Volume 51, Number 2, 2010

ACTA TECHNICA NAPOCENSIS
Electronics and Telecommunications

TLS PROTOCOL: SECURE PROTOCOL WITH CLIENT PUZZLES

Raluca CATARGIU Monica BORDA
Faculty of Electronics,Telecommunications and Information Technology , Cluj-Napoca
Technical University of Cluj-Napoca, 26-28 Barițiu Street, 400027, ralu_catargiu@yahoo.com

Abstract: Client puzzles are commonly proposed as a solution to denial-of-service attacks. We analyzed the TLS protocol and implement Client Puzzles algorithm. We pursued the following objectives: analyze TLS handshake protocol and implement Client Puzzles algorithm using Hash Functions, Diffie-Hellman Base Puzzle Scheme and Threshold Puzzles. The originality of the paper is due to a comparative study of methods used to create the Client Puzzle solution. We also present measurements of execution time when our modified protocol is under attack.

Keywords: *client puzzles, attack, threshold client puzzles.*

I. INTRODUCTION

An attack such as Denial of Service (DoS) is an information security incident in which an organization or an independent user is deprived of the services of a resource that is normally used without a problem.

An attack "denial of service" is characterized by an explicit attempt by attackers to prevent legitimate users of a service to use the service.

There are five basic types of attack:

• Consumption of system resources;
• Configuration data destruction;
• Unsolicited resetting of TCP sessions;
• Destruction of physical network components;
• Obstruction of the communication channel between legitimate users and the victim so that they can not communicate properly;

These "zombie" systems are capable of receiving commands from a central operations center via encrypted channels. The main reason for the very existence of such "zombies" is the generation of bogus traffic targeted to a specific website. In order to make tracking more difficult, the source IP address may be spoofed but in the same time may be chosen from the same subnet in order to avoid egress filtering [3].

The client should always commit its resources to the authentication protocol first and the server should be able to verify the client commitment before allocating its own resources. The client's costs can be artificially increased by asking it to compute solutions to puzzles that are easy to generate and verify but whose difficulty for the solver can be adjusted to any level. The server should remain stateless and refuse to perform expensive cryptographic operations until it has verified the client's solution to a puzzle[1].

For ecommerce sites, the attacker could easily arrange an attack such that the website remained available, but web surfers are unable to complete any purchases. Such an attack is based on going after the secure server that processes credit card payments. The SSL/TLS protocol, as it stands, allows the client to request the server to perform an RSA decryption without first having done any work. RSA decryption is an expensive operation; the largest secure site we are aware of can process 4000 RSA decryptions per second. If we assume that a partial SSL handshake takes 200 bytes, then 800 KB/s is sufficient to paralyze an ecommerce site. Such a small amount of traffic is much easier to hide[3].

The original idea for the introduction of Client Puzzle in cryptography has had in his 1974 paper, Merkle[2] for key agreement rather than access control. Aura, Nikander and Leiwo apply client puzzles to authentication protocols in general [1]. Rivest, Shamir, and Wagner posed the related problem of time-lock cryptography in their 1996 paper [4].

The idea of requiring the client to commit its resources first was described early by Dwork and Naor [5]. They suggested increasing the cost of electronic junk mailing by asking the sender to solve a small cryptographic puzzle for each message. The cost would be negligible for normal users but high for mass mailers.

II. TLS PROTOCOL

The primary goal of the TLS Protocol is to provide privacy and data integrity between two communicating applications. The protocol is composed of two layers: the TLS Record Protocol and the TLS Handshake Protocol. At the lowest level, layered on top of some reliable transport protocol (e.g., TCP [TCP]), is the TLS Record Protocol. The TLS Record Protocol provides connection security that has two basic properties:

• The connection is private. Symmetric cryptography is used for data encryption (e.g., DES [DES], RC4 [RC4], etc.);
• The connection is reliable. Message transport includes a message integrity check using a keyed MAC. Secure hash functions (e.g., SHA, MD5, etc.) are used for MAC computations;

Handshake Protocol produces the cryptographic parameters of the session state.

Manuscript received May 8, 2010; revised June 9, 2010

III. CLIENT PUZZLE ANALYSIS

To date, researches have proposed several schemes, such as the hash function based puzzles, the Diffman-Hellman based puzzles and the game theory based client puzzles.

A. Hash Function - based Puzzle Scheme

Aura in paper [1] (compared with Juels and Brainard present more convincing the specifications one-way hash function) a defending server sends a puzzle's parameters to a client and the client perform brute-force to find the correct solution. The method purpose the server to decide difficulty level of the puzzle (k) and together with the Ns is sent to the client.

The advantages are [8]:

1. Before server verifies the solutions, no memory space need to be distributed to the clients.
2. The value k if controlled by the server
3. The solutions are hard to know before because of the sort valid time for one single Ns.

The shortcomings are:

a. The server has to keep a record to check if certain Ns has not been calculated before.
b. The computation cost for the client is hard to predict and the waiting time for the solution is unknown.
c. The vital weakness of this scheme is that the server has to perform a similar hash computation to check if the client solution is correct.

B. Diffie-Hellman Based Puzzle Scheme

In 2004, Waters and Juels[7] proposed a scheme, witch is based on Diffie-Hellman key agreement, and permits the outsourcing of puzzles. To eliminate puzzle construction as a target o DOS attacks, an external service (named "bastion") is introduced in their paper to help multiple servers distribute puzzles. Many servers can rely on the same bastion, while the bastion need to know witch server are deploying its service. A web server may limit the maximum number of TCP connections of channels, when it encounters
a DoS attack. At every time interval, the bastion generates client puzzles for distinct channels, which are suitable for all servers that rely on the bastion's service. Moreover, each client has a unique identity, such as a public key (Figure 1).

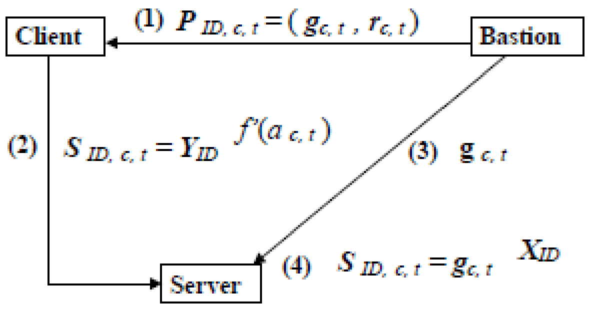
Client
Bastion
Server
(1) $P_{ID,c,t} = (g_{c,t}, r_{c,t})$
(2) $S_{ID,c,t} = Y_{ID}^{f(a_{c,t})}$
(3) $g_{c,t}$
(4) $S_{ID,c,t} = g_{c,t}^{X_{ID}}$

Figure 1.Diffie Hellman Client Puzzle Construction.

The advantages are [8]:

1. This method allows clients to solve client offline obtained from a bastion and solve it in valid time. The client than combines this solution with a server's public key to compute the specific solution (offline).
2. The deployment of virtual channels help the server detect DoS attacks and allocate system resources rationally. By inspecting the usage of channels, the servers can determine whether it is under attack and limit communication to a restricted collection of channels.

The problems of this method are:

a. In this scheme there is only one channel for one puzzle and if an attacker can solver several of them, he can easily find the resources of others channels. Due to this attack the legitimate clients who solve the puzzle for these channels cannot obtain the response.
b. If an attacker is able to obtain the packet he/she may exploit the solution computed by legitimate clients.
c. All servers that depend on the same bastion are an uniform configuration (same number of channels, timeslot).
d. The solution is exponential and is expensive.

C. Threshold Client - Puzzle

Suppose that a server authenticates a number of legitimate clients and the initial puzzle difficulty is set to zero. When a strong attack is in progress, the server has the tendency to gradually increase the puzzle difficulty up to high values in order to cope with the important amount of work required to service the attacker’s requests. While puzzle difficulty may be increased up to impossible, this also means a DOS attack in its own right targeted against legitimate clients who may never solve a puzzle such difficult.

Although not very likely, a strong attack is possible. If an attacker had access to other N computers (with N being sufficiently large so we speak about massive computing power), then time needed to solve a puzzle with difficulty k would be divided by N.

In order to determine the time to resolve a puzzle, a threshold was introduced [9] for both the scheme in *A* and *B* above:

$$\mathbf{T_{estimated} = (2^{k+2}) * T_{SHA},}$$
$$\mathbf{T_{estimated} = (2^{k}) * T_{MD5}}$$

Where: $T_{estimated}$ - estimate time for solving the puzzle,
T_{SHA} - time for solve SHA operation,
Hash (ServerNonce, ClientNonce, ClientID,x) = 0...0Y,
T_{MD5} - time for solve MD5 operation
(x<1, k>|x<k+1, L>)=MD5, where x<1, k>
k -puzzle difficulty level control by the server
T_{SHA} and T_{MD5} - 0.01 – 0.02 milliseconds

The basic idea is to add the timestamp at which the server nonce was generated to the list «NS, NC, X, k» which is kept by the server in order to prevent reusing puzzle instances. When the server receives a solution to a puzzle, it can calculate the time it took the client to solve the puzzle and that should not be less than an estimated duration.

The solution for this problem is:

Limiting the difficulty level with $T_{estimated}$, each response from client should be compared with the calculation made by the server and the results received under that value will be made with error.

IV. CLIENT PUZZLE - Hash Function development

The server in an authentication protocols can ask the client to solve a puzzle before the server creates a protocol state or computes expensive functions such as exponentiation.
Properties for the client puzzle (Figure 2):
1. Creating a puzzle and verifying the solution is inexpensive for the server.
2. The cost of solving the puzzle is easy to adjust from zero to impossible.
3. The puzzle can be solved on most types of client hardware (although it may take longer with slow hardware).
4. It is not possible to know before solutions to the puzzles.
5. While the client is solving the puzzle, the server does not need to store the solution or other client specific data.
6. The same puzzle may be given to several clients. Knowing the solution of one or more clients does not help a new client in solving the puzzle.
7. A client can reuse a puzzle by creating several instances of it.
8. The puzzle should not be solved in less than a predetermined amount of time.

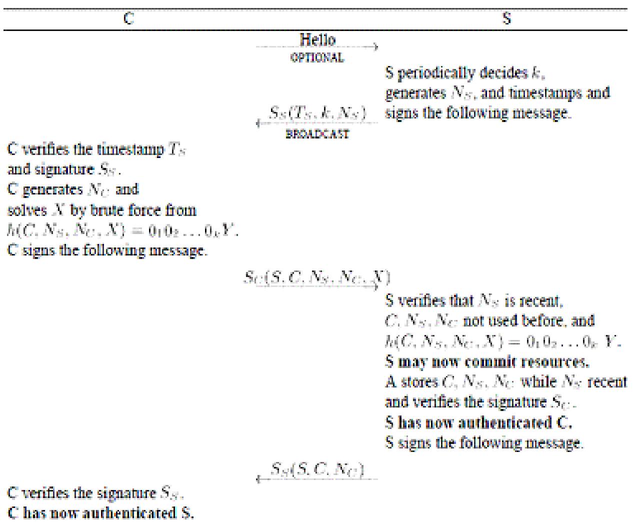
C
S
Hello
OPTIONAL
S periodically decides k, generates N_S, and timestamps and signs the following message.
$S_S(T_S, k, N_S)$
BROADCAST
C verifies the timestamp T_S and signature S_S.
C generates N_C and solves X by brute force from $h(C, N_S, N_C, X) = 0_1 0_2 \ldots 0_k Y$.
C signs the following message.
$S_C(S, C, N_S, N_C, X)$
S verifies that N_S is recent, C, N_S, N_C not used before, and $h(C, N_S, N_C, X) = 0_1 0_2 \ldots 0_k Y$.
S may now commit resources.
A stores C, N_S, N_C while N_S recent and verifies the signature S_C.
S has now authenticated C.
S signs the following message.
$S_S(S, C, N_C)$
C verifies the signature S_S.
C has now authenticated S.

Figure 2.Dos-Resistant Authentication with Public-Key Signature.

To create new puzzles, the server periodically generates a nonce Ns and sends it to the clients; the nonce needs to be random and not predictable like, for example, time stamps. (About 64 bits of entropy is sufficient to prevent the attacker from creating a database of solutions from which it could frequently find a matching nonce).
The server also decides the difficulty level (k)of the puzzle and together form the puzzle that is sent to the client.

To solve the puzzle (Figure 3), the client generates a random nonce Nc. The purpose of this nonce is twofold. First, if the client reuses a server nonce Ns, it creates a new puzzle by generating a new Nc. Second, without the client nonce an attacker could consume a specific client's puzzles by computing solutions and sending them to the server before the client does. (About 24 bits of entropy is enough to prevent an attacker from exhausting the values of Nc given that Ns changes frequently).

The server changes the value of Ns periodically (for example, every 60 seconds) to limit the time clients have for know before solutions. As long as the server accepts solutions for a certain value of Ns, it must keep book of the correctly solved instances so that the solutions cannot be reused.

$$h(C, N_S, N_C, X) = \overbrace{000\ldots000}^{\text{the } k \text{ first bits of the hash}} \underbrace{Y}_{\text{the rest of the hash bits}}$$

Figure 3.Client Puzzle Solution.

Where:
- h – Cryptographic hash function, such as MD5 or SHA
- C – client identity
- NS – server generated nonce
- NC – client generated nonce
- X – solution of the puzzle
- 000...000 – the first bits of the hash value; must be zero

The cost of solving the puzzle depends exponentially on the required number k of zero bits in the beginning of the hash. If k=0, no work is required. If k = 128(for MD5) or if k=192(for SHA), the client must reverse the entire one-way hash function, which is computationally impossible.

To determine the difficulty (k) of the puzzle, the opinions are divided, according to [1] the values are between 0 and 64(but also included that the puzzle can be solved on a wide range of hardware because the hash functions are one of the simplest cryptographic operations), according to [3] the best approach would be the number of already committed RSA operations rather than the current processor load or the number of incoming requests. Unfortunately, the puzzle difficulty follows an exponential curve and thus it is limited in practical purposes. To solve a puzzle of difficulty k, the client needs to perform on average $2^k - 1$ operation.

In order to obtain a more accurate scale for the puzzle difficulty parameter, Jules and Brainard [6] proposed that puzzles be split into several smaller puzzles of equal difficulty that should be solved separately and the general result is the combined individual result. Aura, Nikkander and Leiwo [1] stated that the same granularity can be achieved by combining sub-puzzles of varying difficulty, at a slightly lower cost for the server, but that is yet to be confirmed by experiment.

To defeat TCP SYN attacks, Juels and Brainard[6] proposed a new scheme to remediate the flaws in the TCP connection establishment protocol. This is the client protocol, witch is referred as an infrastructure of clients puzzles. It is described how can an attacker named A to make use of client/server protocol M. Meanwhile, a number of legitimate clients {Ci} exists in the same network and may ask for service from server S. Assumptions:
1. A cannot modified packets from Ci to S;
2. A cannot delay packets sent from any Ci to S
3. A can read any messages sent to any IP address.

V. EXPERIMENTAL WORK

In order to complete the experimental work, we developed in these environments:

- Microsoft Visual Studio 2005(C#),
- MentalisSecurityLibrary - an excellent open source library and modified it for SSL/TLS protocol with client puzzle algorithm,
- VeriSign SSL Certificate - a trial certificate (14 days) for server authentication.
- IIS (v6) - Internet Information Service - to manage the server for the site (Default Web Site).

The client-server handshake protocol was modified; the Client Puzzle algorithm introduced another two messages: *PuzzleRequest* and *PuzzleReply* between *ClientHello* and *ServerHello* messages.

The *PuzzleRequest* message is a server sign function Ss (Ts, k, Ns) applied to the following values:

- Puzzle difficulty k; where k >=0 and k<=100,
- Server Nonce (Ns-64 bits) generated periodically:Ns = a+32767*b, where a, b random numbers
- Timestamp Ts - current computer's date time.

To create a *PuzzleReply* message, the client should calculate the solution with brut force and it has to cycle the values(in interval 0-max(64bits)) until are number 0s in Y(equal with the difficulty k) to find the solution X.

The values of the *PuzzleReply* message create a sign message Sc(S,C,Ns,Nc,X):

- Server ID S, Client ID C,
- Server Nonce Ns, Client Nonce Nc(24 bits),
- The solution X.

In Client Puzzles experiments the values vary for k (puzzle difficulty). A server load value large will generate multiple calculations for the client.

When at least one of the clients is malicious, the server load increases dramatically and so does the puzzle difficulty. The legitimate clients are forced to solve difficult puzzles and at a certain point, the delay experienced by them is prohibitively long, causing a DOS attack targeted against legitimate clients themselves.

Execution times for each difficulty level are described in Table 1.

For a value of k greater then 20 the puzzle solution becomes prohibitive and execution time is exponential.

Puzzle Difficult k	Execution Time (ms)	Puzzle Difficult k	Execution Time(ms)
1	0	12	900
2	0	14	1100
3	0	16	1800
7	3	17	3200
8	5	18	7228
9	15	19	13200
10	35	20	30151

Table 1. Client Puzzle Execution Time

The values of execution time in Table1 follow an exponential increase; the solution is to compare the values of each puzzle execution time with a threshold.

The value of threshold is depending on hash function used; the hash function used is MD5 and the threshold value is Testimated = $2^k * T_{MD5}$. The estimated time represents the acceptance threshold for the client puzzle. If the execution time of a puzzle is less than the threshold value, the handshake protocol will generate an error.

Aura, Nikkander and Leiwo [1] proposed that puzzles be split into several smaller puzzles of different difficulty; V.Bocan [10] presents that every new puzzle is based on puzzles generated in Table 1. The exponential time execution values will be transformed into linear values; e.g. for k=100 execution time smaller than 30000 ms (Table2).

Puzzle Difficult k	Execution Time (ms)	Puzzle Difficult k	Execution Time(ms)
10	1504	70	1821
20	5225	90	2123
30	6405	100	25200

Table 2 Client Puzzles Linear Algorithm

The solution to make the pre-computation attacks even more difficult the hash functions used to compute client puzzles could be alternately changed.

VI. CONCLUSIONS

In this paper we have studying the methods used to create Client Puzzles and we have shown that Client Puzzles are vulnerable to attacks (DoS). The main measure was to protect the legitimate clients and in this scope was introduced the threshold value.

REFERENCES

[1] T. Aura, P. Nikander, J. Leiwo "DOS-resistant Authentication with Client Puzzles", 2001. http://research.microsoft.com/users/tuomaura/Publications/auranikander-leiwo-protocols00.pdf.
[2] R. C. Merkle "Secure Communications Over Insecure Channels", Communications of the ACM, April 1978.
[3] D. Dean, A. Stubblefield "Using Client Puzzles to Protect TLS", http://www.csl.sri.com/users/ddean/papers/usenix01b.pdf
[4] R. L. Rivest, A. Shamir,D,A, Wagner "Time lock puzzles and timed-release Crypto", Massachusetts Institute of Technology Cambridge, MA, USA, 1996.
[5] C. Dwork and M. Naor. "Pricing via processing or combating junk mail. In Advances in Cryptology", Springer-Verlag London, UK, 1992.
[6] Ari Juels, J. Brainard – "Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks", Springer Berlin / Heidelberg, 1999.
[7] J.A. Halderman, B.Waters, A.Juels and E.W.Felten "New client puzzle outsourcing techniques for dos-resistance", Washington DC, USA, 2004.
[8] Yi Gao "Efficient Trapdoor-based client puzzle system against DoS attack", University of Wollongong,Australia, 2005.
[9] V. Laurents "Requirements for Client Puzzles to Defeat the denial of Service and the Distributed Denial of Service Attacks", The International Arab Journal of International Technology, 2005.
[10] V. Bocan "Studiu asupra nivelului de sigurata oferit de protocoalele de autentificare", Technical University of Timisoara, 2004.