Adaptive Threshold Puzzles

Valer Bocan, *Member, IEEE* and Mihai Făgădar-Cosma

***Abstract* — Threshold puzzles represent an improved version of client puzzles, which have been proposed to add DoS resistance to authentication protocols. They involve the time management of solved puzzle instances, thus making the protocol resistant to strong attacks. This paper addresses the need of adapting the puzzle complexity to the computational power of the client and introduces a new concept: the adaptive threshold puzzles.**

***Keywords* — Denial of Service, Attack, Authentication, Client puzzles, Adaptive threshold puzzles.**

I. Introduction

Denial of service attacks are a major problem in today's interconnected world. Attackers are known to exploit the end-user ignorance and break into hundreds of thousands of systems to install their tool of choice. These "zombie" systems are capable of receiving commands from a central operations center via encrypted channels and the main reason for their existence is to generate bogus traffic targeted towards a specific website. In order to make tracking more difficult, the source IP address may be spoofed but in the same time it may be chosen from the same subnet in order to avoid egress filtering [1].

In order to add DoS-resistance to any authentication protocol, the design principle should be that the client always commits its resources before the server does and at any point during protocol execution, the cost for the client should be greater than that for the server. The client cost may be increased artificially by asking it to do some work whose difficulty may be effortlessly chosen by the server. At the same time, the verification for correctness should not place a burden on the server since that would defeat the very purpose of the technique.

Although the idea of using cryptographic puzzles for key agreement appeared in early 1980s [2], client puzzles started to be used as means of enforcing authentication protocols only recently, after a series of DoS attacks produced significant financial losses for a list of major websites, including Yahoo!, Amazon and eBay [3]. Currently, client puzzles are used for authentication protocols in general [4], to prevent TCP SYN flooding [5] and as a regulating measure against junk mail [6]. Also, time-locked cryptography was addressed [7], but its inherent sequential nature makes it very difficult for the server to verify the solution.

V. Bocan is a Ph.D. student with the Department of Computer Science and Engineering, "Politehnica" University of Timisoara, Romania (phone: +40 722 714798; e-mail: vbocan@dataman.ro).

M. Fagadar-Cosma is with the Digital Signal Processing Laboratories, Department of Computer Science and Engineering, "Politehnica" University of Timisoara, Romania (phone: +40 721 000499; e-mail: mfagadar@yahoo.com).

II. Client Puzzles

Before committing resources the server should ask the client to solve a problem, as seen in Fig. 1. Regardless of the specific implementation, a good puzzle should have the following properties [4], the last of which being new:

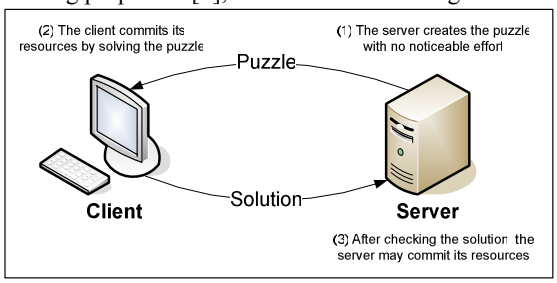

Fig. 1. Principle of the client puzzle protocol

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 precompute 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.

The natural choice for a client puzzle is the brute force reversal of hash functions such as MD5 or SHA1 since they have a simple structure and can run on a variety of hardware platforms. The use of a reduced round cipher instead of the hash function has also been proposed [5] but that is beyond the scope of this paper.

A. Creating a new puzzle

Periodically (once every few minutes), the server generates a random value N_S. In order to prevent attacks by guessing the nonce, the value should have 64 bits of entropy and should not be a predictable value such as a time stamp. This entropy should be enough to prevent an

attacker to precompute <nonce-result> pairs and the occasional matches caused by birthday attacks would not do too much harm here. The server has to decide the difficulty level k of the puzzle, based on the current conditions. To sum up, the puzzle that is broadcast to clients is the pair:

$$< N_S, k > . \quad (1)$$

B. Solving the puzzle

To solve the puzzle, the client C generates a nonce N_C. The purpose of this nonce is twofold. First, if the client reuses a server nonce N_S, it can create a new instance by generating a new N_C. Second, without the client nonce an attacker could compute the puzzle and send the result back to the server before the client does. 24 bits of entropy should be enough to prevent the attacker from exhausting the values of N_C given that N_S changes frequently.

The client must repeatedly apply a hash function h to a quantity X and the puzzle is considered solved when the first k bits of the result Y are equal to 0, as shown in equation (2).

$$h(C, N_S, N_C, X) = Y. \quad (2)$$

Since the server changes N_S periodically, while it considers N_S recent, it must keep a list of correctly solved instances in the form of N_S–N_C pairs so that previous solutions cannot be reused.

Since there are no known shortcuts to find out X, the only possibility is to search for it by brute-force. The difficulty level k (i.e. the number of zeros at the beginning of Y) dictates how long the puzzle will take to solve. If k equals 0 then no work is required, whereas if k equals 128 (for MD5) or 192 (for SHA), the client must reverse an entire one-way function which is computationally impossible.

C. Puzzle difficulty

The parameter k represents the puzzle difficulty. The task of establishing it at the time of puzzle generation is rather tricky, since there is no obvious metric that one can use in a real-world implementation. The best approach would be the number of already committed RSA operations rather than the current processor load or the number of incoming requests [1]. 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 operations. Literature data [4] shows that reasonable values for k are between 0 and 64. By experimenting, we have found out that the reasonable range is much narrower and for small difficulty levels, the time needed to solve the puzzle for level k may be greater than the time for level $k+1$.

As of today, the average web client is capable of approximately 4500 – 5000 MIPS leading to 0.02 ms per cryptographic operation. Thus, the puzzle difficulty curve looks as in Fig. 2. For difficulty levels above 20, the time needed to solve the puzzle is prohibitive, hence the limited practical applicability. A cryptographic operation is considered an attempt (not necessarily successful) to solve the puzzle and includes the time needed to build up the quantity X and the actual computation of either an MD5 or a SHA function.

In order to obtain a more accurate scale for the puzzle difficulty parameter, the puzzles can be split into several smaller puzzles of equal difficulty [5]. These smaller puzzles can be solved separately and the general result will be a combination of the individual results. Literature studies mention that the same granularity can be achieved by combining sub-puzzles of varying difficulty, at a slightly lower cost for the server [4], but that is yet to be confirmed by experiment.

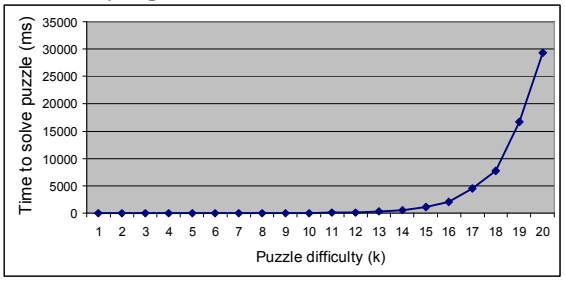

Fig. 2. Solving time for different puzzle difficulties

III. Threshold Puzzles

Client puzzles have proved effective both in theory and in practice. They are secure and perform well in most scenarios. Regardless of the particular client being serviced, the puzzle difficulty is chosen based on a metric that refers strictly to the server resource commitment. Since puzzles may be broadcast and are generated at precise intervals, this "one size fits all" solution is not perfect since different clients have various computing powers. We have noted that client puzzles are vulnerable to a particular form of attack (called henceforth "strong attack") due to the highly parallel nature of the puzzle. A strong attack is defined as a denial of service attack mounted by an attacker with access to massive computing power. The attacker is able to solve puzzles in a time much shorter than a legitimate client. The schematic of a strong attack is shown in Fig. 3.

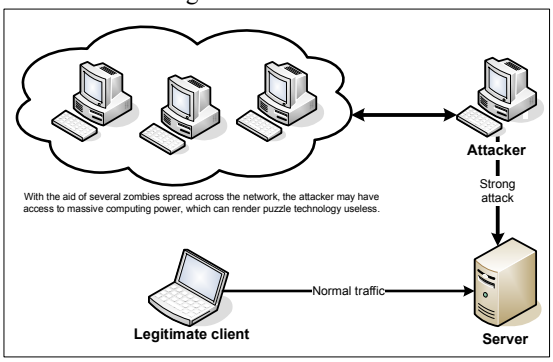

Fig. 3. Schematic of a strong attack on an authentication protocol protected by client puzzles

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 such a difficult puzzle.

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. There are many real-world examples of how hundreds of thousands of computers are put to work together for a common purpose, like the SETI program [8] and the effort to break the RSA algorithms [9], so strong attacks are definitely possible.

Threshold puzzles [10], [11] address the strong attack issue in two ways: first, by limiting the difficulty level so that the puzzle remains within usability margins and second, by adding a minimum response time to the puzzle definition.

A. Limiting the puzzle difficulty level

Although the current design of the client puzzle as it is described in [4] specifies a difficulty range from 0 (no work required) to 128 or 192 (impossible, depending on the hash function used), a real-world implementation of an authentication protocol is likely to choose a reasonable range for the puzzle difficulty, say between 0 and 25, due to the exponential scale which gives a narrow usability margin. Having difficulty levels close to impossible may open a new avenue of attack against the legitimate clients themselves and this is an issue even more serious than attacking just the server.

B. Establishing the minimum response time

The basic idea is to add the timestamp at which the server nonce was generated to the list $<N_S, N_C, X, k>$ 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. If it is, then the server is under a strong attack and should immediately cease communication with the client in question. On average it takes $2^k - 1$ operations to solve a puzzle of difficulty k, as shown in equation (3):

$$T_{estimated} = (2^k - 1) \cdot T_{operation} . \quad (3)$$

$T_{operation}$ represents the minimum time for performing a cryptographic operation (currently in the range 0.01-0.02 ms) and must be determined experimentally or by using Moore's law. Thus, the estimated time represents the acceptance threshold for the client puzzle.

IV. Adaptive Threshold Puzzles

Although threshold puzzles add more resistance to authentication protocols in front of strong attacks, they are not adapted to the computational power of the client. In order to solve a puzzle of a given complexity k, a "light" client such as a PDA or a laptop computer, will take considerably more time than a "heavy" one, like a PC workstation or a cluster of personal computers. Therefore, the puzzle difficulty level must be adjusted, so that it matches the computational capabilities of the clients. This is where we further refine the threshold puzzle concept, by introducing the **adaptive threshold puzzles**.

Upon request of the client, who may claim that the default puzzle is too difficult to solve, in order to determine the computational power of the claimant, the server will send a **probe puzzle**, containing a problem which must be solved by the client. This problem may still consist in the brute-force partial inversion of a dispersion function (much like a regular client puzzle) or it may require a different algorithm. Its difficulty and solving time should exhibit a linear dependence. In order to fulfill this requirement, a linearization algorithm can be employed [12], which gives an acceptable cvasi-linear dependence between problem complexity and solving time.

Based on the time it takes the client to solve the probe puzzle, the server can determine its computational power, P_C. To overcome the situation in which a malicious client deliberately solves the probe puzzle in a longer time in order to conceal its true P_C value, the client must be encouraged to use its full available power during probe puzzle solving. One solution may be to grant a number of connections per time unit proportional to the reported P_C. For instance, if a powerful client is allocated a maximum number of N connections per time unit, a less powerful client (such as a PDA) will be allocated only $N/2$ or $N/3$ connections per time unit, given that the latter is 2 or 3 times less powerful than the former. If a malicious client has reported a lower P_C than it actually has, it will be allocated a lower number of accepted connections per time unit. If the client exceeds this number, the exceeding connections will be dropped by the server, so that the server will not be affected by an alleged attack from this client.

After the server learns the computational power P_C of each of the client, the threshold puzzle mechanism is employed, however for each client the complexity k is adjusted. Assuming that the server wishes to set the puzzle difficulty to k for clients with an average computational power of $P_{reference}$, for a client with the power P_C, based on equation (3), the difficulty level will be set to a value k_C, as shown in equation (4):

$$k_C = round\left(k \cdot \log_2\left(\frac{P_C}{P_{reference}} \right) \right). \quad (4)$$

V. DoS-Resistant Authentication using Adaptive Threshold Puzzles

Client puzzles and threshold puzzles have been used to add DoS-resistance to authentication protocols [4], [10]–[13]. Using adaptive threshold puzzles, the protocol is subject to further changes, due to the introduction of the probe puzzle, which is employed by the server to determine the computational power of each client.

Let us assume that, at a given moment in time, the server decides to establish the difficulty level for its threshold puzzles to k. This value is chosen to suit a large variety of

clients, with an average computational power given by $P_{reference}$.

The protocol begins with a new client C requesting a connection to the server, in the form of a **ClientHello** message. The client may elect to solve the default puzzle provided by the server through broadcast messages, or it may choose to request to be probed for computational power, in order to be granted easier puzzles for subsequent connections. Should that be the case, we propose a new step to the normal threshold puzzles protocol [10], in which the server will create a probe puzzle, which is sent to the client in a time-stamped **ProbeRequest** message. The client solves the probe puzzle and answers with a time-stamped **ProbeResponse** message, containing the solution to the probe puzzle. Based on the time it takes the client to solve the puzzle, the server will determine its computational power (P_C) and use it to estimate the difficulty level k_C for the new client, according to equation (4).

The normal threshold puzzle protocol can be used subsequently, with one noted difference that new puzzles for client C are generated as $<N_S, k_C>$ pairs instead of $<N_S, k>$ pairs. These puzzles are wrapped by **PuzzleRequest** messages sent by the server to its clients. As a protection method against malicious clients, the server must also limit the number of accepted connections per time unit for each individual client, based on its reported computational power.

Any client willing to talk to the server has to generate a random nonce N_C and must correctly solve the threshold puzzle contained in the **PuzzleRequest** message and supply the C, N_C and X parameters for verification, inside a **PuzzleResponse** message. In case it wants to initiate several connections to the same server, the client may reuse the puzzle by generating a new N_C.

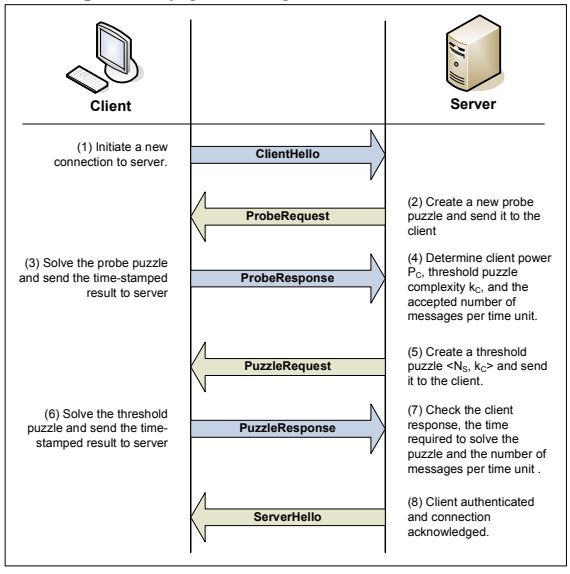

Fig. 4. Schematic of an authentication protocol protected by adaptive threshold puzzles

Upon receipt of a solved threshold puzzle, the server checks whether the client C has already submitted a solution with the same N_S and N_C. This check ensures that solutions are not replayed. The server also checks whether the puzzle was solved in a time shorter than $T_{estimate}$, and if the client exceeded its allocated number of connections per time unit. In both cases, the server considers itself under attack and drops the connection to the client in question, without committing any resources. If the time exceeds the estimate and the number of messages is within the established limit, the server will proceed with calculating the hash, verify the signature, acknowledge the connection with a **ServerHello** message, and continue with the normal protocol execution as shown in Fig. 4.

VI. Conclusions and Future Work

Client puzzles have traditionally been the solution of choice for active defense against denial of service attacks. The original design of the puzzle was augmented in the form of the threshold puzzle and because the one-size-fits-all scenarios does not work perfectly in all cases, we further refined the concept in order to take into account the real processing power of the client. Adaptive threshold puzzles adapt the difficulty of the problem with respect to each client being serviced, enabling more judicious allocation of server resources.

Experimental results based on the theory presented herein will make the subject of a future article.

References

[1] D. Dean and A. Stubblefield, "Using Client Puzzles to Protect TLS", in Proc. 10th Annu. USENIX Security Symposium, Washington, 2001.

[2] R. C. Merkle, "Secure Communications Over Insecure Channels", *Communications of the ACM*, vol. 21, no. 4, pp. 294-299, Apr. 1978.

[3] Computer Emergency Response Team, 2000, *CERT Advisory CA–2000.01 Denial of service developments*, Available: http://www.cert.org/advisories/CA-2000-01.html.

[4] T. Aura, P. Nikander and J. Leiwo, "DOS-resistant Authentication with Client Puzzles", in *Proc. 8th Int. Workshop on Security Protocols*, Cambridge, UK, 2000.

[5] A. Juels and J. Brainard, "Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks", in *Proc. of NDSS*, San Diego, 1999, pp. 151-165.

[6] C. Dwork and M. Naor, "Pricing via Processing or Combating Junk Mail", in *Proc. CRYPTO '92*, Springer Verlag, 1992.

[7] R. R. Rivest, A. Shamir and D. A. Wagner, 1996, *Time-lock Puzzles and Timed-release Cryptography*, Available: http://lcs.mit.edu/~rivest/RivestShamirWagner-timelock.pdf.

[8] SETI @home Program, Available: http://setiathome.ssl.berkeley.edu/.

[9] The Distributed.net Organization, Available: http://www.distributed.net.

[10] V. Bocan, "Threshold Puzzles: The Evolution of DoS-resistant Authentication", *Periodica Politehnica*, *Transactions on Automatic Control and Computer Science*, vol. 49, no. 63, 2004.

[11] V. Bocan, "Single Sign-On Systems under Denial of Service Attacks", PhD. report #3, Dept. of Computer Science and Eng., "Politehnica" Univ. of Timisoara, Timisoara, Romania, 2004.

[12] V. Bocan, "A Study on the Security Level Provided by Authentication Protocols",PhD. report #2, Dept. of Computer Science and Eng., "Politehnica" Univ. of Timisoara, Timisoara, Romania, 2004.

[13] Mentalis C# Security Library, Available: http://www.mentalis.org.