Denial of Service Resilience of Authentication Systems

Valer BOCAN, Mihai FAGADAR-COSMA
Alcatel-Lucent Romania

ABSTRACT

Cryptographic authentication systems are currently the *de facto* standard for securing clients access to network services. Although they offer enhanced security for the parties involved in the communication process, they still have a vulnerable point represented by their susceptibility to denial of service (DoS) attacks. The present chapter addresses two important aspects related to the security of authentication systems and their resistance against strong DoS attacks, represented by attack detection and attack prevention. In this respect we present a detailed analysis of the methods used to evaluate the attack state of an authentication system as well as of the countermeasures that can be deployed to prevent or repel a DoS attack.

INTRODUCTION

Denial of service attacks on authentication systems can take two possible forms. On one hand, an attacker can prevent the network from sending the messages that it should normally transmit to its clients. On the other hand, it could force the network into sending messages it should not normally transmit. By far, the most popular DoS attack is server flooding that prevents legitimate clients from obtaining the services they request from that server.

One cause for the vulnerability to DoS in authentication systems is that the dialog between peers takes place before even a minimum pre-authentication is performed, which renders the server incapable of distinguishing legitimate from malicious traffic. Enforcing the authentication of all requests would represent a DoS attack by itself, since the server would be busy checking all digital signatures, no matter if these are valid or not. Such a method would be as dangerous as a TCP stack overflow is in case of TCP SYN attacks.

Another vulnerability is the lack of resource accounting. In this respect Spatscheck and Peterson (1999) consider that there are 3 key ingredients for protecting against DoS attacks: accounting all resources allocated to a client, detecting the moment when these resources rise above a predefined threshold and constraining the allocated resources by reducing them to a minimum level in case an attack has been detected and recovering the blocked resources.

The third vulnerability resides in the intrinsic design of the communication protocols, as described by Crosby and Wallach (2003). A new class of low-bandwidth attacks exploits the deficiencies of data structures employed in various applications. For example, hash tables and binary trees can degenerate into simple linked lists when input data is selected accordingly. Using the typical bandwidth of a dial-up modem, the authors have managed to bring a Bro server on the edge of collapsing: 6 minutes after the attack has begun, the server was ignoring 71% of traffic and was consuming its entire computational power.

Taking in consideration the global market tendency towards on-line availability, DoS attacks prove to be more dangerous than initially predicted therefore identifying them as soon as they take place is a decisive aspect. From the moment the attack has begun until it is detected and countermeasures are deployed, the targeted servers are blocked and all legitimate requests are ignored, which can result in significant financial losses. Chained attacks can occur if the communication protocol continues its dialogue with the attacker even after anomalies have been detected. The basic idea behind the so called fail-safe or fail-stop protocols is for the message-exchange to be discontinued with any client that does not follow the normal course of the protocol.

Considering the attack forms and characteristics described above, a resilient authentication system must fulfill two main requirements. First, the system must be able to detect an incoming attack as soon as possible in order to be able to respond accordingly and prevent any possible losses. Second, the system must be able to defend itself against an ongoing attack, either through its intrinsic characteristics or by deploying a set of countermeasures against the attacker. Given these requirements, we have structured this chapter into two main parts. In the first part we address the strategy and the techniques that enable an authentication system to efficiently detect DoS attacks, and their implementation into a detection engine called SSO-SENSE. In the second part we focus on the threshold puzzles concept as an a efficient way to protect against DoS attacks and analyze the case study of the SSL Handshake algorithm from both an implementation and a performance perspective.

BACKGROUND

The Client Puzzles Concept

An efficient measure for preventing DoS attacks during the authentication phase would be to ensure that the client allocates its resources proportionally with the resources allocated by the server. As a result, at any time during the execution of the authentication protocol, the computational cost for the client will be higher than that of the server. This can be achieved by asking the client to solve a puzzle with a difficulty established by the server. The solution to the client puzzle should be easily accessible to the server, in order to obtain a low resource usage, while the client should be forced to allocate computational resources into solving the puzzle according to the complexity requested by the server. Merkle (1978) was the first to come up with the idea of using cryptographic puzzles, but he applied the concept only for key exchange and not for the authentication itself. Later, the client puzzles concept has been successfully applied against TCP SYN attacks by Juels and Brainard (1999), who also outline the vulnerability of SSL protocols against DoS attacks and provide a rigorous demonstration of their security characteristics. Aura, Nikander and Leiwo (2000) have applied the puzzles to authentication protocols in general while Dwork and Naor (1992) have proposed measures for regulating unsolicited messages within a protocol.
According to Aura, Nikander and Leiwo (2000) and to Harris (2001), the client puzzle must have a set of well-defined properties, as follows:

- Creating a puzzle and verifying the solution must not require significant resources on the server side;
- The cost of solving the puzzle should easily be changed from 0 to infinity;
- The puzzle should be solvable on most hardware platforms;
- Pre-calculating the solution to the puzzle must be impossible;
- While the client solves the puzzle, the server must not store the solution or other client-specific information; the same puzzle can be distributed to several clients, knowing that the solutions provided by one or more clients do not help in calculating a new solution;
- A client may reuse a puzzle by creating one or more of its instances.

The client puzzle concept is shown in *Figure 1*.

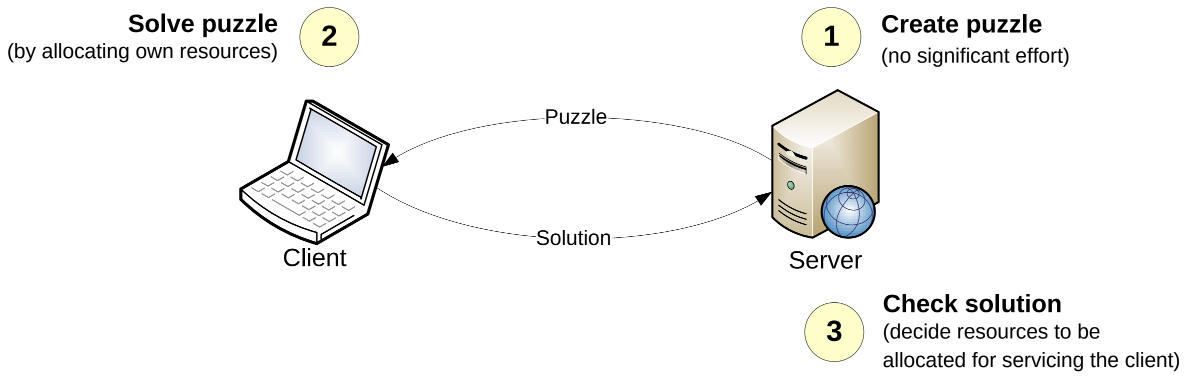

Figure 1. The client puzzle concept

Client puzzles can take various forms, the most popular ones being the partial inversion of a hash function (Juels & Brainard, 1999; Aura, Nikkander & Leiwo, 2000), the discrete logarithm inversion applied by Waters, Juels, Halderman and Felten (2004), the time-locked puzzles proposed by Rivest, Shamir and Wagner (1996) and linked puzzles (Groza & Petrica, 2006; Ma, 2006).

Creating and Solving a Puzzle

Periodically (usually once every several minutes) the server generates a random 64 bit value N_S, enough to prevent the client from guessing the puzzle result. The server also attaches a value k to the puzzle, representing the puzzle complexity. In short, the puzzle which is sent by the server to its clients can be represented as a <Ns, k> pair.
In order to solve the puzzle, the client generates a random value N_C, with a double purpose: first, the client can generate a new puzzle by reusing the value N_S provided by the server to create the N_C value and second, it prevents an attacker from calculating the puzzle and sending it to the server before the legitimate client. A 24 bit size should be enough for this value, given the fact that it changes very often.
The client has to apply repeatedly a hash function to a quantity Y, and the puzzle is considered solved when the 1st k bits of the quantity Y are equal to 0, according to the equation

$$h(C, N_S, N_C, X) = Y,$$

where:

- h represents the hash function (e.g. MD5 or SHA),
- N_S is the random value generated by the server,
- N_C is the random value generated by the client,
- C represents the client identity and
- X is the solution to be sent to the server.

The server also maintains a list of the recently used $<N_S, N_C>$ pairs in order to prevent them from being reused.
Since there is no known way of determining X other than by brute-force, the client will be forced to use its computational power to reach the solution. The value k controls the puzzle complexity and thus the time needed for the client to reach the solution. The edge cases are k=0 which means no effort at all and k=128 for MD5 or 192 for the SHA function means a puzzle nearly impossible to solve.

Shortcomings of the Classical Puzzle Concept

Client puzzles make a good choice for securing authentication protocols, as long as the attack is not distributed (Schneier, 2000). One shortcoming is the fact that the computational power available to the client is ignored, since the puzzle difficulty is established using a metric that takes in consideration only the server engagement. As a result, an attacker which has access to significant computational power can unleash a so-called strong attack against the server, and can solve puzzles faster than a legitimate client.
During a strong attack, the server gradually increases the puzzle complexity up to levels which are very difficult for normal clients, which represents a DoS attack in itself. If the attacker has access to multiple computers and can solve the puzzle using parallel computing, it can reduce the solving time considerably. There are many real-life examples of such applications in literature, the SETI @home Program (2010) and the effort to break the RSA algorithm supported by the Distributed.net Organization (2010) being only two of them.
These considerations emphasize the necessity of adopting a puzzle mechanism which enables the server to assess the computational power of its clients and adapt the puzzle complexity accordingly, also taking in consideration the overall security status of the system. This requirement is even more complex given the variety of computational devices that can connect to a network service and the fact that the service must respond within an acceptable timeframe to all its legitimate clients, while at the same time keeping any possible attackers at bay.

DOS ATTACK DETECTION IN AUTHENTICATION SYSTEMS

Measurable Characteristics

The first step in being able to effectively protect against a DoS attack is to detect its presence. Thus, it is mandatory to determine if the server load is caused by a temporary and transitory aspect, by a peak in the network traffic, or by an attack. Deploying countermeasures induces a penalty experienced by clients as a drop in performance, and this situation can become unpleasant as it appears more frequently.
Due to the complex nature of authentication systems and to the wide range of possible configurations into which they may be deployed, attack detection is an important challenge especially because of the lack of standard risk assessment metrics. In this section we propose three measurable characteristics that can be used as metrics in evaluating the system status and determining the threat level, and apply them as a case study to SSO systems based on the Liberty protocol suite (Liberty Alliance Project, 2010a, 2010b).

Protocol Discipline

The first aspect in this category is the order of protocol execution. The exchange of messages between server and clients must follow a certain order and any disturbance in this order can indicate a possible attack on the system. Given the example of the Liberty protocol suite, a Single Logout request should not be received by a service or identity provider if the Single Sign-On and Federation request has been previously received (Bocan & Fagadar, 2005b), as shown in *Figure 2*.

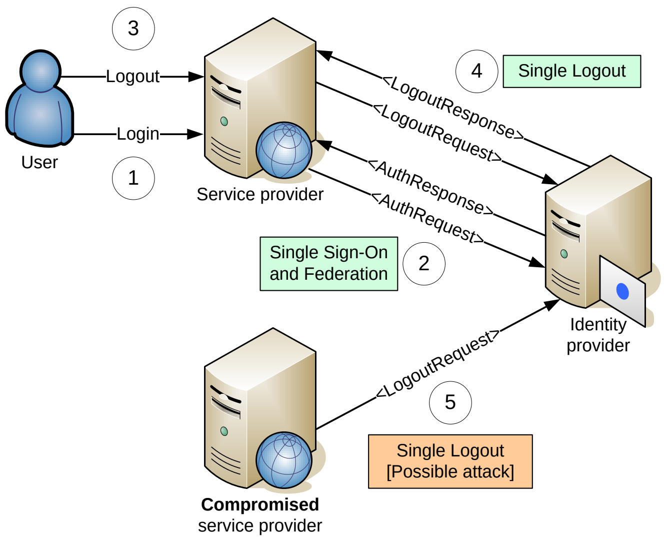

Figure 2. Detecting a possible DoS attack against a SSO authentication system by monitoring the protocol execution order

The request content is another characteristic to be considered. For each protocol, the request has a set of specific attributes which specify how the request should be processed by the receiving entity. A large variation of these attributes coming from a client may indicate a possible attack. For example, the AuthRequest message of the Single Sign-On and Federation protocol has an attribute which requests the identity provider to emit a temporary and anonymous identifier on behalf of the client, when data is exchanged among service providers. Too many requests of this kind received from the same client can indicate a DoS attack against the identity provider.

Network Health

Network health can be assessed by measuring traffic parameters on the server reception side and can be used to identify several aspects, such as:

- *network operation anomalies* represented by an abrupt increase in the measured values over a given time span;

- *flash crowd anomalies* due to the synergistic behavior of a group of users (e.g. when a new software product is available for download); this kind of anomaly can be detected in well-known points in the network such as download locations or mirror servers;
- *abuse anomalies* generated by ongoing DoS attacks or port scanning activities.

A software sensor can be placed in the system in order to monitor the anomalies and provide the relevant information - in raw or aggregated form - to the attack detection engine. According to several studies (Siaterlis & Maglaris, 2004; Kim, Lau, Chuah & Chao, 2004), reception parameters such as the time between two consecutive requests, request size, number of requests per second received from the same client can be used successfully to detect an ongoing DoS attack, but they must be incorporated into a more complex system for evaluating the health status of the authentication system.

System Health

System health is a crucial aspect which directly impacts the quality of service (QoS). If the system is functioning at nominal levels it can provide the QoS established by design. Altering these levels, either due to software or to hardware conditions, will lead to an immediate change in the QoS which will have as consequences delays in providing the response to the clients, intermittent responses or even a lack of response from the server.
System health can be characterized by several parameters, including processor load factor, available memory, frequency of I/O operations, resource reserve for virtual machines and the current state of system services, drivers or daemon processes.

Obtaining and Correlating the Threat Assessment Metrics

The information retrieved from the software sensors which measure the three main system characteristics presented above must be aggregated in order to reach a decision regarding the current threat level of the server.
The sensors which monitor the protocol discipline must be implemented directly into the SSO protocol, so that they have unrestricted access to the message exchange between peers. The sensors responsible of monitoring the network and system health can use the performance indicators built into the operating systems, such as *Simple Network Management Protocol* (SNMP), *Management Information Base* (MIB) indicators or *Windows Management Instrumentation* (WMI) specific to Microsoft Windows operating systems.
The threat level, at its simplest form, can be modeled as a real numerical value ranging between 0 and 1. A value of 0 indicates with the highest degree of certitude that the system is working properly and no attack is in progress, while a value of 1 represents a clear indication that the system is being affected by an ongoing attack which could lead to service failure. Any value in between can be considered as a degree of certitude associated to the detection of an ongoing attack.
As shown in one of our papers (Bocan & Fagadar, 2005b), Bayesian inference can be a good correlation mechanism for the information provided by the sensors, which generates as a direct output the probability that an attack has been identified, equal to the threat level.

CASE STUDY: THE SSO-SENSE RISK ASSESSMENT MODULE

System Overview

The SSO-SENSE is a specialized software module which relies on Bayesian inference theory in order to monitor network traffic and detect DoS attacks in Single Sign-On (SSO) environments. SSO-SENSE aims at providing better security for SSO systems, which are particularly vulnerable to DoS attacks since the protocols require every message to be signed.

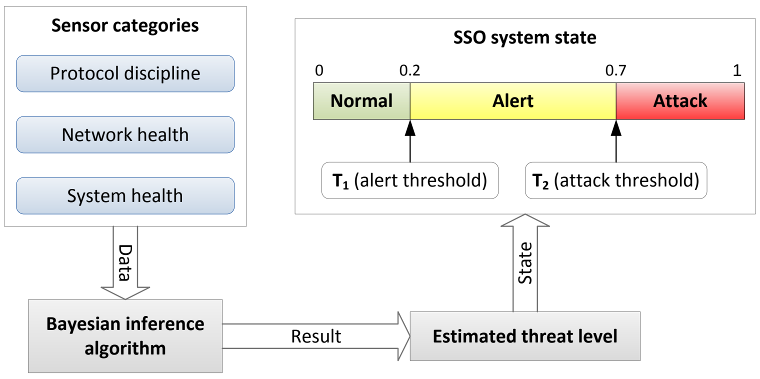

Figure 3. Block diagram of the SSO-SENSE module

As shown in *Figure 3*, SSO-SENSE gathers its input from a set of software sensors grouped across the three main categories mentioned earlier (protocol discipline, network health and system health) and applies Bayesian inference in order to estimate the threat level T of the system. Depending on this level, the system can be in one of the three states:

- *normal*, in which the system works within nominal parameters without being influenced by an attack;
- *alerted*, which indicates that the SSO system may be targeted by an attack and countermeasures like puzzle technology should be employed in order to reduce and control the rate of requests coming from the possible attacker;
- *attacked*, indicating a clearly identified DoS attack is in progress in which the server should forcibly close its connection with the attacker in order to save its resources for other legitimate clients.

In order to identify the current system state, SSO-SENSE compares the threat level with two thresholds, the alert threshold T_1 and the attack threshold T_2, with $0 < T_1 < T_2 < 1$, chosen in such a way as to efficiently delimit two adjacent states.

Evaluating the Threat Level using Bayesian Inference

When applying the Bayesian inference algorithm, SSO-SENSE considers a set of three mutually exclusive hypotheses H_1, H_2 and H_3, each associated with one of the system states. Initially, when system starts, a safe state is considered, for example $P(H_1) = 0.9$ for the normal state, $P(H_2) = 0.09$ for the alert state and $P(H_3) = 0.01$ for the attack state, where $P(H_i)$ represents the probability of hypothesis H_i being true.

When an event E is signaled by a software sensor which detects a cross-threshold condition within the measurable characteristics of the authentication system, that event will be used as evidence to compute the normalization factor Λ, as follows:

$$\Lambda = P(E \mid H_1) \cdot P(H_1) + P(E \mid H_2) \cdot P(H_2) + P(E \mid H_3) \cdot P(H_3)$$

where, according to the Bayesian inference theory, $P(E \mid H_i)$ represents the conditional probability of seeing the evidence E if hypothesis H_i is true.

Next, SSO-SENSE evaluates the posterior probability of each hypothesis in the light of evidence E, according to Bayes' theorem:

$$P(H_i \mid E) = P(E \mid H_i) \cdot P(H_i) / \Lambda.$$

In the end, the current threat level T is determined, according to the following formula:

$$T = \max[P(H_1 \mid E), P(H_2 \mid E), P(H_3 \mid E)].$$

The event E which triggers the threat evaluation can be classified into one of the following categories:

- A break into the authentication protocol, such as message inversion, message loss or messages with incomplete parameters;
- Time between two successive requests coming from the same client;
- An authentication error;
- Sudden increase in CPU load for a short period of time;

- Sudden increase in network traffic, except for flash crowd anomalies;
- Port scanning activities.

Applying Heuristic Attack Detection to the Authentication Process

The various methods used to protect against DoS attacks in authentication systems can be enhanced in terms of behavior and efficiency by using the additional information provided by an attack detection engine. While a complex intrusion detection engine applied globally to the whole network may help in a general manner, attack detection on the authentication vector addresses a specific issue. In this case, the attacker does not aim to break into the system and steal valuable information but instead his goal is to compromise the authentication and to prevent legitimate clients from accessing a network service, an aspect which is not covered by the existing intrusion detection systems.

The SSO-SENSE has been specifically designed for monitoring the authentication process, which represents the gate into the system. Let us consider the simple case of an SSO system, illustrated in *Figure 4*.

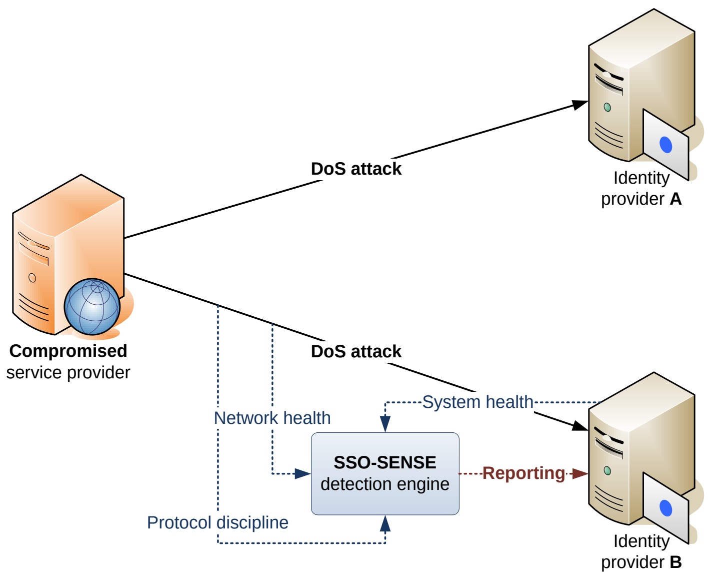

Figure 4. Heuristic attack detection on the authentication vector

A compromised service provider launches a DoS attack against two identity providers, which are both protected against attacks by their built-in client puzzle technology. The difference is that identity provider B benefits of the presence of the SSO-SENSE engine, which allows it to take efficient actions against the attacker with lesser impact on legitimate clients.

When using the classical client puzzle concept, the maximum puzzle difficulty k_{max} is calculated according to the formula:

$$k_{max} = \log_2(\max(1, Q \cdot t_S / t_C)) + 1,$$

where:

- Q represents the current size of the request queue on the server;
- t_S represents the average server time per puzzle solving operation;
- t_C represents the average client time per puzzle solving operation.

If no additional information is present regarding an imminent attack, identity provider A will choose the puzzle difficulty proportionally with the size of the request queue or according to a predefined variation rule. This will result in the same server behavior regardless of the presence of an attack.

If additional information is present, like in the case of identity provider B, the current state of the system can be used to decide on the variation rule of puzzle complexity k, as follows:

- In the *normal state*, the server does not reach its maximum load except for a few peak moments. As a consequence, lower complexity puzzles can be chosen from the lower region of the $[0, k_{max})$ interval.

- In the *alerted state*, the server load constantly rises above the usual threshold, without necessarily indicating that an attack is in progress. However, as a precaution, medium complexity puzzles will be selected within the $[0, k_{max}]$ range.
- For the *attacked state*, in which an attack has been identified with a reasonable certainty, the server will use k_{max} as difficulty level for all puzzles, even if it has not reached its maximum load.

Experimental Results

We have implemented SSO-SENSE on the Microsoft .NET 3.5 framework, using the C# language. In order to determine the behavior of the Bayesian inference module, the test scenario was composed of two real-life situations:

- Repeated failed authentication attempts, purposely generated by an attacker to increase the server load;
- Repeated breaks the normal flow of the protocol - by sending responses which do not comply with the protocol specification - which cause the server to keep waiting for a proper response until a timeout occurs.

During the simulation, the initial state of the SSO system was the *normal state*, with parameters Q = 400000, $t_S = 0.003$ and $t_C = 0.5$. As a result, $k_{max} = 12$.

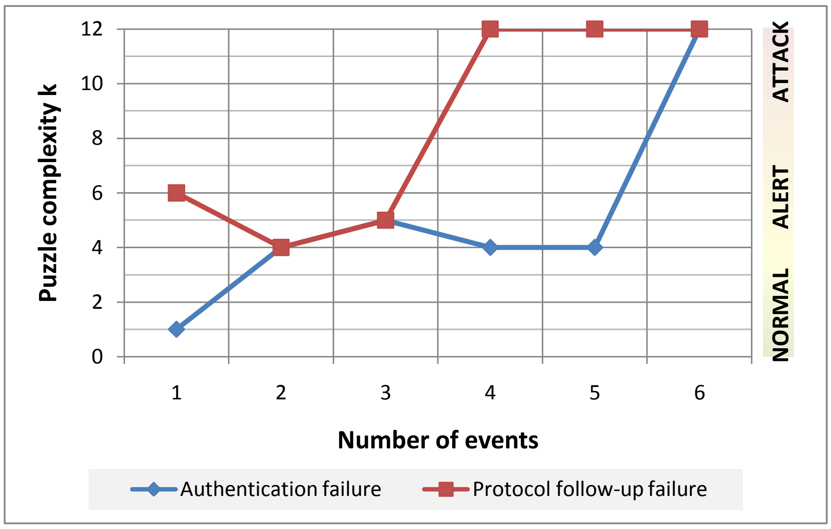

Figure 5. SSO-SENSE simulation results

Failed authentication attempts are low gravity events, therefore the transition from the normal to the attack state is done by passing through several alert states. As shown in the first series of *Figure 5*, after the first failed attempt the system maintains the normal state since such an event can be commonly generated by legitimate users. However, the following events will increase the system awareness by raising the threat level and forcing the system into a more defensive state, thereby signaling the system administrator that the gravity of the situation has been growing.
Protocol errors are more serious as established through the conditional probabilities associated to the hypotheses in the Bayesian inference engine. As a result, the system converges to the *attacked state* faster, as illustrated in the second series of *Figure 5*.

THRESHOLD PUZZLES AND ADAPTIVE THRESHOLD PUZZLES

The Need for Threshold Puzzles

The behavior of an authentication system protected by the current client puzzle technology can sometimes be suboptimal. There are two main reasons for this:

1. There is no upper limit to the puzzle complexity, which can result in clients spending too much time solving high complexity puzzles;
2. There is no minimum response time which could prevent an attacker from finding the puzzle solution too fast and overloading the server with a burst of requests.

Starting from these two limitations of client puzzles, we propose an improved solution which overcomes them by applying thresholds to both the puzzle complexity and the client response time. We will call the improved solution a *threshold puzzle* (Bocan, 2004).

Establishing an Upper Limit for the Puzzle Difficulty

The current client puzzle design specifies a difficulty range between 0 (no solving effort required) and 128 or 192 (theoretically impossible to solve, according to the employed hash function). Since this difficulty increases exponentially the current implementations of this mechanism are limited to using a narrow value range. High difficulty levels would result in DoS attacks targeted at legitimate clients, since these clients may spend a significant amount of time solving the puzzle.

In order to obtain an optimal perception of puzzle difficulty at client level, the solving time spent by the client must be lower than the time needed by the server to service a client request given the current load, which translates into the following inequality:

$$T_{client} \leq T_{server}.$$

Considering M to be the average number of operations required to solve the puzzle and t_C, the average time per operation at the client level, we can define the solving time at the client level as

$$T_{client} = M \cdot t_C$$

M can in turn be expressed in relation to the puzzle complexity k, since

$$M = 2^k \cdot (2^k + 1) / 2^{k+1} \approx 2^{k-1}$$

which leads to the approximation $M \approx 2^{k-1}$. As a result, we obtain the following equation:

$$T_{client} = 2^{k-1} \cdot t_C.$$

The time in which the server is able to respond to a request is proportional to the current size of the request waiting queue Q as well as to its average time per operation t_S, therefore the time spent by the server becomes

$$T_{server} = Q \cdot t_S.$$

The initial inequality can now be written as

$$2^{k-1} \cdot t_C \leq Q \cdot t_S$$

which gives us the upper limit for the puzzle complexity k, since

$$k \leq \log_2(Q \cdot t_S / t_C) + 1.$$

In case of a low server load, the logarithm quantity $Q \cdot t_S / t_C$ may be smaller than 1, which could result in a negative value for k. In order to avoid this situation, we will consider the following inequality which ensures that k has always a positive value

$$k \leq \log_2(\max(1, Q \cdot t_S / t_C)) + 1.$$

Figure 6 shows the comparative evolution between the difficulty of client puzzles and threshold puzzles for $t_S = 0.003$ and $t_C = 0.5$. It can be clearly seen that the client perception of performance degradation as server load increases is less significant when threshold puzzles are used.

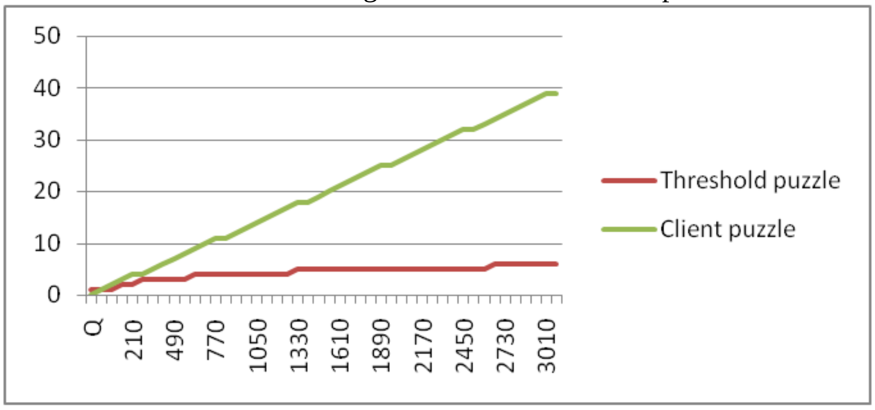

Figure 6. Comparative evolution of puzzle difficulty between client puzzles and threshold puzzles

Setting-up a Minimum Response Time Threshold

For a DoS attack to be successful, the attacker must be capable of sending numerous requests to the server in a short time interval, despite the usage of client puzzles technology. To prevent the attacker from finding the solution too fast, the sever can associate to the puzzle a minimum time threshold necessary to find the solution, according to the puzzle difficulty. If the client response is received faster than this threshold, the server can interpret this as an attack and limit or terminate its communication with the client.

The basic idea consists of adding a timestamp T_S to the puzzle in order to mark the precise moment when the server has generated its random value N_S. When the puzzle solution is received, the server is able to calculate the exact time required by the client to find the solution. This time span should not be lower than a server estimation based on the difficulty level k. If it is, then the server can consider that it is under a strong DoS attack and should cease all communication with the client.

As shown in the previous paragraph, the average number of operations needed to solve a puzzle is 2^{k-1}, so the estimated time T_E for finding the solution, which is the minimum response time threshold, can be calculated with the following formula:

$$T_E = 2^{k-1} \cdot T_{operation}$$

where:

- k represents the puzzle difficulty;
- $T_{operation}$ is the average time needed to perform a cryptographic operation.

The Threshold Puzzles Algorithm

Given the theoretical considerations presented in the previous sections, we can now outline the complete threshold puzzles algorithm.

1. When a new communication channel is opened between the server and a client, the server checks the system state, using its own metric (for example, the one provided by the SSO-SENSE detection engine). If the server load does not exceed a critical threshold, the algorithm stops since no defense mechanism is required.
2. The server generates a unique random value N_S (also called a nonce), with a 64 bit entropy and records the timestamp T_S at which the value was generated.
3. The server sets the puzzle difficulty level k and estimates the minimum response time threshold T_E. The difficulty level k will be limited to an upper threshold so that an uncontrolled increase in difficulty will not have repercussions on clients with limited computational power.
4. The server creates a new puzzle, in the form of a tuple $<N_S, k, T_S>$, which is sent to the client.
5. When the puzzle is received, the client performs the following operations:
    a. Checks the timestamp T_S to ensure that the information is recent.
    b. Generates a random number N_C.
    c. Searches for the puzzle solution X, by applying repeatedly a hash function h to the arguments C, N_S, N_C and X, where C represents the client identity. Considering $Y = h(C, N_S, N_C, X)$ to be the output of the hash function, the puzzle is considered to be solved when the first k bits of Y are equal to 0.
    d. Sends back the solution to the server.
6. When the server receives the solution from the client, it executes the following steps:
    a. Calculates the time needed for the client to solve the puzzle, $T_{solve} = T_R – T_S$, where T_R represents the reception time. If T_{solve} is smaller than T_E, the client has a large computational power at its disposal and the server may choose to either terminate or to deliberately delay the communication with the client.
    b. Checks if the client has previously submitted a solution with the same N_S and N_C parameters. If so, the server ceases all communication with the client.
    c. Checks that solution X is correct.
7. If all the above requirements have been satisfied, the server can allocate resources for executing the authentication protocol with client C.

Adapting Threshold Puzzles to the Computational Power of the Client

Another aspect related to the threshold puzzle technology is the wide range of communication devices that can be used to access a certain network service and interact with an authentication system. Devices such as state-of-the-art desktop systems, laptops and notebooks, PDAs or smartphones can be used to connect to the same network service, and their computational power can range from very fast to medium or slow. In conjunction with the threshold puzzles technology, the user experience in relation to the server response time could vary greatly in similar load conditions. As a consequence, threshold puzzles need to be adapted to the computational power of each client in order to obtain a seamless user experience. We will call this concept *adaptive threshold puzzles* (Bocan & Fagadar, 2005a).

The main idea behind this approach is to enable the server to determine the computational power of the client and to adapt the puzzle difficulty accordingly. This assessment can take place during the first dialog between client and server, when the server sends an exploratory puzzle to the client and, based on the response time, identifies its computational power. The exploratory puzzle can either be a partial hash function inversion (similar to the client puzzles concept) or a completely different approach can be used depending on the implementation. A linear dependency between the difficulty and the solving time of the exploratory puzzle is desired.

The computational power P_C of the client can be calculated by the server based on the time it needed to solve the exploratory puzzle. It is possible though, for a malicious client to intentionally delay the response in order to hide the true value of P_C and to appear weaker in front of the server. If such a client launches an attack, it may lure the server into sending it lower complexity puzzles. Therefore, a method must be found in order to encourage the client to use its whole computational power when solving the exploratory puzzle. A solution to this problem is for the server to allocate a limited number of connections within a certain time span, according to the reported P_C value for a client. For example, a powerful client will be allocated a number N of connections within a time slot, while a slow client like a PDA or a smartphone will be allocated only N/2 or N/3 connections within the same time slot. As a result, the malicious client will receive a smaller number of connections than expected, and those connections that exceed this number will be ignored by the server. This way, clients who deliberately hide their true computational power will not be able to launch an attack to the full extent of their capabilities.

Once P_C is evaluated by the server, the adapted complexity k_C of the adaptive threshold puzzle can be calculated using the formula

$$k_C = \text{round}(\, k \cdot \log_2(P_C / P_{ref}))$$

where:

- P_{ref} is the reference computational power defined at server level,
- P_C is the reported client computational power,
- k represents the reference puzzle difficulty, correlated with P_{ref}.

The reference puzzle difficulty k uses the same formula as it the case of threshold puzzles

$$k \le \log_2(\max(1, Q \cdot t_S / t_C)) + 1$$

where:

- Q represents the current size of the request queue on the server,
- t_S represents the average server time per cryptographic operation,
- t_C represents the average client time per cryptographic operation.

The Adaptive Threshold Puzzles Algorithm

Based on the theoretical aspects described in the previous paragraph, we can now list the complete adaptive threshold puzzles algorithm as an evolution of the threshold puzzles algorithm presented before.

1. When a new communication channel is opened between the server and a client, the server checks the system state. If the server load does not exceed a critical threshold, the algorithm stops since no defense mechanism is required.
2. If it is the first time the client connects to the server, its computational power must be estimated. The server creates an exploratory puzzle, which is the simple partial inversion of a hash function, with a medium complexity (e.g. k = 6) and sends it to the client.
3. The client solves the exploratory puzzle and sends the solution back to the server.

4. The server checks the exploratory puzzle solution and estimates the computational power of the client, P_C. Depending on this quantity, the server will determine the maximum number of requests allowed for this client within a time slot.
5. The server generates a unique random value N_S (nonce), with a 64 bit entropy and records the timestamp T_S at which the value was generated.
6. The server calculates the adapted puzzle complexity k_C and estimates the minimum response time threshold T_E.
7. The server creates a new puzzle, in the form of a tuple $<N_S, k_C, T_S>$, which is sent to the client.
8. The client receives the puzzle, calculates the solution X as shown in the threshold puzzles algorithm and sends it back to the server.
9. The server checks the solution both in terms of correctness and in terms of solving time, according to the rules mentioned in the threshold puzzles algorithm.
10. If solution is correct, the server will apply a restriction on the maximum number of requests allowed for the client within a time unit, according to the P_C value determined in step 2.
11. If all the above requirements have been satisfied, the server can allocate resources for executing the authentication protocol with client C.

CASE STUDY: THE SSL HANDSHAKE PROTOCOL WITH ADAPTIVE EFFORT DISTRIBUTION

Overview of the SSL Handshake Algorithm

Given its wide-spread and popularity, we have chosen Secure Socket Layer (SSL) protocol to test the adaptive threshold puzzles technique. Before describing the changes made to the protocol in order to support the puzzle technology, we will present a short overview of its original, unmodified version. *Figure 7* illustrates the SSL Handshake protocol, which will be the target of our improvements. During this phase, the message exchange between peers contains information regarding the cryptographic capabilities of the client as well as the configuration chosen by the server to enable the communication with the client.

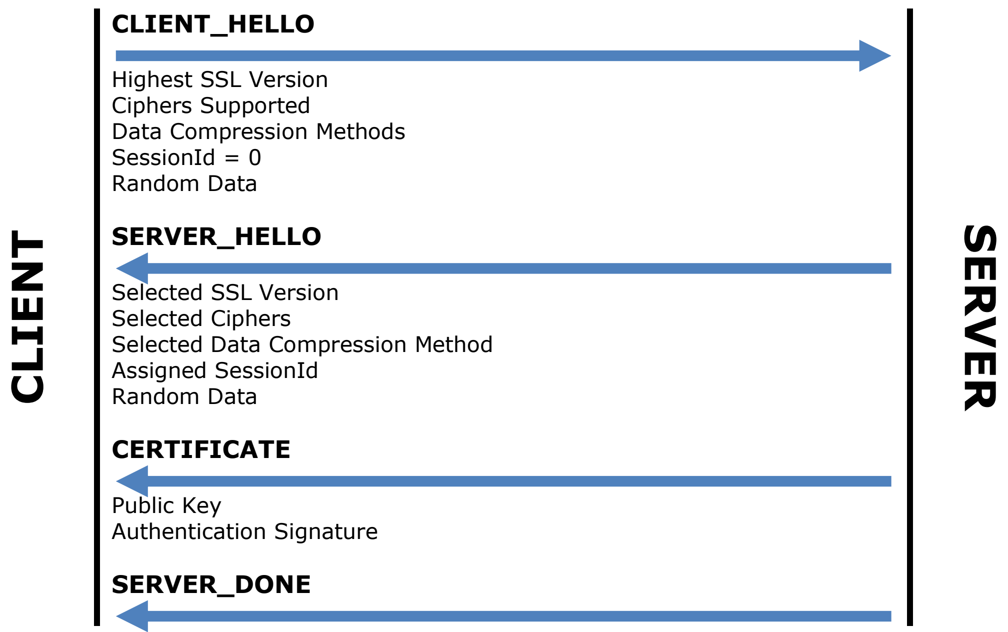

Figure 7. The original SSL Handshake protocol

It can be easily seen that the CERTIFICATE message contains the digital signature of the server, which represents the server engagement (and resource allocation) regardless of client identity and its true intentions. If the client does not intend to continue its dialog with the server but instead aims to overload it with useless requests that will end up being signed by the server, we are dealing with a typical DoS attack.

From the perspective of service availability, allocating server resources unconditionally is a major drawback. This is why we have focused on improving the mostly used authentication protocol, SSL, by using the adaptive threshold puzzle technology. The changes we brought are extensions to the original protocol, which provides it with the capability of balancing the authentication effort between peers without affecting its cryptographic validity.

Adding Adaptive Effort Distribution to the SSL Handshake Protocol

In order to extend the SSL Handshake protocol with support for adaptive threshold puzzles, we need to add additional messages to the information exchange between client and server.
First, the server must be able to assess the computational power of a new client, by introducing two new messages to the protocol: PUZZLE_EXPLORE_CHALLENGE and PUZZLE_EXPLORE_SOLUTION. Through the PUZZLE_EXPLORE_CHALLENGE message, the server asks the client to solve an exploratory puzzle of medium difficulty. The client will submit the puzzle solution via the PUZZLE_EXPLORE_SOLUTION message. If the server already knows the client (prior requests have been received from the client in the past), it may choose to skip this phase for a predetermined time frame or for an undetermined time frame, according to the application specifics.
Second, the authentication effort must be distributed between peers, instead of being unilaterally supported by the server. This is achieved by adding two new more message types: PUZZLE_CHALLENGE and PUZZLE_SOLUTION which represent the information exchange between the server and the client during the normal threshold puzzle solving process. During authentication, the server must keep its load under a critical threshold. If this threshold is crossed, the server must control the avalanche of authentication requests, and it achieves this goal by keeping the client busy with a puzzle proportional in complexity with the server load.

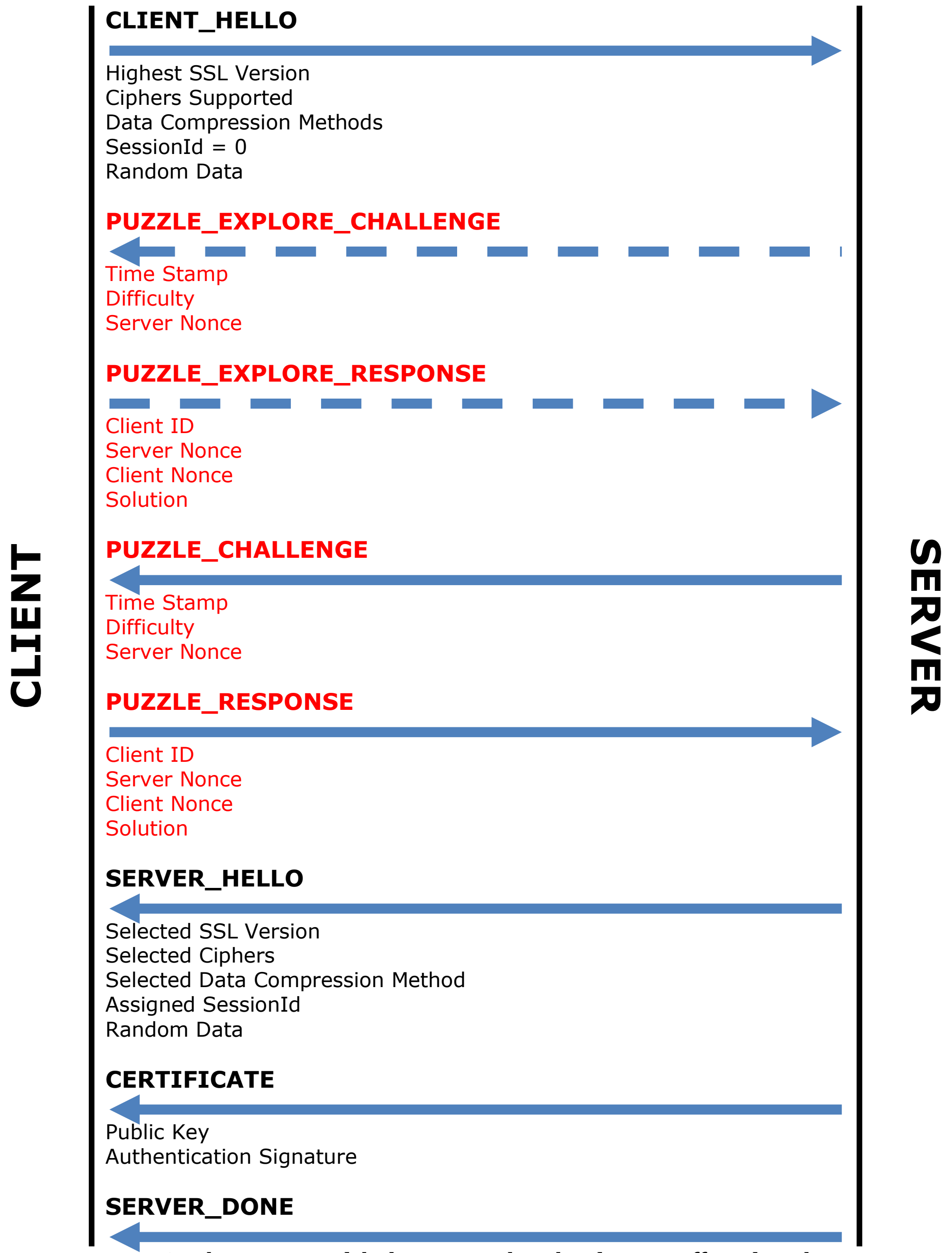

Figure 8. The SSL Handshake protocol with adaptive effort distribution

As shown in *Figure 8*, both the PUZZLE_EXPLORE_CHALLENGE and the PUZZLE_CHALLENGE messages contain the server timestamp (T_S), the puzzle difficulty (k and k_C respectively) and the server nonce (N_S). The client responds with the PUZZLE_EXPLORE_SOLUTION and PUZZLE_SOLUTION respectively, both containing its identity (C), the server nonce, the client-generated nonce (N_C) and the puzzle solution (X).
As it can be seen from the message exchange diagram, adding the new message types does not affect the cryptographic integrity of the protocol. Instead, the client is delayed with a duration proportional to the current server load and health state. Since the puzzle-related messages are exchanged before the CERTIFICATE and SERVER_DONE messages, the server does not commit to resource allocation until the client is validated. We call this adaptive effort distribution since the client delay and the resource allocation at server level are related to the computational power of the client.

A Test Prototype for the Threshold Puzzles Technology

In order to simulate the behavior of the threshold puzzle technology in a real context and to outline its benefits compared to the standard client puzzles, we have built the prototype illustrated in *Figure 9*.

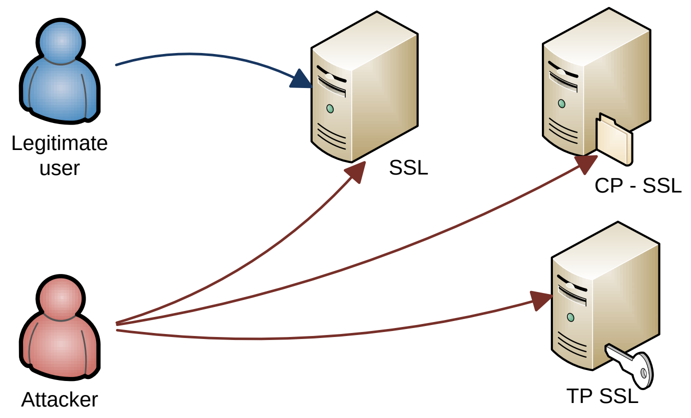

Figure 9. Test prototype for the Threshold Puzzles technology

The prototype contains the following modules:

- *Legitimate user*: a normal client which attempts a SSL authentication according to the protocol specification.
- *Attacker*: a client with access to large computational power, able to make a large number of authentication requests within a short time frame.
- *SSL server*: a regular, unprotected SSL server.
- *CP SSL server*: a SSL server protected by standard Client Puzzles (CP) technology.
- *TP SSL server*: a SSL server protected by Threshold Puzzles (TP) technology.

Since in laboratory conditions it is hard to gather a really large computational power, in order to simulate this aspect we have configured the *Attacker* module to submit a random, incorrect, solution, while in the same time we have disabled the solution check at the server level. This way, the attacker appears to be solving the solution in a much shorter timeframe than the legitimate clients, without impacting in any other way the simulation results.
The simulation ran on two systems connected through a 100 Mbps local network with no disturbing traffic. The first system, representing the server, hosted the *SSL server*, *CP SSL server* and *TP SSL server* modules, while the second one hosted multiple instances of the *Legitimate user* and the *Attacker* modules.

Experimental Results

On the test environment described previously we have run several scenarios designed to determine the behavior of the Client Puzzles and Threshold Puzzles technologies, as follows:

- **Clients authenticated by a regular SSL server** – used as a reference when benchmarking the system performance, this scenario allowed us to measure the average time needed to completely execute the SSL protocol without any puzzle technology extensions. For a number of 100 authentications of 15 clients, the average server response time was 4545 milliseconds.
- **Clients and attacker authenticated by a regular SSL server** – the simulation contained 14 normal clients and an attacker. The attacker sends a burst of CLIENT_HELLO messages in order to trigger an equal number of responses signed by the server. The protocol execution is never finished by the attacker, who deliberately ignores the server response. The attacker has sent false requests with a rate of 30 requests / second, during which the server reached a 100% load. In this case, the average server response time for legitimate clients has increased to 11320 milliseconds.
- **Clients and attacker authenticated by the CP SSL server** – this scenario is the same as the previous one, with the exception that Client Puzzle technology has been activated at server level. Since we have simulated the availability of a large computational power at the attacker level and disabled the solution check at server level, for the same rate of 30 requests / second

the server reached the 100% load and the average authentication time has increased again to 12730 milliseconds. This increase can be explained by the overhead introduced by the two additional protocol messages: PUZZLE_CHALLENGE and PUZZLE_RESPONSE.

- **Clients and attacker authenticated by the TP SSL server** – we have repeated the above scenario, using Threshold Puzzles technology. In this case, the average authentication time of legitimate clients has decreased significantly to 4553 milliseconds, almost the same as the one obtained when the server was not subject to any attack. This was due to the fact that the attacker solved the puzzles too fast, which lead to the server blocking its communication channels with the attacker.

Conclusion

Figure 10 summarizes the test results of the SSL Handshake protocol simulations.
During a so-called strong attack, the Client Puzzles technology did not have the expected results since the attacker was able to find the puzzle solution in a very short time frame and still send a large number of requests to the server. The average response time was similar to that of an unprotected server.
On the other hand, the server which employed Threshold Puzzles has systematically rejected the requests which were accompanied by solutions found too fast for their complexity level, so that the authentication time perceived by legitimate clients has not been altered.

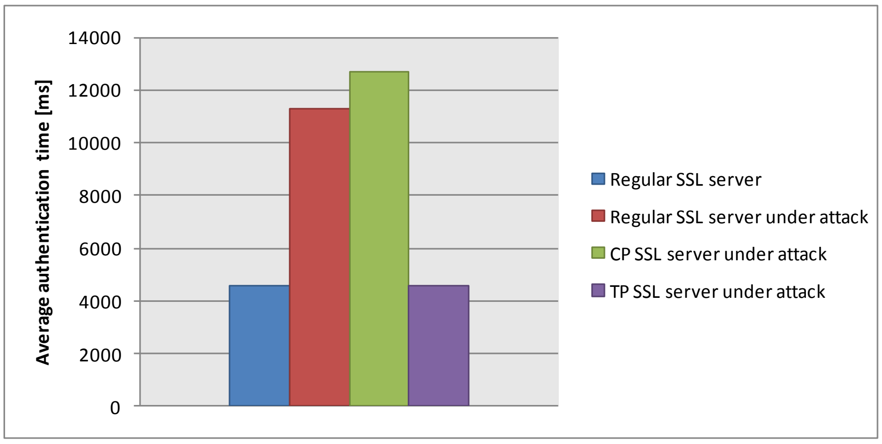

Figure 10. Benchmark results for SSL Handshake protocol simulations

FUTURE RESEARCH DIRECTIONS

Classic puzzle designs have two important drawbacks that make them less ideal for the purpose of thwarting Denial-of-Service attacks. The first and most serious drawback is *parallelizability*, i.e. the possibility of computing the puzzle solution in parallel. In scenarios where a single adversary is in control of a large number of compromised machines, the huge computation power thus gained can be used to compute the solution much faster than the server expects. The second drawback of classic puzzle designs is the *lack of fine granularity*, i.e. the server is not able to adjust the solution time precisely.
Both non-parallelizability and fine granularity are important properties of good puzzles, but they are difficult to obtain. Therefore, new constructions have emerged as a solution to this problem and one such construction is the *puzzle chain* (Groza & Petrica, 2006; Ma, 2006). Instead of having one single puzzle of varying difficulty, one may use a chain of interdependent puzzles of smaller difficulty. This allows fine adjustments in time solving by altering the chain length with the added benefit that the intermediate or the final solution cannot be computed in parallel.
Puzzle chains come with a unique set of drawback themselves. One important issue is keeping and maintaining the chain state at the server level as well as transmitting it to the client. This requires important storage and high-bandwidth communication channels, but clever and judicious scheduling of resources alleviates this problem.

We intend to continue our research on increasing the availability of authentication protocols through the use of puzzle chains and provide a framework for general implementations in order to leverage our experimental findings. Though we are in the early stages of our research, we have already drawn the outlines of the augmented Adaptive Threshold Puzzles concept:

- As the puzzle chain solving time is driven by two factors – individual link difficulty and the chain length itself – we are in the position of integrating this into our earlier SSO-SENSE detection engine. While the server would be able to establish the desired puzzle difficulty based on the current system load – difficulty range which itself is rather narrow, the SSO-SENSE module will issue to clients chains of varying lengths, based on the threat level sensed by the system and on the computational power advertised by clients. This design allows a very flexible and democratic model or resource allocation and keeps under a natural self-control any client which misbehaves.
- The mapping process based on the computational power of the client corroborated with the sensed system threat level allows for a fine and judicious allocation of resources, where no resource livelock is possible. This means that clients are ordered by a democratic ranking where no single client is able to obtain entire or the majority of server resources.

CONCLUSION

DoS attacks represent a permanent threat to the present communication systems in general and to authentication systems in particular. In this chapter we have shown that authentication systems, including Single Sign-On (SSO) systems, are vulnerable to DoS attacks due to the lack of control over the resources allocated during the authentication process. This can result in severe performance degradation or even failure in delivering the authentication service to legitimate clients.

To overcome the vulnerability of authentication systems facing the threat of DoS attacks, we have brought several contributions meant to allow the early detection and prevention of such attacks. As a first step, the SSO-SENSE heuristic threat assessment engine was introduced, to facilitate the detection of DoS attacks at an early stage and to allow efficient deployment of countermeasures against the attacker. In the second stage, we have developed the Threshold Puzzles and Adaptive Threshold Puzzles technologies, to address the scenarios which were not covered by classical client puzzles. In the last step, we have modified the widespread SSL Handshake protocol in order to support the Adaptive Threshold Puzzles technology for an efficient protection against DoS attacks. Based on the experimental results collected from a simulation platform, we can conclude that the proposed changes lead to a considerable increase in DoS resilience for an authentication system and that they prove to be a viable solution in securing authentication-based network services.

REFERENCES

Aura, T., Nikander, P., & Leiwo, J. (2000). DOS-resistant authentication with clientpuzzles. *Lecture Notes in Computer Science*. *Proceeding of the Cambridge Security Protocols Workshop 2000* (pp. 170-177). Cambridge, UK. doi: 10.1.1.106.9259

Bocan, V. (2004). Threshold Puzzles. The Evolution of DoS-Resistant Authentication. *Periodica Politehnica, Transaction on Automatic Control and Computer Science, 49*(63). Timisoara, Romania.

Bocan, V., & Fagadar-Cosma, M. (2005, November). Adaptive Threshold Puzzles. *Proceedings of EUROCON 2005. The International Conference on "Computer as a tool"* (pp. 644-647). Belgrade, Serbia. doi: 10.1109/EURCON.2005.1630012

Bocan, V., & Fagadar-Cosma, M. (2005, November). Towards DoS-resistant Single Sign-On Systems. *Proceedings of EUROCON 2005. The International Conference on "Computer as a tool"* (pp. 668-671). Belgrade, Serbia. doi: 10.1109/EURCON.2005.1630018

Crosby, S. A., & Wallach, D. S. (2003, August). *Denial of Service via Algorithmic Complexity Attacks*. Paper presented at the 12th USENIX Security Symposium, Washington, DC.

Dwork, C., & Naor, M. (1992). Pricing via Processing or Combating Junk Mail. *Proceedings of CRYPTO '92* (pp. 139-147). Berlin, Germany: Springer-Verlag.

Groza, B., & Petrica, D. (2006, May). On Chained Cryptographic Puzzles. *Proceedings of 3rd Romanian-Hungarian Joint Symposium on Applied Computational Intelligence, SACI'06* (pp. 182-191).Timisoara, Romania.

Harris, S. (2001, September). *DoS Defense*. Information Security Magazine.

Juels, A., & Brainard, J. (1999). Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks. *Proceedings of the NDSS'99* (pp. 151-165).

Kim, Y., Lau, W. C., Chuah, M. C., & Chao, H. J. (2004). PacketScore: Statistics-based Overload Control against Distributed Denial-of-Service Attacks. *Proceedings of IEEE INFOCOM* (pp. 2594-2604). Hong Kong, SAR. doi: 10.1.1.137.6263

Liberty Alliance Project (2010). Liberty ID-FF Protocols and Schema Specification 1.2. Retrieved April 26, 2010, from http://www.projectliberty.org/liberty/content/view/full/179/(offset)/15/

Liberty Alliance Project (2010). Liberty Specs Tutorial. Retrieved April 26, 2010, from http://www.projectliberty.org/liberty/specifications__1/

Ma, M. (2006, April). Mitigating denial of service attacks with password puzzles. *Proceedings of International Conference on Information Technology: Coding and Computing, Vol. 2* (pp. 621-626). Las Vegas, NV. doi: 10.1109/ITCC.2005.200

Merkle, R. C. (1978). Secure Communications Over Insecure Channels. *Communications of the ACM, 21(4).*

Rivest, R. R., Shamir, A., & Wagner, D. A. (1996). Time-lock Puzzles and Timed-release Cryptography. Retrieved April 26, 2010, from http://lcs.mit.edu/~rivest/RivestShamirWagnertimelock.pdf

Schneier, B. (2000, March). *Distributed denial of service attacks*. Crypto-gram Newsletter.

SETI @home Program (2010). Retrieved April 26, 2010, from http://setiathome.ssl.berkely.edu

Siaterlis, C., & Maglaris, B. (2004). Towards Multisensor DataFusion for DoS Detection. *Proceedings of the 2004 ACM symposium on Applied Computing* (pp. 439-446). ACM Press. doi: 10.1.1.9.8572

Spatscheck, O. & Peterson, L. (1999). Defending against Denial of Service Attacks in Scout. *Proceedings of the 1999 USENIX/ACM Symposium on OSDI* (pp. 59-72). doi: 10.1.1.37.157

The Distributed.net Organization (2010). Retrieved April 26, 2010, from http://www.distributed.net

Waters, B., Juels, A., Halderman, J. A., & Felten, E. W. (2004). New Client Puzzle Outsourcing Techniques for DoS Resistance. *Proceedings of 11th ACM Conference on Computer and Communications Security*. doi: 10.1.1.58.737

KEY TERMS & DEFINITIONS

Authentication system – a mechanism that establishes the identity of two parties either one way or both ways. Authentication systems usually employ cryptography and involve a secret quantity known by the parties.

Denial of Service – usually abbreviated as DoS, is an attack targeted against a computer system which causes it to malfunction. The attacks “deny” access of the legitimate clients to the resources and services of the computer system by overwhelming it with false requests that are usually indistinguishable from legitimate ones.

Client Puzzle – a technology originally proposed as a way to increase the computational cost for the client in order to limit the request rate for the server. The technology most commonly involves the partial inversion of a cryptographic hash function.

Threshold Puzzle – a technology similar to the client puzzle which limits the puzzle difficulty level in order to avoid overloading legitimate clients with low computational power.

Adaptive Threshold Puzzle – a technology similar to the threshold puzzle that takes into account the computational power of the client. An adaptive threshold puzzle is able to discriminate its clients and ask puzzle solutions of varying difficulties.

Bayesian Inference – is a method of statistical inference that calculates the probability of an event to be true based on observations of evidences.

Adaptive Effort Distribution – a technique that adapts the threshold puzzle concept to the existing Secure Sockets Layer (SSL) protocol. This essentially means the addition of four new messages to the existing protocol design with the aim of gathering knowledge of the client computational power.