ISSN 1512-1232

CLIENT PUZZLE TECHINQUE TO OVERCOME DDOS IN TLS

First R.Sathyaraj

Assisstant Professor,School of Computing science and Engineering,
VIT University Vellore, Tamilnadu, India

***Abstract*:**

Denial of service by server resource exhaustion has become a major security threat in open communications networks. Public-key authentication does not completely protect against the attacks because the authentication protocols often leave ways for an unauthenticated client to consume a server's memory space and computational resources by initiating a large number of protocol runs and inducing the server to perform expensive cryptographic computations. Placing puzzles at the IP layer fundamentally changes the service paradigm of the Internet, allowing any device within the network to push load back onto those it is servicing. An advantage of network layer puzzles over previous puzzle mechanisms is that they can be applied to all traffic from malicious clients, making it possible to defend against arbitrary attacks as well as making previously voluntary mechanisms mandatory. Although client puzzles are often proposed as a solution to denial-of service attacks, this research explore TLS DDoS attack mitigation. This method using DH based puzzle construction and shows expected result with puzzle and without using puzzle.

Keywords: *IP, TLS, DDOS.*

I. Introduction

Denial-of-service attacks have become a major problem on the Internet. Major web sites have been taken down for several hours at a time by distributed denial of- service (DDoS). The attackers have shown an interesting combination of skill and ignorance. They are able to break into tens or hundreds of machines and install their tool of choice. They then use these "zombie" machines to actually launch the DDoS attack. Some of the tools even use encrypted communications between the attacker and zombie machines. The tools forge the source IP address on the traffic they generate in order to make determining the zombie machine somewhat harder. They will pick IP addresses that are on the same subnet, in order to overcome egress filtering. However, the tools work via brute force: they just generate random traffic (perhaps with a political message) aimed at a particular machine. While generating a gigabyte per second of traffic aimed at a single machine will bring most websites down to their knees, the sheer volume traffic stands out for anyone doing network monitoring. 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.

II. Related work

Client puzzles have been proposed previously in the literature to combat DoS attacks [1]. The basic idea behind a client puzzle is to have a client prove its legitimacy by devoting some its time

ISSN 1512-1232
and resources before a remote host will perform any action. Unlike other packet filtering schemes, client puzzles is a technique that does not need to distinguish between legitimate traffic and attack traffic. Instead, client puzzles rate limit all incoming traffic, including attack traffic, by requiring each client to solve puzzles to receive service. Recently, client puzzle schemes integrated into the IP layer have been proposed to mitigate flooding attacks. Congestion Puzzles by Wang and Reiter [7] is a recently proposed IP-layer puzzle scheme. When the puzzle mechanism is activated, each client is requiring solving puzzles before their packets are forwarded by a congested router. Clients continuously send separate probe packets along with regular data packets. When a router downstream detects congestion, it relays the probe packets toward the destination by changing the ICNMP code number to resemble a ping when it reaches the victim. This packet will be modified to contain the puzzle information (i.e., a nonce and the difficulty level). When a client receives the challenge, it begins to continuously solve puzzles and embeds the solutions in separate ICMP packets. The client takes the nonce it received from the router, creates its own nonce and uses both of those items to create the puzzle. Therefore, the client does not need to contact the congested router to get a new puzzle. The client sends the solutions, embedded in ICMP packets, towards the destination which are later intercepted by the router for puzzle verification. After correct verification of the puzzles, tokens are added into a token bucket at the congested router. When a data packet arrives, tokens are removed from the bucket. While each client is sending data packets and puzzle solution packets, it is also concurrently sending probe packets so it can receive new puzzle information. The major drawback of Congestion Puzzles is that an attacker can exploit the token bucket design by flooding the network with packets (without solving puzzles) in the hopes that this action will remove tokens that were supplied earlier by legitimate clients. The authors call it the 'free-riding' problem.

III. Technical challenges

Implementing a puzzle protocol at the IP layer is a difficult problem that requires careful consideration of several technical issues, including: seamless intimation of the puzzle-handshake procedure into IP, granularity of puzzle generation (e.g., per packet puzzles or per-flow puzzles), computation and storage overhead imposed on the puzzle server, communication overhead, and countering protocol circumvention. A client puzzle protocol is a connection-oriented protocol that requires a three-way handshake between the puzzle solver (i.e., client) and the puzzle generator/verifier (i.e., "puzzle server" or router in an IP-layer puzzle scheme). A client initiates the protocol by sending the first service-request packet to a puzzle server, the puzzle server responds by sending back a puzzle challenge; the client responds by sending back the puzzle solution. Since a TCP connection is established using a somewhat similar three-way handshake, a puzzle protocol can be readily integrated with TCP. Unfortunately, the same is not true for IP. Unlike TCP, IP is an inherently connectionless protocol that transfers each packet from hop to hop until it reaches the destination. Obviously, requiring a client and router to perform a three-way handshake for every puzzle is not practical. Thus, an ideal approach to integrating the handshake procedure with IP is to enable each client to create its own puzzles (with some initial input from the puzzle generator) so that constant interaction with a puzzle generator is not needed. But at the same time, puzzles need to be unpredictable so that a client cannot pre-compute solutions ahead of time. Designing a protocol that has both features is a challenging problem. To a large extent, the method employed for handling puzzle information (e.g., difficulty level, nonce, solution, etc.) determines the way the handshake procedure is integrated with IP. Congestion Puzzles proposed by Wang and Reiter [7] uses a two-channel approach: puzzle information and regular data are kept in separate packets. By using the two-channel approach, they were able to integrate a puzzle protocol within the IP layer without requiring the client and router to engage in repeated three-way handshakes.

ISSN 1512-1232

IV. Design

The TLS protocol breaks up the underlying TCP stream into a record oriented protocol. The unshaded portions of Figure 1 diagram the opening TLS handshake.

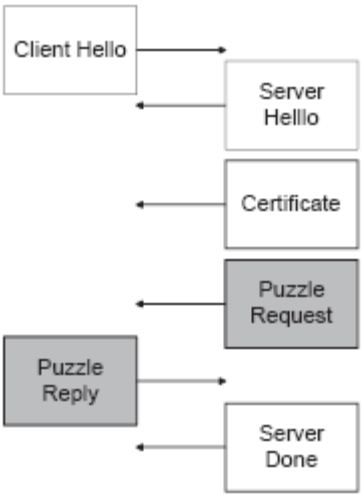

Figure 1. The TLS handshake protocol. The shaded portions are our additions.

The TLS specification specifies that unknown (to a particular implementation) record type shall be ignored. Therefore, we use a new record type for the puzzle messages. This allows us to we remain backwards compatible with old TLS implementations that do not support puzzles. Though such implementations may time out a connection if they do not reply to a puzzle, they will not notice any protocol violations. This technique is only applicable to TLS and does not work for SSLv3 as SSLv3 does not discard unknown record types. When the server is not under attack, no changes in the TLS protocol are required. In order to prevent the denial-of-service attack against TLS, we need to add a new message after the Server Hello message and before the Server Done message. See the shaded portions of Figure 1. This message contains a cryptographic puzzle and is only sent when the server is under load. The server will then wait on a response message before continuing with the handshake protocol.

A. The Client Puzzles:

To be useful as a client puzzle, a puzzle needs to be solvable in a predictable amount of time. The puzzle generally should not take too long to solve (*e.g.,* no more than a second or so on a relatively slow machine), but at the same time, there should be no known shortcut to solving the puzzle. In addition, the server needs to be able to generate puzzles while doing much less work than the client solving them. Of course, the server also needs an efficient method of verifying the correctness of a proposed solution.

B. Puzzle Construction:

In this section we present our main Diffie-Hellman based puzzle construction scheme, which will be the construction of choice for the rest of the paper. We begin by enumerating the goals we would like our puzzle construction to meet. Next, we present our D-H based construction along with an identity-based variant. Finally, we present two other puzzle constructions that prove interesting to examine. Lack of space forbids our including formal definitions and security proofs here; thus what is presented are construction sketches only and heuristic hardness claims. This is not to discount the importance of a formal model. On the contrary, formal definitions for puzzle hardness [8] are only incipient in the literature and would naturally require extension to the outsourcing scenario as a prerequisite for security analysis. This is beyond the scope of our present investigation.

Let us introduce some notation. Let $f_k : \{0,1\}^* \rightarrow \{0,1\}^k$ be a one-way hash function whose range consists of k-bit strings. It is convenient to model f as a random oracle. The value k is a security parameter; we drop this superscript where appropriate for visual clarity. A parameter *l*

ISSN 1512-1232
serves to govern the hardness of the puzzle constructions we describe. For a channel c, a timeslot τ, and a defending server, I, let $\Pi_{I,c,\tau}$ denote a published and authenticated puzzle. Let $\sigma_{I,c,\tau}$ denote the corresponding solution (which we assume to be unique).

We let y_I denote the public key associated with a particular defending server I, while x_I denotes the corresponding private key; we let y and x be the respective keys of the bastion. We omit the subscript I where context makes it clear.

C. Goals for this scheme

Puzzle outsourcing for our purposes introduces a new set of constraints and requirements.

We enumerate the most important of these here:

1. Unique puzzle solutions: The practicality of our solution depends on the ability of a defending server to precompute puzzle solutions prior to their associated timeslot, and subsequently to check their correctness via table lookup. Consequently, it is important that puzzles have unique solutions (or a very small number of correct ones).

2. Per-channel puzzle distribution: We want the bastion to be able to compute and disseminate puzzle information on a per-channel basis. In other words, the bastion should be able to publish information for a particular channel number c that may be used to deduce the corresponding puzzle for any defending server. (Different servers should have different puzzle solutions, though, so that one server's ability to enumerate its own puzzle solutions does not expose other servers to attack.) With this property, the bastion does not even need to know which servers it is helping to defend. This reduces the amount of information the bastion must compute and publish, and it removes the need for explicit relationships or coordination between defending servers and bastions.

3. Per-channel puzzle solution: Another desirable property is for the work done by a client to apply on a per-channel basis, rather than a per-puzzle basis. In particular, we would like a client that has solved a puzzle for a particular channel to be able to efficiently compute the token for the same channel number on any server. As we have already noted, this does not mean that tokens should be identical across servers—only that there should be considerable overlap in the brute-force computation needed to solve the puzzle for a given channel-number across servers. In particular, it is not desirable for one server to be able to use its shortcut to compute the tokens associated with another server, as this would result in a diffusion of trust across all participating servers rather than in the bastion alone. The per-channel puzzle solution property is useful because it allows a client to begin solving puzzles before deciding which server to visit.

4. Random-beacon property: Sometimes it is possible to achieve a property even stronger than per channel puzzle distribution. Ideally, puzzles might not require explicit calculation and publication by a bastion. Instead they might be derived from the emissions of a random beacon. We use the term random beacon to refer to a data source that is: (1) unpredictable, i.e., dependent on a fresh source of randomness; (2) highly robust, i.e., not subject to manipulation or disruption; and (3) easily accessible on the Internet. A puzzle construction based on a random beacon would eliminate the need for explicit bastion services. (Apart from the architectural advantages, this could have the benefit in some circumstances of eliminating any point of legal liability for reliable puzzle distribution.) Hashes of financial market data or even of Internet news sources, which both can be obtained from numerous locations, would be candidate random beacons. Surprisingly, under this construction not only would the bastion (random beacon) not have to know what defending servers were relying on its services, but in f act it wouldn't even need to know its data was being used to construct puzzles!

5. Identity-based key distribution: When puzzles are based on the public key of a defending server, the public key itself must be distributed via a robust directory. A desirable alternative is identity-based distribution, wherein the public-key of a particular defending server can be derived from the server name and a master key known to all defending servers. This is closely analogous to the well-known primitive of identity-based encryption [9].

6. Forward security: A final desirable property is forward security. Specifically, that time-limited passive compromise of a bastion should not undermine the DoS protection it confers.

ISSN 1512-1232

D. A DH based construction

We now describe a puzzle construction based on Diffie- Hellman key agreement [10]. It has all of the properties above except the random-beacon property (i.e., it has properties 1,2,3,5 and 6).

Let G be a group of (prime) order q. Let g be a published generator for the group and *l* be a parameter denoting the hardness of puzzles for this construction. (As explained below, we require a strong, generic-group assumption on G.) We propose a simple solution in which the bastion selects a random integer $r_{c,\tau} \in_R Z_q$ and a second random integer $a_{c,\tau} \in_R [r_{c,\tau}, (r_{c,\tau} + l) \bmod q]$. (Recall that l is the hardness parameter for the puzzle.) Let f 'in this case be a one-way permutation on Z_q, and let $g_{c,\tau} = g^{f'(a\,c,\tau).}$ The intuition is as follows. The value $g_{c,\tau}$ may be viewed as an ephemeral Diffie-Hellman public key. A puzzle solution for defending server I is the D-H key that derives from its public key $y_I = g^{xI}$ (xI is the secret key) and the ephemeral key $g_{c,\tau}$. Solving a puzzle means solving the associated D-H problem. To render the problem tractable via brute force, the bastion specifies a small range $[r_{c,\tau}, (r_{c,\tau} + l) \bmod q]$ of possible seed values for its ephemeral key. In other words, the bastion publishes $\Pi_{I,c,\tau} = (g_{c,\tau}, r_{c,\tau})$. For a client (or attacker) to solve the puzzle requires brute force testing of all of the seed values. In particular, for a given candidate value a', the client tests whether $g_{c,\tau} = gf'(a')$. For a particular defending server I, the solution to the puzzle is $\sigma_I = y_I^{f'(a\,c,\tau)}$. Of course, a defending server can use its private key x_I as a shortcut to the solution of the puzzle. The defending server can compute $\sigma_I = y_I^{f'(a\,c,\tau)} = g_{c,\tau}^{x_I}$. In other words, it essentially computes a Diffie-Hellman key. For a defending server, solution of a puzzle essentially requires just one modular exponentiation.

On average, puzzle solution by a client (or attacker) requires *l*/2 modular exponentiations over G. Since puzzle hardness needs to be precisely characterized, we believe that any concrete computational hardness claim would have to depend on a random-oracle assumption on f' and also a generic-model assumption for the underlying group G [11]. Thus it is important to choose G appropriately. (Several common types of algebraic groups are believed to have the ideal properties associated with the generic model, e.g., most elliptic curves and the order-q subgroup G of the multiplicative group Z^*_p, where p = kq + 1 for small k [11].)

We summarize the client and server operations as follows.

Client

• During period Ti, downloads random puzzles from the bastion service and solves them with spare computational resources.

• During time period Ti+1, uses the solutions that were solved during the previous period Ti.

• When initiating a request from a certain server, the client machine checks to see if the server has a public key for DoS prevention. If so, the client combines its puzzle solution and the server's public key to make a token for a particular channel on the server. The token is appended to the request.

• If the client has multiple puzzle solutions for multiple channels and one is not working on a particular server, the client may retry the request using a different token for a different channel.

• A client that has just booted up and stated solving puzzles may have to wait up to an entire time period before it has a solution that can be used. However, once the client is in the cycle of solving puzzles it will always have a valid solution.

Server

• During time period Ti, downloads all the puzzles for the channels and computes a token list from them using its private key. The list is used during the next period, Ti+1.

• If the system load is low and there is no DoS attack, then the server ignores the tokens and processes requests as though there were no DoS prevention system.

• During an attack the server only accepts requests that have valid tokens for solutions. The request token for a particular channel is quickly checked against the table of valid tokens. The amount of resources granted will be limited on a per channel basis.

ISSN 1512-1232

V. Implementation

In order to measure the effectiveness of our solution, we modified the OpenSSL library to support servers and clients that understood our puzzle protocol. On the server side, hooks were added to the mod_ssl Apache module and as a client the TLS enhanced version of lynx was used.

A. The OpenSSL Library

OpenSSL is an open-source library that includes support for the SSL and TLS protocols as well as the underlying cryptographic operations needed by SSL and TLS. OpenSSL handles connections on a per-socket basis and does not keep any global process state. This prevents a clean separation between our modified OpenSSL library and server applications because we need to measure the global server load. On the client side however, the application never needs to be aware whether the puzzle protocol took place. Clients can trivially support the protocol just by relinking with the modified library. In OpenSSL, the TLS handshake is implemented as a state machine representing the current location in the protocol. To add support for puzzles on the server, a new state was added after the server certificate request state. In this state the server either sends a puzzle request and switches to a state waiting to receive the puzzle reply or immediately switches to the server done state. The puzzle reply state will wait to receive a puzzle solution before switching to the server done state. If a puzzle solution is never received the connection will time out. On the client side, we treat the reception of a puzzle as an "unexpected event", in the incoming message handler because the client is expecting a handshake record. The puzzle solution is then computed and returned to the server before the handshake processing continues. The biggest challenge is deciding whether a server should send a client puzzle. Because OpenSSL has no notion of application or system wide state, it has no way to count the number of RSA operations a server has committed to. To remedy this problem, we provide callbacks to alert the application whenever we commit to or finish an RSA private decryption. We also add a callback that allows the server to decide whether to send a client puzzle on the current connection, and if so, how many bits the puzzle should be. This control flow is shown in Figure 2.

B. Performance without Client Puzzles

Using one client and less than 550Kbps of traffic, we were able to completely load the server. The number of pending RSA operations was continually increasing for the first 100 TLS connections made to the server during a simulated attack. At this point, there were no more Apache processes available to handle additional requests, so the number of pending requests falls as RSA operations complete with no new operations being committed to, as clients are unable to make new connections to the server. Figure 3 shows the latency experienced by a legitimate user trying to connect to the server during this period during a representative benchmarked run. By using two attacking computers, we were able to double the latency experienced by the legitimate user. These simulated attacks can be continued indefinitely by the attacking computers. These results show that an unprotected TLS server is indeed vulnerable to these attacks.

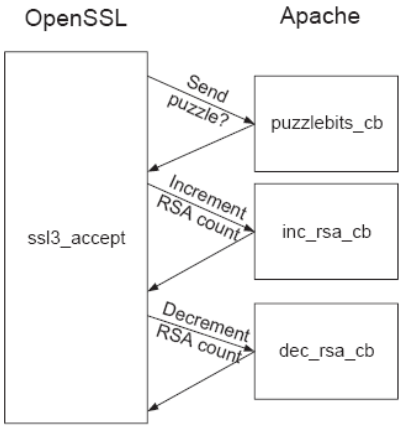

Figure 2: Control flow in OpenSSL with client puzzles

ISSN 1512-1232

VI. Conclusion

Client puzzles are an effective means of countering a denial-of-service attack against TLS servers. We showed how the robustness of authentication protocols against denial of service attacks can be improved by asking the client to commit its computational resources to the protocol run before the server allocates its memory and processing time. The server sends to the client a puzzle whose solution requires a brute-force search for some bits of the inverse of a one-way hash function. The difficulty of the puzzle is parameterized according to the server load. The server stores the protocol state and computes expensive public-key operations only after it has verified the client's solution. The puzzles protect servers that authenticate their clients against resource exhaustion attacks during the first messages of the connection opening before the client has been reliably authenticated.

References

1. D. Dean and A. Stuile field "Using Client Puzzles to Protect TLS" In Proceedings of the 10th USENIX Security Symposium August 2001.
2. Brent Waters, Ari Juels, J. Alex Halderman, and Edward W. Felten, New Client Puzzle Outsourcing Techniques for DoS Resistance.
3. Tuomas Aura, Pekka Nikander , and Jussipekka Leiwo DOS-resistant Authentication with Client Puzzles
4. Timothy J. McNevin, Jung-Min Park, and Randy Marchanyt Chained Puzzles: A Novel Framework for IP-Layer Client Puzzles 2005 International Conference on Wireless Networks, Communications and Mobile Computing
5. W. Feng, E. Kaiser, W. Feng, A. Lul, 'The Design and Implementation of Network Puzzles," in Proceedings of INFOCOM 2005, March 2005.
6. R K. C. Chang 'Defending against flooding –based distributed denial of- service attacks: a tutorial," in EEE Communications Magazine. Volume 40, Issue 10. October2002. Pages 42-51.
7. X Wang and M. K. Reiter. 'Mitigating Bandwidth-Exhaustion Attacks using Congestion Puzzles (Extended Abstract)," in Proceedings of the 11th ACM Conference on Computer and Communications security (CCS '04). October 25-29 , 2004.
8. M. Jakobsson and A. Juels. Proofs of work and bread pudding protocols. In Communications and Multimedia Security, pages 258–272. Kluwer Academic, 1999.
9. D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. SIAM J. of Computing, 32(3):586–615, 2003.
10. W. Diffie and M.E. Hellman. New directions in cryptography. IEEE Transactions on Information Theory, 22:644–654, 1976.
11. C.-P. Schnorr and M. Jakobsson. Security of discrete log cryptosystems in the random oracle and generic model. In The Mathematics of Public-Key Cryptography. The Fields Institute, 1999.

Article received: 2010-12-21