless.

difficulty filter affecting the time to find solution.

salt A server selected variable-length bytestring.

challenge_solution The solution response to the puzzle, as solved by the client.

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

SHA-256(challenge_solution || salt || label) contains difficulty leading zeros.

where "||" denotes concatenation and where label is the NUL-terminated value "TLS SHA256CPUPuzzle" (including the NUL terminator).

Clients offering to support this puzzle type SHOULD support a difficulty value of at least 18. [[TODO: is this a good value? https://en.bitcoin.it/wiki/Non-specialized_hardware_comparison has a comparison of SHA256 on various hardware.]]

5.3. SHA-512 CPU Puzzle Type

The SHA-512 CPU Puzzle Type is identical to the "SHA256 CPU Puzzle Type" except that the SHA-512 [RFC5754] hash function is used instead of SHA-256. The label used is the value "TLS SHA512CPUPuzzle".

Clients offering to support this puzzle type SHOULD support difficulty values of at least 17. [[TODO: is this a good value?]]

5.4. Equihash: Memory-hard Generalized Birthday Problem Puzzle Type

Using Equihash, the asymmetric memory-hard generalized birthday problem PoW [NDSS2016], this puzzle will force a client to use a significant amount of memory to solve. The solution to this puzzle can be trivially verified.

```
      struct {
         opaque token<0..2^16-1>;
         uint16 n;
         uint16 k;
         uint16 difficulty;
         uint8 salt<0..2^16-1>;
      } BirthdayPuzzleChallenge;

      struct {
         opaque token<0..2^16-1>;
         uint8 V<20>;
         uint8 solution<0..2^16-1>;
      } BirthdayPuzzleResponse;
```

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 included an authenticated version of n, k, difficulty and salt in this token if they wish to be stateless.

salt A server selected variable-length bytestring.

n, k parameters affecting the complexity of Wagner's algorithm.

difficulty secondary filter affecting the time to find solution.

V 20 byte nonce used in solution.

solution list of 2^k (n/(k+1)+1)-bit nonces used in solution, referred to as xi below.

In the further text, the output of blake2b is treated as a 512-bit register with most significant bits coming from the last bytes of blake2b output (i.e. little-endian conversion).

To compute the response, the client must find a V and 2^k solutions such that:

blake2b(salt||V||x1) XOR blake2b(salt||V||x2) XOR ... XOR blake2b(I||V||x(2^k)) = 0
blake2b(label||salt||V||x1||x2||...||x(2^k)) has difficulty leading zero bits.

where "||" denotes concatenation and where label is the NUL-terminated value "TLS BirthdayPuzzle" (including the NUL terminator). Incomplete bytes in nonces xi are padded with zero bits, which occupy the most significant bits.

The client MUST provide the solution list in an order that allows a server to verify the solution was created using Wagner's algorithm:

blake2b(salt||V||x(w_2^l+1)) XOR blake2b(salt||V||x(w_2^l+2)) XOR ... XOR blake2b(I||V||x(w*2^l+2^l)) has nl/(k+1) leading zero bits for all w,l.

and two 2^(l-1)(n/(k+1)+1)-bit numbers Z1 and Z2 must satisfy Z1<Z2 where

Z1 = x(w_2^l+1)||x(w_2^l+2)||...||x(w_2^l+2^(l-1)) Z2 = x(w_2^l+2^(l-1)+1)||x(w_2^l+2)||...||x(w_2^l+2^l) as in([NDSS2016] section 4A, 5C). The server MUST verify these intermediate equations.

A solution can be found using Wagner's algorithm as described in [NDSS2016]. The amount of memory required to find a solution is 2 ^ (n/(k+1)+k) bytes. A solution requires (k+1)2^(n/(k+1)+d) calls to the blake2b hash function.

Clients offering to support this puzzle type SHOULD support n, k values such that 2^(n/(k+1)+k) is at least 20MB.

Servers SHOULD look to minimize the value of k as 2^k blake2b hash operations will be required to verify a solution.

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.

Depeding 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

The story of client puzzles dates back to Dwork and Naor [DN92] and Juels and Brainard [JB99]. Some of this draft 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, 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 discused in [I-D.ietf-ipsecme-ddos-protection].

10. References

10.1. Normative References

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-13 (work in progress), May 2016.

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

[RFC5754] Turner, S., "Using SHA2 Algorithms with Cryptographic Message Syntax", RFC 5754, DOI 10.17487/RFC5754, January 2010, <http://www.rfc-editor.org/info/rfc5754>.

10.2. Informative References

[DN92] Dwork, C. and M. Naor, "Pricing via Processing or Combatting Junk Mail", Proceedings of Crypto'92 , 1992, <http://www.wisdom.weizmann.ac.il/~naor/PAPERS/pvp_abs.html>.

[I-D.ietf-ipsecme-ddos-protection] Nir, Y. and V. Smyslov, "Protecting Internet Key Exchange Protocol version 2 (IKEv2) Implementations from Distributed Denial of Service Attacks", draft-ietf-ipsecme-ddos-protection-06 (work in progress), April 2016.

[I-D.josefsson-scrypt-kdf] Percival, C. and S. Josefsson, "The scrypt Password-Based Key Derivation Function", draft-josefsson-scrypt-kdf-05 (work in progress), May 2016.

[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.

[JB99] Juels, A. and J. Brainard, "Client Puzzles: A Cryptographic Defense Against Connection Depletion Attacks", Proceedings of NDSS'99 , 1999, <http://www.wisdom.weizmann.ac.il/~naor/PAPERS/pvp_abs.html>.

[NDSS2016] Biryukov, A. and D. Khovratovich, "Equihash: Asymmetric proof-of-work based on the Generalized Birthday problem", February 2016, <https://www.internetsociety.org/sites/default/files/blogs-media/equihash-asymmetric-proof-of-work-based-generalized-birthday-problem.pdf>.

[SEC2001.DEAN]
Dean, D. and A. Stubblefield, "Using Client Puzzles to Protect TLS", Proceedings of the 10th USENIX Security Symposium , August 2001, <https://www.usenix.org/legacy/events/sec2001/full_papers/dean/dean.pdf>.

Authors' Addresses

Erik Nygren
Akamai Technologies

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

Samuel Erb
Akamai Technologies

EMail: serb@akamai.com

Alex Biryukov
University of Luxembourg

EMail: alex.biryukov@uni.lu

Dmitry Khovratovich
University of Luxembourg

EMail: khovratovich@gmail.com

Ari Juels
Cornell University

EMail: juels@cornell.edu