TLS Working Group
Internet-Draft
Intended status: Experimental
Expires: October 31, 2014

Y. Nir
Check Point

April 29, 2014

Using Client Puzzles to Protect TLS Servers From Denial of Service Attacks
draft-nir-tls-puzzles-00

Abstract

This document proposes a mechanism for mitigating denial of service (DoS) and distributed denial of service (DDoS) attacks on TLS servers. Attackers are limited in their ability to cause resource waste on the server by requiring proof of work by the client before these CPU resources are expended.

Status of this Memo

Copyright Notice

1. Introduction

TLS ([TLS]) is vulnerable to a simple denial of service (DoS) attack. Arbitrary hosts on the Internet can begin a TLS handshake with a server. The attacking clients need only generate a valid TLS ClientHello and a valid-looking ClientKeyExchange, both of which require very little computing resources.

To respond, the server has to perform some CPU-intensive operations. Depending on the chosen ciphersuite, the server will need to perform an RSA decryption or a Diffie-Hellman exchange plus a RSA, DSA, or ECDSA signature. With such asymmetric work, it's easy for an attacker or a modestly sized bot-net to overwhelm a server's resources, creating a DoS or distributed DoS (DDoS) attack.

The extension described in this document mitigates this attack, by forcing the attacker to perform some work (a partial break of a hash function) before the server agrees to process the ClientKeyExchange. This has a marked effect on the latency, so Section 4 recommends against using this feature when the server is not under extreme load.

1.1. Conventions Used in This Document

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].

2. Protocol Overview

A supporting client will send a new empty extension in the ClientHello message. This new extension is called client_puzzle.

A supporting server's reply depends on whether the feature is activated or not. It is expected that conforming servers will have an option to set this feature to be unconditionally deactivated and unconditionally activated, but at least the latter will be for testing purposes only. The usual configuration will be to activate the feature dynamically in response to a heavy load which may be a result of an attempt at a DoS attack.

When deactivated, the server MAY either respond with an empty client_puzzle extension in its ServerHello, or it MAY omit the extension entirely. Either way, no further action is required from either client or server. TBD: Should we mandate one or the other?

When activated, the server SHOULD generate a 32-byte value called a Mystery, and send a non-empty client_puzzle extension as described in Section 2.1. This extension is sent in the ServerHello. Exceptions to the SHOULD-level requirement are in Section 5.

A conforming client that has received a non-empty client_puzzle extension MUST respond with one of the following:

- o Terminate the connection with a decode_error alert if the client_puzzle extension sent by the server was malformed.
- o Terminate the connection with a puzzle_error alert if either the puzzle is too hard, or all attempts to solve the puzzle failed. This new alert is described in Section 2.2.
- o A PuzzleSolution handshake message with the solution to the puzzle. This new handshake message is defined in Section 2.3. The client MUST send this new handshake message before the optional Certificate message and the (non-optional) ClientKeyExchange message.

When the feature is activated and a puzzle has been sent in the extension, the server MUST NOT process the Certificate or the ClientKeyExchange messages if the PuzzleSolution is missing. In that case, it MUST terminate the connection with a puzzle_error alert. A puzzle_error alert is also used if the solution in the PuzzleSolution message is incorrect.

If the puzzle solution is correct, the server continues the handshake normally.

The diagram below outlines the protocol structure.

```
         Client                                               Server

         ClientHello(*)              -------->
                                                       ServerHello(*)
                                                         Certificate*
                                                   ServerKeyExchange*
                                                  CertificateRequest*
                                     <--------        ServerHelloDone
         PuzzleSolution
         Certificate*
         ClientKeyExchange
         CertificateVerify*
         [ChangeCipherSpec]
         Finished                    -------->
                                                   [ChangeCipherSpec]
                                     <--------               Finished
         Application Data            <------->       Application Data

       (*) The ClientHello and ServerHello include the client_puzzle
           extension to indicate support
```

2.1. The client_puzzle Extension

The client_puzzle extension is a ClientHello and ServerHello extension as defined in section 2.3 of [TLS-EXT]. The extension_type field is TBA by IANA.

The format of the non-empty extension is to be added, but it has to include the following fields:

- o A 32-byte field that contains the SHA2-256 hash of the (not transmitted) Mystery.
- o A 1-byte field that contains the difficulty level of the puzzle.
- o A 32-byte "masked puzzle" value, which is the same as the Mystery, but with the last n bits set to zero, where n is the number in the difficulty field.

2.2. The puzzle_error Alert

This alert code is to be assigned by IANA. This alert is always fatal.

2.3. The PuzzleSolution Handshake Message

The format of the PuzzleSolution record is to be added, but it will contain the 32-byte Mystery. The content type is TBA by IANA.

The client finds this value by trying different values for the masked

bits in the "masked puzzle" field in the client_puzzle extension.

3. Processing

3.1. Client Processing

The client MUST always send an empty client_puzzle extension in the ClientHello.

If no client_puzzle extension is present in the ServerHello, no further action is needed.

If an empty client_puzzle extension is present in the ServerHello, no further action is needed.

If a non-empty client_puzzle extension is present in the ServerHello, the verify that it is properly formatted and that the difficulty level is acceptable. The difficulty level is acceptable if all 2^n possibilities can be tested by the client in a reasonable amount of time. If either check fails, the client MUST terminate the connection with a puzzle_error alert.

The client then proceeds to attempt to solve the puzzle. If the puzzle is unsolvable, the client MUST terminate the connection with a fatal puzzle_error alert.

Having found a correct Mystery value, the client sends that value in a PuzzleSolution message, and proceeds with the rest of the TLS handshake.

3.2. Server Processing

Puzzle protection is enabled or disabled depending on load. See Section 4 for considerations.

When disabled, it is acceptable for the client to not send a client_puzzle extension. In this case, the server MUST NOT send a client_puzzle extension.

When disabled, if the client does send an empty client_puzzle extension, the server MAY reply with an empty client_puzzle extension. If the feature is administratively disabled, then the server SHOULD NOT send an empty client_puzzle extension, whereas if it's only disabled because the server is not under load, it SHOULD send the extension.

When puzzles are enabled, but the client did not send a client_puzzle

extension, the server MUST terminate the connection with a puzzle_error alert.

When puzzles are enabled and the client has sent an empty client_puzzle extension, the server generates a random 32-byte string called the Mystery. To construct the extension, the server hashes the Mystery, sets a difficulty level (n), and adds the Mystery to the extension data field, clearing the last n bits. The server also stores the Mystery value.

When the client returns a PuzzleSolution handshake message, the server compares the Mystery value to the value in the PuzzleSolution. If they match, or if their SHA2-256 hashes match, processing continues as usual. If not, the server MUST terminate the session with a puzzle_error alert.

4. Operational Considerations

[This section needs a lot of expanding]

Not all Clients support this extension, so this protection should only be enabled when the server detects an unusual load that could indicate an attack. Detecting this is beyond the scope of this document.

The difficulty level should be set by balancing the requirement to minimize the latency for legitimate clients and making things difficult for attackers. A good rule of thumb is for taking about 1 second to solve the puzzle. A typical client or bot-net member in 2014 can perform slightly less than a million SHA2-256 hashes per second per core, so setting the difficulty level to n=20 is a good compromise. It should be noted that mobile initiators, especially phones are considerably weaker than that. Implementations should allow administrators to set the difficulty level, and/or be able to set the difficulty level dynamically in response to load.

Note that when this feature is enabled, it causes a latency of about one second. Such a delay may be unnoticeable for SMTP servers, but would be very irritating to browser users. These considerations should be taken into account when determining activation thresholds and difficulty levels.

Clients should set a maximum difficulty level beyond which they won't try to solve the puzzle and log or display a failure message to the administrator or user.

This mechanism is appropriate for TLS. For DTLS, initiating multiple

handshakes so as to load the server with memory allocations may be a more attractive attack vector, one that this feature does not protect against. See Appendix A for an alternate design that could be appropriate for DTLS.

5. Security Considerations

To be added

6. IANA Considerations

IANA is requested to allocate:

- o An alert type in the "TLS Alert" registry, with puzzle_error in the Description field, "Y" in the DTLS-OK field, and this document as reference.
- o An extension type in the "ExtensionType Values" registry. The extension name should be "client_puzzle" and the reference should be this document.
- o A handshake message type in the "TLS HandshakeType" registry with description "puzzle_solution", "Y" for DTLS-OK, and this document as reference.

7. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[TLS] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[TLS-EXT] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, April 2006.

Appendix A. An Alternate Design

This appendix broadly describes an alternate design. I rejected this in favor of the design in the main part of the document, because that one seemed more appropriate for a TCP-based protocol such as TLS.

With this design, the client does not need to indicate support. When the server is under load, it responds to a ClientHello with an alert that carries the puzzle, and terminates the TLS connection. The client then solves the puzzle, and starts a new connection, including

the puzzle solution in a ClientHello extension.

Advantages of this alternate design:

- No state is needed on the server while the client is solving the puzzle. Specifically, no sockets are occupied.
- No new handshake messages are required.
- Appropriate for DTLS, because the statelessness while the client is solving the puzzle mitigates the state allocation attack.

Disadvantages of this alternate design:

- Requires an extra TCP handshake, increasing latency in a protocol extension that already has a latency issue.
- Has a challenge with recognizing puzzles and preventing re-use of a solution. Mysteries need to be generated in such a way that the server will recognize them as Mysteries that it has generated, and also won't allow re-use of a solved Mystery by multiple clients.

The challenge above is not insurmountable. It is possible to generate the Mystery in a stateless way by making it depend on client IP address and on a timestamp. Alternatively, the server can allocate state and store the issued Mysteries, deleting them when they are received to prevent re-use, and expiring them after some appropriate time span (at least round-trip time plus the time it takes the slowest client to solve the puzzle).

Author's Address

Yoav Nir
Check Point Software Technologies Ltd.
5 Hasolelim st.
Tel Aviv 6789735
Israel

Email: ynir.ietf@gmail.com