Defending Web Services Against Denial of Service Attacks Using Client Puzzles

Suriadi Suriadi, Douglas Stebila, Andrew Clark, and Hua Liu*
Information Security Institute, Queensland University of Technology
Brisbane, Queensland, Australia
Email: {s.suriadi,stebila,a.clark}@qut.edu.au, hua.liu@connect.qut.edu.au

***Abstract*—The interoperable and loosely-coupled web services architecture, while beneficial, can be resource-intensive, and is thus susceptible to denial of service (DoS) attacks in which an attacker can use a relatively insignificant amount of resources to exhaust the computational resources of a web service. We investigate the effectiveness of defending web services from DoS attacks using client puzzles, a cryptographic countermeasure which provides a form of gradual authentication by requiring the client to solve some computationally difficult problems before access is granted. In particular, we describe a mechanism for integrating a hash-based puzzle into existing web services frameworks and analyze the effectiveness of the countermeasure using a variety of scenarios on a network testbed. Client puzzles are an effective defence against flooding attacks. They can also mitigate certain types of semantic-based attacks, although they may not be the optimal solution.**

***Keywords*-web services; denial of service attacks; client puzzles**

I. Introduction

The loosely-coupled architecture of web services technologies allows seamless integration and composition of separate applications (including legacy applications) built on various platforms. Unfortunately, these benefits come with technological complexities that can be vulnerable to denial of service (DoS) attacks. This vulnerability extends to not only XML-related processing but also many web services standards (such as WS-Security [1], WS-Policy [2], and others) which have been widely implemented by major industrial players. The problem of DoS in web services is not new and has been widely studied *and validated* in many academic and industry-based works [3]–[8].

DoS attacks have been used as tools to make political statements [9] and extortions [10]. The latest high-profile DoS attacks against MasterCard, Visa, and other organizations linked to the late-2010 WikiLeaks incident [11] only highlight the vulnerability and susceptibility of many organizations to DoS attacks. The increased use of web services technologies to deliver major governmental services (such as the Australian Standard Business Reporting (SBR) system[1]) and to enable cloud computing (including Amazon clouds[2]) only highlights the urgency of addressing the DoS problem in web services.

Recent work [5] shows that flooding attacks are still an effective way to exhaust a web service provider's CPU resources. Unfortunately, as discussed in Section VI, most existing work has not addressed the *resource imbalance* issue that is the key to successful flooding-based DoS attacks. Furthermore, most of the previously proposed mitigation strategies require additional *separate* components (outside of the web services realm) to be deployed in the runtime environment for their solutions to be effective.

The work presented in this paper attempts to (1) rectify this resource imbalance by requiring clients to perform some work to arrive at a puzzle solution to prove their legitimate intention in requesting services, and (2) provide a DoS mitigation capability that can be *integrated* into any existing web services applications without the need for additional components or infrastructure outside of the web services application's realm, similar to how the WS-Security standard provides an integrated confidentiality, integrity, and some authenticity protection in web services applications themselves. We *do not* claim that the proposed solution can be used to mitigate all types of DoS attacks; rather, the proposed solution can mitigate some DoS attacks and can be integrated seamlessly with existing web services platforms.

Contributions: The main contribution of this paper is the study of the effectiveness of client-puzzles as an integrated built-in DoS defence mechanism for two main types of DoS attacks: flooding attacks and semantic attacks. While client puzzles should *theoretically* be an effective DoS defence mechanism, the complexities of existing web services platforms may introduce overheads which could render the client puzzles protection ineffective. Therefore, it is important that we *validate the theoretical effectiveness of client puzzles* through experiments. To our knowledge, this is the first time that client puzzles have been used as a DoS defence mechanism in web services.

We implemented a hash-based cryptographic client puzzle in both .NET WCF- and Java Metro-based web services. We conducted several experiments which show that, despite the complexities often associated with web services technologies, the minimal overhead needed to verify a client's puzzle solution enables this technique to be an effective defence mechanism against flooding attacks. In particular, it is useful to protect web services applications whose

* This work was supported by the Australia-India Strategic Research Fund project TA020002.

[1]https://www.sbr.gov.au/content/public

[2]http://s3.amazonaws.com/ec2-downloads/2010-11-15.ec2.wsdl

invocation triggers one or more resource-intensive actions, such as a series of web services operations orchestrated using the Business Process Execution Language (BPEL) or complex mathematical operations.

We have also performed experiments to study the effectiveness of client puzzles in mitigating a type of semantic-based DoS attack which attempts to force a server to perform heavy cryptographic processing according to the WS-Security standard [1]. Our preliminary results show that while client puzzles offer some protection against this type of attack, they are not an ideal mitigation technique.

This paper is structured as follows: Section II provides an explanation of the client puzzle technique that we have integrated into existing web services platforms, followed by a brief description of the two DoS attacks used in the experiments in this paper. Section III explains how we have integrated the client puzzle technique into our test environment and Section IV describes the experiments we have conducted and the results. Section V discusses the effectiveness and limitations of the client puzzle technique in mitigating the DoS problem in web services. We discuss related work in Section VI and conclude in Section VII.

II. Background

We provide a description of the implemented client puzzle technique and a summary of the DoS attacks which the client puzzles are supposed to mitigate in our experiments.

A. Client Puzzles

Client puzzles, also called *proofs of work*, can be used to counter resource-depletion denial of service attacks: before a server is willing to perform some computationally expensive operation, it requires that the client commit some of its own resources and solve some moderately hard puzzle. Client puzzles were first proposed by Dwork and Naor [12] to control junk email by having recipients only accept emails if they were accompanied by a correct puzzle solution, and have since been extended to protect cryptographic protocols such as authentication [13], [14] and key exchange [15], [16] protocols, as well as network protocols such as TCP [17] and TLS [18], [19].

The most commonly proposed type of client puzzle is a *hash-based computation-bound* puzzle, in which a client is required to find a partial preimage in a cryptographic hash function. For example, in the puzzle of difficulty d proposed by Aura *et al.* [14], the client C and server S supply nonces N_C and N_S, respectively, and the client must find a solution X such that

$$H(C, N_S, N_C, X) = \underbrace{0 \ldots 0}_{d} \| Y \ , \qquad (1)$$

where H is a cryptographic hash function, such as SHA-1, and the output starts with at least d 0 bits followed by any string Y. If H is a preimage-resistant hash function, then, it should take a client approximately 2^{d-1} calls to H to find a valid solution. However, the verification cost for a server is very low, as it only takes a single hash function call to check equation (1).

Client puzzles can be analyzed in *interactive* or *non-interactive* attack scenarios [20]. In the interactive case, the server nonce N_S is updated for every puzzle and thus an attacker cannot replay the same solutions or prepare many solutions and flood the server all at once. This comes at the cost of requiring online interaction for every request, so non-interactive puzzles can be more suitable for asynchronous or scenarios with fixed message flows. We have employed non-interactive puzzles in our experiments by providing the server nonce in an infrequently updated WSDL file.

B. DoS attacks on web services

We have tested the effectiveness of client puzzles for mitigating two types of DoS attack: a generic flooding attack, and a heavy cryptographic processing attack. The latter is a type of *semantic attack* which aims to exhaust server resources through a few specially crafted requests that require extraordinary processing (rather than the relying on the delivery of requests at a high rate).

1) Flooding Attack: This attack attempts to exhaust a server's resources by sending a large amount of *legitimate* requests. The request messages in this case are well-formed and valid without any malicious XML structure or content. Consequently, such an attack cannot be detected by relying on a signature-based XML firewall. Normally, such an attack is mitigated through some forms of lower network layer packet analysis, such as IP address analysis. In this paper, we study the effectiveness of integrated client puzzles for mitigating such attacks.

2) Semantic Attack: Heavy Cryptographic Processing Attack: A well-known type of a web services semantic attack is the heavy cryptographic processing attack in which an attacker sends a payload with an oversized WS-Security header containing many cryptographic elements (such as nested encryption [3], or a large number of digital signatures). The goal is to overload the server's resources, either through parsing a large security header or by forcing the server to process the numerous cryptographic directives.

III. Integrating Client Puzzles in Web Services

Although client puzzles can be implemented at the network or transport layer [17], [18], we aim to provide a solution that can be integrated into any web service without relying on additional components. Also, similar to the reasons that WS-Security protections are needed even when HTTPS is already widely used, transport layer protection only applies point-to-point, not end-to-end as required by web services. Thus, we chose to implement the client puzzles at the message layer as part of the SOAP header.

This paper examines the effectiveness of client puzzles in mitigating some DoS vulnerabilities at the server's side. To simplify the experimental setup, we implemented non-interactive client puzzles; this suffices for our purposes since non-interactive client puzzles already allow us to study the resources being consumed at the server side in verifying clients' solutions. More importantly, it allows us to study the ability of a server to fulfil honest clients' requests when the server is under DoS attacks.

Our implementation focuses on the integration of the client puzzle processing logic on the server side using existing web services development platforms, namely the Microsoft .NET Windows Communication Foundation (WCF) framework and the Java Metro framework. The server's use of client puzzles is communicated to the client as part of the server's policy in its WSDL file (as a custom WS-Policy assertion); a snippet of the generated WSDL advertising the client puzzle requirements (including difficulty level and server nonce) is given in Figure 1 (lines 4-5).

```
1 <wsdl:definitions ...>
2 <wsp:Policy wsu:Id="clientPuzzlePolicy">
3   <wsp:ExactlyOne> <wsp:All>
4     <wsp:clientPuzzle a:difficulty="8" xmlns:a="...">
5       abcdef</wsp:clientPuzzle>
6 ...</wsp:All></wsp:ExactlyOne></wsp:Policy>
7 <wsdl:types...>...</wsp:types>...
8 </wsdl>
```

Figure 1. Custom WS-Policy assertion for client puzzles in a WSDL.

We subsequently configured a client to understand the custom client-puzzle policy assertion and act accordingly by solving the puzzle and including the solution in the SOAP header of a request message. A sample SOAP request payload containing a client puzzle solution in the SOAP header is provided in Figure 2 (lines 2-6).

```
1 <s:Envelope...><s:Header...>
2   <ClientPuzzleSolution xmlns="..." ...>
3     <timestamp>634243948044717802</timestamp>
4     <clientNonce>LMBfqB</clientNonce>
5     <puzzleSolution>abcdef..</puzzleSolution>
6   </ClientPuzzleSolution> ......
7 </s:Header><s:Body>...</s:Body></s:Envelope>
```

Figure 2. Inclusion of client puzzle solution in a SOAP request.

To prevent clients from re-using a solution to launch replay attacks, we also require clients to include both a client nonce N_C and timestamp T (represented as the number of ticks (10^{-7}) since 1 Jan 0001): the client must provide a client puzzle solution X such that $H(N_S, N_C, T, X)$ starts with d zeros.

A. Java Metro Implementation

We implemented the client puzzles using the extensible `Tube` interface in Java Metro version 2.0. The `Tube` interface allows developers to modify (add, remove, or re-order) the sequence of modules to be called when processing web services messages at both the client's and the server's side. The client puzzle processing logic is implemented through the creation of a new Metro `Tube` module called the `ClientPuzzleTube` (which extends the `AbstractFilterTubeImpl` class) in which the client puzzle solution verification logic (at the server's side) is encoded. Then, we create a factory class called `ClientPuzzleTubeFactory` which implements the `TubeFactory` interface to instantiate the `ClientPuzzleTube` during the pipeline creation.

The order of tube (or module) processing is configured in the `metro.xml` file. This file is then bundled together with the other two Java classes as a `jar` file before being deployed on the server. In our configuration, the `ClientPuzzleTube` is placed near the beginning of the tube at the server's (endpoint's) side to ensure the processing of the puzzle solution occurs *before* other more resource-intensive tubes (such as security tubes).

B. .NET WCF Implementation

We also integrated client puzzles into a .NET (version 3.0) WCF-based web service. The inclusion and processing of a custom WS-Policy assertion (along with the ensuing processing to comply with a custom assertion at both the client's and server's side) are supported by the implementation and subsequent configuration of the `IPolicyExportExtension`, `IPolicyImportExtension`, and other related interfaces and classes. The web service is deployed on an Internet Information Services (IIS) server (version 7).

IV. Experiments and Results

Our testbed consists of a variety of clients and servers. The servers run web services applications with client puzzles integrated as described above. The servers and clients are connected via a switch.

The server for Java Metro-based web services runs on a laptop with an Intel Core 2 Duo 2.4 GHz processor and 4 GB RAM. These web services are deployed on a Glassfish server version 2.1.1 using its *default* configuration.

The server for .NET WCF web services runs on a desktop with an Intel Core 2 Duo 3 GHz processor and 4 GB RAM. These web services are deployed on an IIS server (version 7) using its *default* configuration as well.

In all experiments, the target/victim of our DoS attacks is a web services server. Client requests (both honest and malicious requests) are launched using the `wget` utility that is included with most Linux distributions.

To assess the effectiveness of the client puzzles, we conducted each experiment twice, first against a web service that does not employ client puzzles, and then against the same web service with client puzzles in use. We ran experiments

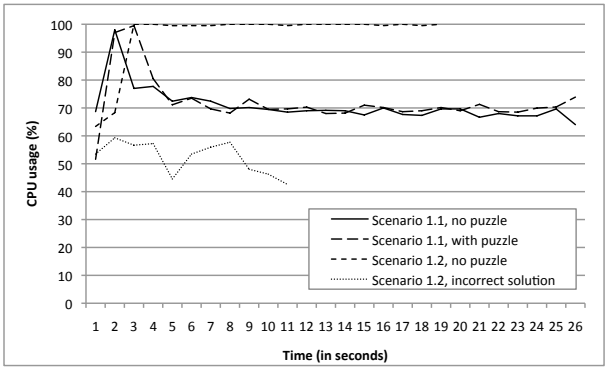

Figure 3. Java Metro flooding attacks – CPU usage.

for the two types of attacks described above: a generic flooding attack (Experiment 1) and a heavy cryptographic attack (Experiment 2). For each experiment, a variety of traffic scenarios were used.

A. Experiment 1: Flooding Attack

In this experiment, we set up a web service with one operation: performing a substantial complex mathematical operation (consisting of several thousand 64-bit floating-point exponentiations). The operation takes on average around 84 ms to complete when executed *independent* of the web services wrapping. Using this server as the victim, we ran several scenarios to assess the effectiveness of client puzzles in mitigating flooding attacks.

1) Scenario 1.1: Java Metro – Baseline Behaviour: We started this experiment by obtaining the baseline behaviour of the victim under 'normal' operation to show that the use of client puzzles does not incur significant processing overhead. Our baseline traffic generator sent 20 batches of (roughly) 25 simultaneous requests with a 1 second gap between batches. By comparing the CPU usage of the Java Metro-based web service without and with client puzzles, we can determine if any significant processing overhead is introduced in processing clients' puzzle solutions.

The results of this experiment, depicted in Figure 3, show that the CPU usage is quite similar, regardless of whether a client puzzle is used or not, so we can conclude that the processing of client puzzle solutions introduces no significant CPU usage overhead.

2) Scenario 1.2: Java Metro – Flooding Attack – Malicious Traffic Only: This scenario simulates a flooding attack by doubling the intensity of traffic compared with Scenario 1.1. A key difference between 'normal' and 'flooding' traffic is the (significantly) higher request rate in the latter case. We sent 10 batches of traffic (each with 50 simultaneous requests) with a one second gap between batches. The requests sent to the server without client puzzles do not contain any client puzzle solution, while those requests sent to the server implementing client puzzles contain an incorrect puzzle solution (we assume that the attacker simply sends random solutions to maintain the flooding attack intensity). Comparing the results from these two experiments allows us to learn if client puzzles allow the victim to protects itself from excessive CPU usage.

The results of this attack scenario, shown in Figure 3, indicate that client puzzles significantly reduce the amount of CPU consumption in a flooding attack: whereas CPU usage remained at 100% for the duration of the flooding attack when no client puzzle was used, the CPU usage never exceeded 60% when false requests could be dropped easily through the use of client puzzles. The shorter duration of the experiment in this Scenario 1.2 (due to only 10 batches of traffic) as opposed to the previous Scenario 1.1 (25 batches of traffic) explains the shorter curves for Scenario 1.2 CPU usage in Figure 3.

3) Scenario 1.3: Java Metro – Flooding Attack – Mixed Traffic: In this scenario, we mixed both honest traffic and malicious flooding traffic. The main goal is to observe if honest clients' requests are still being served when the server is under flooding attacks of various intensity. Several mixed traffic profiles are considered:

Profile 1: Constant Traffic Rate, Varying Honest and Malicious Requests: In this experiment, we mixed honest traffic (with correct puzzle solutions) and malicious traffic (with incorrect, random puzzle solutions) in three different ratios. In each combination, we progressively increased the malicious traffic and lowered the honest traffic. Because the payload size for both honest and malicious request is roughly the same (the only difference is that the malicious packets contain incorrect puzzle solutions), a total of *roughly* 75 requests per second is maintained. There was a 1 second break in between batches of traffic. The three different traffic combinations are given in Table I.

Profile	Total Batches	Type	Rate (req/s)	Successful Requests	Server Status
1a	20	Honest	25	498 (99.6%)	Alive
	10	Malicious	50		
1b	20	Honest	15	300 (100%)	Alive
	10	Malicious	60		
1c	20	Honest	5	99 (99%)	Alive
	10	Malicious	70		

Table I
SCENARIO 1.3 - PROFILE 1 TRAFFIC DETAILS AND RESULTS.

The rationale for our traffic profile is to show that the relatively high CPU usage in traffic profile 1a is due to processing honest requests, *not* due to puzzle solution processing. The results of our experiments are shown in Figure 4. As expected, given the roughly equivalent traffic rate, the CPU usage decreases as the number of honest requests decreases and the number of malicious requests increases. Table I also shows that a very high percentage of honest requests are served successfully without noticeable delay.

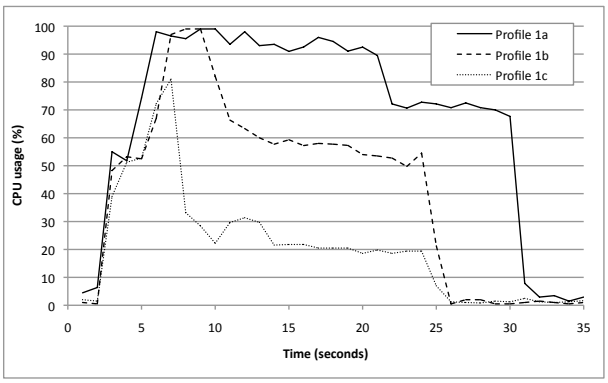

Figure 4. Scenario 1.3, Profile 1 – CPU usage.

Profile	Total Batches	Type	Rate (req/s)	Success Requests	Server Status
2a (no puzzle)	10	Honest	25	111 (44.4%)	Stall
	10	Malicious	100		
2b (w/puzzle)	10	Honest	25	250 (100%)	Alive
	10	Malicious	100		
2c (w/puzzle)	30	Honest	25	750 (100%)	Alive
	30	Malicious	100		
2d (w/puzzle)	30	Honest	25	748 (99.7%)	Alive
	30	Malicious	150		
2e (w/puzzle)	30	Honest	25	299 (39.8%)	Stall
	30	Malicious	180		
2f (w/puzzle)	30	Honest	15	449 (99.7%)	Alive
	30	Malicious	180		

Table II
SCENARIO 1.3, PROFILE 2 – TRAFFIC DETAILS AND RESULTS.

These results demonstrate the effectiveness and efficiency of client puzzles in weeding out malicious traffic while still maintaining acceptable service performance to honest users.

Profile 2: Stress Testing: In this experiment, we wanted to know how far we can push the server to maintain acceptable service performance to users in the presence of varying flooding attack intensity. We flooded a server with six different attack profiles as detailed in Table II. With the exception of profile 2f (further detailed below), all traffic profiles maintain the same honest request rate: roughly 25 requests per second per batch with 1 second pause in between batches of traffic. Shorter traffic profiles were used in profile 2a and 2b (only 10 batches), while longer ones were used in profiles 2c to 2f.

The results of our experiments are also shown in Table II. Obviously, given the same flooding attack rate, the client puzzles improve the success response rate significantly compared to when no client puzzles are used (compare results for profiles 2a and 2b). Furthermore, acceptable service performance to honest clients is maintained when the flooding rate increases to at least around 150 malicious requests per second. When the flooding rate reached 180 requests per second with 25 honest requests per second as well (profile 2e), less than half of the honest requests were served.

In practice, when the flooding rate increases, the server may increase its puzzle difficulty to lower the request rate. This is the scenario we depict in profile 2f in which it is assumed that server has increased its puzzle difficulty resulting in a reduction of the honest traffic rate to 15 requests per second. In this scenario, almost all honest requests were served successfully. We note that for most client puzzles, increasing puzzle difficulty does not increase the cost of solution verification.

4) Scenario 1.4: .NET WCF : We have experimented with the same client puzzles on a web service built on the .NET WCF platform. Experimental parameters vary from those used in Scenarios 1.1 to 1.3, however, the results clearly point to the same conclusions.

We first compared the CPU usage of the server when 10,000 requests are sent to a simple web service without client puzzles and the same service with client puzzles. The results indicated that, similar to Java Metro, no significant processing overhead is introduced when client puzzle is used (graph not shown due to space limitations).

Next, we investigated the extent to which CPU resources can be conserved when an attacker sends 10,000 malicious requests (manifested by the inclusion of random puzzle solution). Our results show that the .NET platform is able to reject requests with invalid puzzle solutions very efficiently resulting in a significantly lower CPU usage (<5%) in comparison to when no client puzzles is used (around 90%).

Finally, we ran a mixed traffic experiment in which 1,000 honest requests (with valid puzzle solutions) were sent to the server along with 10,000 malicious requests (with incorrect puzzle solutions). Similar to Scenario 1.3, our goal was to assess the ability of the server to serve honest client requests when it is under a flooding attack. In our experiment, a very high percentage of honest requests (about 94%) were served successfully in this mixed traffic scenario.

B. Experiment 2: Semantic Attack – Heavy Cryptography

This experiment aims to determine if client puzzles are an effective countermeasure against semantic web services attacks. For this experiment, we developed a Java Metro-based web service as the victim (similar experiment with the .NET platform is yet to be completed). The web service performs a simple service (adding two integers) but requires clients to use X.509 certificate-based mutual authentication with signature and encryption protections. We used the `wget` utility to send the attack traffic.

1) Scenario 2.1: Baseline Behaviour: Similar to Scenario 1.1, Scenario 2.1 attempts to show that the use of client puzzles does not introduce any significant overhead. 10 batches of 50 simultaneous honest requests (with a 1 second gap between batches) were sent to the Java web service (both without and with client puzzles enabled). The server's observed CPU usage, as expected, showed no significant

processing overhead introduced by the use of client puzzles (graph not shown due to space limitations).

2) Scenario 2.2: Heavy Cryptography – Attack Traffic Only: The attack payload used in this scenario consisted of a very large WS-Security SOAP header with several thousand digital signatures (approximately 20 MB payload). Because this is a semantic attack, we only send one request per second, just enough to sustain the effect of the attack on the server over an extended period of time.

In total, 50 requests with malicious payloads were sent (with a 1 second interval between requests) to a web service without client puzzles and one with client puzzles. Each request received a '500 Internal Server Error' response.

The results of our experiments show that when no client puzzle was used, our attack caused the server to spend over 30 minutes before returning the last '500' error. When the same attack traffic (but with an incorrect puzzle solution) was sent to the same server, the server detected the incorrect puzzle solution and returns an error with a much quicker response time: the 50 requests were all processed in about 10 minutes, a significant improvement compared to when no client puzzle was used.

3) Scenario 2.3: Heavy Cryptography – Mixed Traffic: In this scenario, we sent a mix of honest and malicious traffic to the web service, either without or with client puzzles. The honest requests consisted of 100 requests, each sent with a 1 second interval. The malicious request payload was the same as the one used for Scenario 2.2. The effects of the mixed traffic on the server were observed over a 15-minute period.

When client puzzles were not used, only about 25 honest requests were successfully served (out of 100 total honest requests, plus the 50 malicious requests per second).

In contrast, when the same mix was sent to the server with client puzzles in use, all 100 honest requests were successfully served in around 14-15 minutes. Note that in this latter case (with client puzzles), we in fact needed to increase the number of malicious requests so that the effect of the malicious requests would be sustained throughout the whole 15-minute duration of the experiment.

The distribution of inter-arrival time of the response payloads depicted in Figure 5 (calculated from each response payload timestamp), shows the reduction of the inter-arrival response time when client puzzles were used.

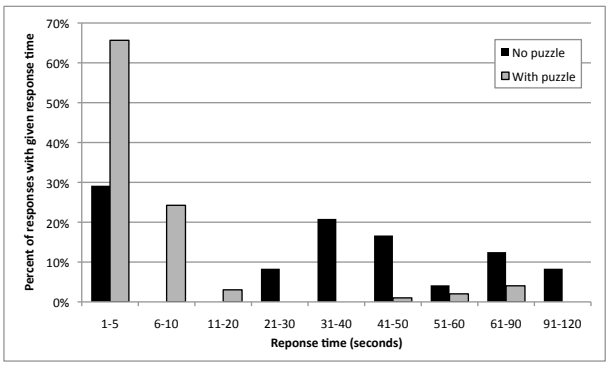

Figure 5. Scenario 2.3 – Response inter-arrival time distribution.

V. Observations and Limitations

From the experiments detailed in Section IV-A and Section IV-B, we can conclude that the integration of client puzzles in existing web services applications can provide some protection against DoS attacks.

The results obtained from our flooding attack experiments strongly suggest the effectiveness of the client puzzles in mitigating such an attack. Our experiments show that client puzzles can conserve CPU resources by discarding malicious traffic early in the process, sparing the server from exhausting its CPU resources to serve malicious or illegitimate requests. Consequently, a web service employing client puzzles can maintain an acceptable performance level for honest clients even when it is under attack.

However, an obvious condition for this technique to be an efficient solution is that the amount of resources needed to process the puzzle solution should be less than the resources needed to fully serve the request. While our experiments have shown that the overhead is low, there is still an overhead incurred, namely parsing the SOAP header and verifying the client puzzle solution. For a very simple web service (such as echoing a message or adding two integers), this client puzzle overhead may consume more resources than the actual web service. *Thus, client puzzles becomes increasingly effective as the resources required to fully execute a web service operation increase.*

Our experiments show that client puzzles can reduce the response time for an honest client request in the semantic attack scenario (Section IV-B). However, this is possible because the attacker in our experiment sent requests with incorrect puzzle solutions. In practice, it is possible for an attacker to spend some time to solve a puzzle correctly and still be able to send low-rate requests with correct puzzle solutions. This will force the server to accept the requests and to process the heavy cryptographic directives, resulting in the overwhelming of the server's resources (similar to the Scenario 2.2 experiment when no client puzzles were used). The server could increase the puzzle difficulty until the attacker cannot generate the rate of puzzle solutions required to sustain the attack. This will have the unfortunate effect of increasing the burden for honest clients, although honest clients will be served eventually. A signature-based XML firewall may be a more effective mitigation strategy for this type of semantic-based attack.

Finally, we have not fully addressed the issue of *freshness* of client puzzle solutions in our implementation. While the timestamp can be used as a method to check a solution's freshness, it does not prevent an attacker from pre-computing

a set of solutions with future timestamps. The standard way to address this problem is by periodically changing the value of the server's nonce N_S to restrict the window of opportunity for an attacker to pre-compute solutions. The use of interactive client puzzles can also guarantee the freshness of a solution and prevent pre-computation of puzzle solutions by an attacker since the server nonce is provided on a per-request basis. Further research is needed to assess how we can efficiently integrate interactive client puzzles into web services applications, and how well this technique can protect a server from DoS attacks. Considerations include the performance impact on the server and the correct mechanism for providing per-request client nonces, either through dynamic WSDL or through some other interactions.

VI. Related Work

There have been several works published in the area of DoS mitigation in web services. One proposed approach is to use a form of XML firewall to filter out malicious packets with the logic being that most web services DoS attacks stem from the vulnerability in the XML structure itself. For example, Gruschka et al. [21] proposed the use of tight XML schemas to prevent certain types of DoS attacks which exploit vulnerabilities in XML parsing. Bebawy et al. [22], and Loh et al. [23] proposed the use of an XML/Web Services firewall to perform some form of source-based filtering and SOAP/XML message inspection (based on some rules and policies) to filter out malicious packets. To address the potentially performance-taxing issues of source and content-based filtering approaches, Padmanabhuni et al. [4] proposed the use of a Patricie Trie data structure to create efficient representations of XML messages for analysis. It should be noted that XML/SOAP firewalls to counter various types of XML-based DoS attacks have already been widely supported in the industry, such as the IBM DataPower[3] and Vordell Gateway[4] products. Other work, such as Pinzón et al. [24] and Yee et al. [25], introduced novel heuristics into DoS mitigation by incorporating neural networks, clustering techniques, association and sequential rules, and fuzzy logic. The main goal of these approaches is to enable the reasoning, classification, and anomaly detection of various types of SOAP messages so that malicious SOAP messages can be rejected. A slightly different approach by Ye [26] was an architecture for web services that is resilient to DoS attacks by offloading the verification of message authenticity and validity to other agents, allowing web service providers to process only those requests that have been successfully authenticated and validated.

The main difference between the work described so far and our work is that most existing work focuses on analyzing the *structure* of the SOAP/web services messages to differentiate between 'good' and 'bad' requests. In other words, they address semantic DoS attacks which are assumed to be detectable by analyzing the structure of the messages. These works have not addressed the problem of flooding attacks in which requests often do conform to the definition of a 'good' SOAP message.

More importantly, most existing work does not address the *resource imbalance* issue that is critical in executing a successful flooding attack. In fact, distinguishing 'good' requests from 'bad' ones results in *even more work* by the web service provider, potentially aggravating the DoS problem. Our work aims precisely at addressing this resource imbalance issue. By doing so, we allow a server to focus its limited resources on legitimate client requests.

We *do not* claim that the related work mentioned above and our work are mutually exclusive. In fact, they are complementary: given the diversity DoS attacks, it only makes sense that one should use a variety of defensive techniques. It is possible, for example, for a server to first use client puzzles to mitigate flooding-style attacks, and then pass the requests to an XML firewall (or other SOAP analysis tool) to filter out semantic DoS attacks.

VII. Conclusions

We have examined the effectiveness of integrating non-interactive client puzzles into existing web services platforms to mitigate some DoS attacks. Based on our experiments, we conclude that client puzzles are effective in mitigating flooding attacks against web services. Client puzzles can also improve the response time for honest clients in some semantic attacks, but will work best against semantic attacks in combination with existing filtering techniques.

Future work includes expanding our experiments with other types of DoS attack and extending our server to *interactively* refresh its puzzle nonce value N_S to analyze the performance impact on both servers and honest clients. Comparing our results with transport-layer client-puzzle implementation is also another interesting future work.

[3]http://www-01.ibm.com/software/integration/datapower/

[4]http://www.vordel.com/products/gateway/index.html

References

[1] OASIS, *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*, OASIS, November 2006.

[2] W3C, *Web Services Policy 1.5 - Framework*, W3C, September 2007.

[3] M. Jensen, N. Gruschka, and R. Herkenhner, "A survey of attacks on web services," *Computer Science - Research and Development*, vol. 24, pp. 185–197, 2009, 10.1007/s00450-009-0092-6.

[4] S. Padmanabhuni, V. Singh, K. M. S. Kumar, and A. Chatterjee, "Preventing service oriented denial of service (PreSODoS): A proposed approach," in *Proceedings of the IEEE International Conference on Web Services*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 577–584.

[5] S. Suriadi, A. Clark, and D. Schmidt, "Validating denial of service vulnerabilities in web services," in *Network and System Security, International Conference on Network and System Security*. IEEE Computer Society, 2010, pp. 175–182.

[6] A. Singhal, T. Winograd, and K. Scarfone, *Guide to Secure Web Services - Recommendations of the National Institute of Standards and Technology*, National Institute of Standards and Technology, August 2007.

[7] M. Powell, "Defending your XML web service against hackers, part I," *MSDN*, 2001, http://msdn.microsoft.com/library/aa480513.aspx.

[8] Layer7 Technologies, "XML threats and web services vulnerabilities: Understanding risk and protection," 2005, http://www.soahub.com/Architecture/PDF/XML_Threats_Web_Services_Vulnerabilities.pdf.

[9] J. Nazario, "Political DDoS: Estonia and beyond," in *USENIX Security '08*. USENIX, July 2008, http://streaming.linux-magazin.de/events/usec08/tech/archive/jnazario/.

[10] J. Leyden, "Techwatch weathers DDoS extortion attack," *The Register*, 2009, http://www.theregister.co.uk/2009/01/30/techwatch_ddos/.

[11] J. Vijayan, "MasterCard SecureCode service impacted in attacks over WikiLeaks," *Computer World*, 2010, http://www.computerworld.com/s/article/9200541/MasterCard_SecureCode_service_impacted_in_attacks_over_WikiLeaks.

[12] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Advances in Cryptology – Proc. CRYPTO '92*, ser. LNCS, E. F. Brickell, Ed., vol. 740. Springer, 1992, pp. 139–147.

[13] A. Juels and J. Brainard, "Client puzzles: A cryptographic countermeasure against connection depletion attacks," in *Proc. Internet Society Network and Distributed System Security Symposium (NDSS) 1999*. Internet Society, 1999, pp. 151–165. [Online]. Available: http://www.isoc.org/isoc/conferences/ndss/99/proceedings/

[14] T. Aura, P. Nikander, and J. Leiwo, "DOS-resistant authentication with client puzzles," in *Security Protocols: 8th International Workshop*, ser. LNCS, B. Christianson, B. Crispo, J. A. Malcolm, and M. Roe, Eds., vol. 2133. Springer, 2000, pp. 170–177.

[15] J. Smith, J. González Nieto, and C. Boyd, "Modelling denial of service attacks on JFK with Meadows's cost-based framework," in *Proc. 4th Australasian Information Security Workshop – Network Security (AISW-NetSec) 2006*, ser. CRPIT, R. Buyya, T. Ma, R. Safavi-Naini, C. Steketee, and W. Susilo, Eds., vol. 54. Australian Computer Society, 2006, pp. 125–134. [Online]. Available: http://crpit.com/confpapers/CRPITV54Smith.pdf

[16] D. Stebila and B. Ustaoglu, "Towards denial-of-service-resilient key agreement protocols," in *Proc. 14th Australasian Conference on Information Security and Privacy (ACISP) 2009*, ser. LNCS, C. Boyd and J. González Nieto, Eds., vol. 5594. Springer, 2009, pp. 389–406.

[17] T. J. McNevin, J.-M. Park, and R. Marchany, "pTCP: A client puzzle protocol for defending against resource exhaustion denial of service attacks," Department of Electrical and Computer Engineering, Virginia Tech, Technical Report TR-ECE-04-10, October 2004. [Online]. Available: http://www.arias.ece.vt.edu/publications/TechReports/mcNevin-2004-1.pdf

[18] D. Dean and A. Stubblefield, "Using client puzzles to protect TLS," in *Proc. 10th USENIX Security Symposium*, 2001. [Online]. Available: http://www.usenix.org/events/sec01/dean.html

[19] J. Rangasamy, D. Stebila, C. Boyd, and J. González Nieto, "An integrated approach to cryptographic mitigation of denial of service attacks," in *6th ACM Symposium on Information, Computer and Communications Security (ASIACCS) 2011*. ACM, 2011, pp. 114–123.

[20] D. Stebila, L. Kuppusamy, J. Rangasamy, C. Boyd, and J. Gonzalez Nieto, "Stronger difficulty notions for client puzzles and denial-of-service-resistant protocols," in *Progress in Cryptography – CT-RSA 2011: The Cryptographers' Track at RSA Conference 2011*, ser. LNCS, A. Kiayias, Ed., vol. 6558. Springer, 2011, pp. 284–301.

[21] N. Gruschka and N. Luttenberger, "Protecting web services from DoS attacks by SOAP message validation," in *Security and Privacy in Dynamic Environments*, ser. IFIP International Federation for Information Processing, S. Fischer-Hbner, K. Rannenberg, L. Yngstrm, and S. Lindskog, Eds. Springer Boston, 2006, vol. 201, pp. 171–182, 10.1007/0-387-33406-8_15.

[22] R. Bebawy, H. Sabry, S. El-Kassas, Y. Hanna, and Y. Youssef, "Nedgty: Web services firewall," in *IEEE International Conference on Web Services*. Los Alamitos, CA, USA: IEEE Computer Society, 2005, pp. 597–601.

[23] Y.-S. Loh, W.-C. Yau, C.-T. Wong, and W.-C. Ho, "Design and implementation of an XML firewall," in *2006 International Conference on Computational Intelligence and Security*, vol. 2, November 2006, pp. 1147 –1150.

[24] C. Pinzón, J. De Paz, J. Bajo, and J. Corchado, "A multiagent solution to adaptively classify SOAP message and protect against DoS attack," in *Current Topics in Artificial Intelligence*, ser. Lecture Notes in Computer Science, P. Meseguer, L. Mandow, and R. Gasca, Eds. Springer Berlin / Heidelberg, 2010, vol. 5988, pp. 181–190, 10.1007/978-3-642-14264-2_19.

[25] C. G. Yee, W. H. Shin, and G. S. V. R. K. Rao, "An adaptive intrusion detection and prevention (ID/IP) framework for web services," in *Proceedings of the 2007 International Conference on Convergence Information Technology*, ser. ICCIT '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 528–534.

[26] X. Ye, "Countering DDoS and XDoS attacks against web services," in *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*. Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 346–352.