KTH Computer Science
and Communication

Analysis of the Client Puzzles protocol

Authors:
Andreas Gabrielsson
andregus@kth.se
8912310037

Simon Österman
simost@kth.se
8802050156

Supervisor:
Mads Dam
Examiner:
Mårten Björkman

Bachelor's Thesis at CSC

May 21, 2012

Abstract

This paper covers a certain proof of work protocol known as the client puzzles. The client puzzles is placed upon the protocol it is supposed to protect and is specifically designed to protect against connection depletion attacks. Our study is to determine how well the client puzzles protocol prevents connection depletion attacks and how it affects other parts of the system. To do this we choose to implement our own version of the client puzzles protocol and to see how it performs as well as read up on what other people has learned about its strengths and flaws. After implementing and trying with different sized puzzles we could determine that the client puzzles actually could provide some protection against connection depletion attacks though it also became clear that the protocol has some other issues. These flaws include increased vulnerability to distributed denial of service attacks by solving large amounts of puzzles on the clients, denial of service attacks by just requesting puzzles without solving them.

Our conclusion of the client puzzles protocol is that while the protocol could solve the security issue it is supposed to, it provides with even more new problems. That combined with the fact that it needs software on all clients makes it a quite bad solution.

Statement of Collaboration

The implementation of the program was done together.

We have done almost everything in collaboration but every part has had a main contributor and that is the name shown in the table below:

Section	Andreas Gabrielsson	Simon Österman
1		x
1.1		x
1.2		x
2.1		x
2.2	x	
2.3	x	
2.4	x	
2.5		x
3	x	
3.1	x	
3.2	x	
3.3	x	
4	x	
4.1.1	x	
4.1.2	x	
4.2		x
5	x	x

Contents

1 Introduction

We have chosen to study the proof-of-work concept. There exists many different proof-of-work protocols designed to protects against different kinds of denial of service attacks but we choose to study the client puzzles protocol which is specifically designed to protect against the connection depletion attack. There have been some studies on the subject and both positive and sceptical reports have been published about the protocol which is why we found it interesting and feasible to make our own contribution.

1.1 Problem statement

We are going to study the strengths and weaknesses of the client puzzles protocol. We want to find out how it helps to prevent connection depletion attacks as well as how it affects the security overall. In particular, we want to find its strengths and weaknesses against commonly used attacks such as distributed denial of service attacks and IP-spoofing attacks. We also want to know whether the protection has any other effects than just the protection of the underlying protocol and if it is a protocol worth adapting.

1.2 Terminology

- IP-Spoofing - Means that the client changes the source IP-address to another than the client's.
- Denial of Service (DoS) - A general name for an attack that drains one or more of the servers resources such as memory or bandwidth.
- Distributed Denial of Service (DDoS) - The same as DoS except that it is distributed amongst a net of computers controlled by the actual attacker(s). The controlled computers are sometimes referred to as zombies.
- Bitstring - A sequence of bits

2 Background

2.1 Proof-of-work

Proof-of-work is a concept based on making a client that is requesting a service show good will by performing some computations to grant access to the service. The idea of proof-of-work is to prevent denial-of-service attacks by forcing the attacker do an excessive amount of computations compared to the provider while the server is under attack. However, when the server does not detect an attack, the proof-of-work protocol does not affect the users. Whether it is a good concept for preventing denial-of-service attacks

such as email-spams or connection depletion is subject to discussion though. While it could prevent some DoS attacks, it also forces the legitimate users to make some additional work each time a service is provided [1]. There are a number of different approaches to the proof-of-work concept, designed to prevent various kinds of DoS attacks.

2.2 Connection depletion attacks

A connection depletion attack is an attack that exhausts some resource during the establishment of a connection using some vulnerability in the protocol used [3].

A TCP SYN flood attack is an example of a connection depletion attack. The TCP SYN flood attack exploits a vulnerability in the TCP three-step handshake [5]. When a TCP connection is established, the client sends a SYN packet to which server responds with a SYN-ACK and place the connection in a buffer for half-open connections waiting for an ACK from the client to complete the handshake. A SYN flood attacker sends a lot of SYN packets to the server and does not acknowledge the connection with an ACK leaving the connections half-opened which hopefully, from the attackers point of view, will fill the buffer for half-open connections within the server so it can not accept any new connections [5].

There are other protocols vulnerable to this kind of attacks as well. The TCP SYN flood attack exploits a memory resource but it's also possible to exhaust other resources. For example, encryption protocols like TLS are vulnerable to a connection depletion attack that exhaust the CPU since encryptions and decryptions are very computational intense for the CPU [2]

2.3 Client Puzzles Protocol

The client puzzles protocol was first announced by Ari Juels and John Brainard in 1999 [3]. This protocol is made to protect any protocol that is vulnerable to connection depletion attacks and it is placed on top of the vulnerable protocol in the OSI-model [11]. An attack is discovered by monitoring the resource in the protected protocol that is vulnerable to an attack. There is a number called ”maxcon” which depicts the maximum number of connections in that resource during a normal execution. When a connection is established in the client puzzles protocol, the client first asks whether it should solve a puzzle. Under normal conditions the server responds ”no” and grants access to the underlying protocol as usual, see figure 1. If the ”maxcon” limit is exceeded, the protocol starts sending puzzles because it then senses that it may be under attack. For a client to establish a connection to the server, it has to solve this puzzle and hand over the solution to the server within a limited time. The server then verifies the puzzle and then, if correctly solved, grants permission to the protocol under protection for a

limited time, see figure 2.

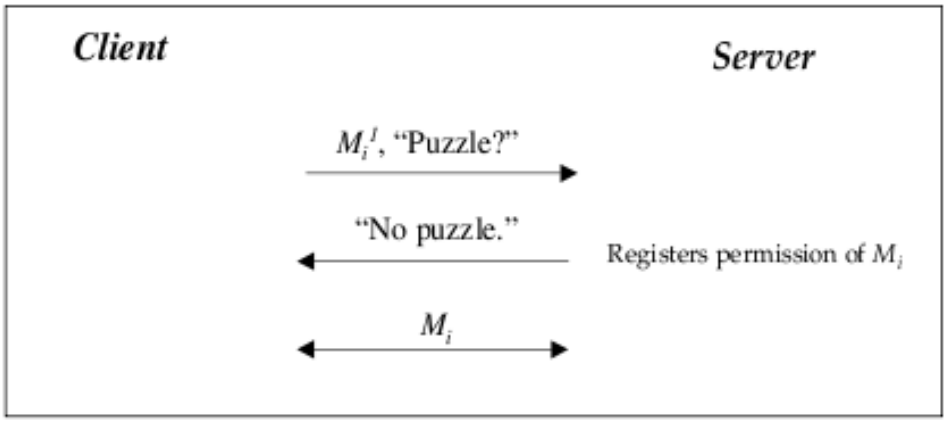

Figure 1: Handshake when no puzzle should be solved

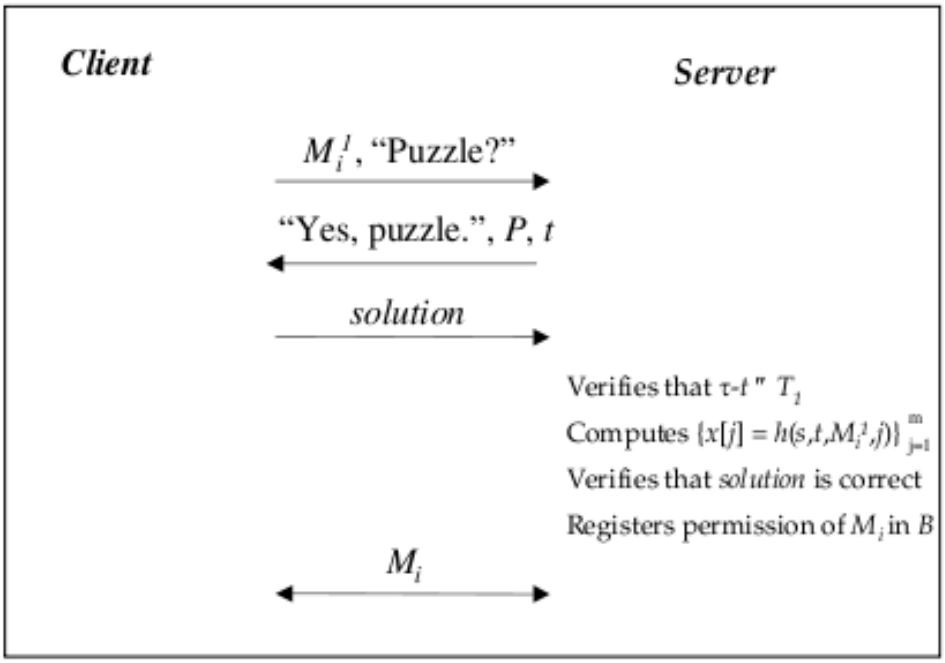

Figure 2: Handshake when client needs to solve a puzzle

2.4 The puzzles

In this section we will describe the puzzles, how they are generated, solved and validated. Everything is based on the description in [3], that description is also more detailed than ours.

The puzzles rely on a partial hash inversion technique. Let P be a puzzle, x a value that is used to generate the puzzle and h a hash function that is not invertible and that produces hash values of the same length, x will be explained in more detail later. We first construct y so that $y = h(x)$ and then $z = h(y)$. A sub-puzzle consists of a part of y where k bits are left out and the entire z. The solution to the puzzles is the k bits of y that was left out. P consists of n such sub-puzzles. In that way, it is easy to control the size of the puzzles without any bigger adjustments, you just add more sub-puzzles to P. An explanation why this is better than adjusting the size by just make it longer is provided in [3].

The value x that is used to produce the puzzle consists of a bit-string s that is known only to the server, the current time stamp t, the first message m sent by the client and a value i that is unique for each sub-puzzle in P. So $x = s.t.m.i$ where . means bit-string concatenation. Along with P, t is also sent to the client. That is to make the puzzles stateless on the server and this will be explained in more detail later. How a sub-puzzle is constructed can be seen in figure 3.

When the server has generated and sent back P and t the client has to solve the puzzle. Since h is not invertible this has to be done by brute force [3]. The client simply tries a value v of length k and concatenate them with y, then it computes $h(v.y)$ and compare to z. If they match v is the solution to the sub-puzzle, again . means bit-string concatenation. For each guess one hash value has to be computed and the number of guesses that has to be done depends on the value k. The maximum number of guesses is 2^k and the average is $2^k/2 = 2^{k-1}$, since P consists of n puzzles the average time to solve a full puzzle is $n \cdot 2^{k-1}$. Along with each value v for every sub-puzzle the solution that is sent back to the server consists of t and the message m that was sent to the server when asking for a puzzle.

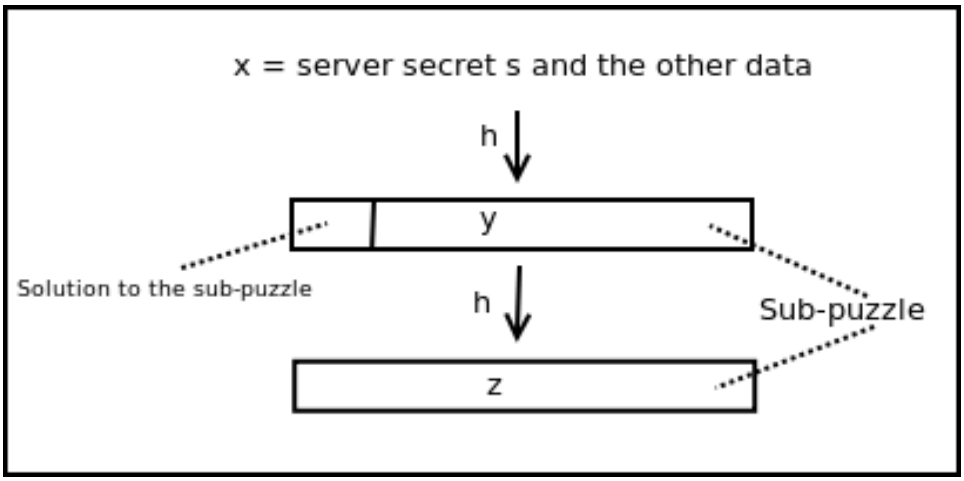

Figure 3: Construction of a sub-puzzle

Before the server grants permission to the underlying protocol it has to verify the solution. As mentioned before, the client has limited time to solve the puzzle so it has to be checked that t plus the time limit is bigger than the current time stamp. Then $h(s.t.m.i)$ is computed and it is verified that the k bits that should be left out matches v for every sub-puzzle. One hash value needs to be computed to verify each sub-puzzle so the total number of hash computations needed is n.

The reason that the server sends t along with P and that the client sends back t and m along with the solutions is to keep this protocol stateless. This is necessary because the reason for this protocol is to prevent filling up the buffer holding the half-opened connections on the server. If the protocol was not stateless it had to keep information about the puzzle sent and this would introduce another weakness that would be vulnerable to connection depletion attacks as well which would compromise the protection by the client puzzles protocol [3].

There are a few assumptions made for the protocol to properly protect the system. First we assume that the attacker does not have sufficient resources to perform a DoS attack by exhausting the bandwidth or a single port by sending a big volume of packets. If that were the case, a connection depletion attack would not be necessary. Second it is assumed that the attacker can spoof the source IP-address. There are also a few other assumptions which can be find in [3].

For client puzzles to protect against denial of service attacks properly they must satisfy a couple of requirements that is mentioned by Laurent and others in [4]:

- The computations of the server, when creating and verifying the puzzles, must be significantly less than the computations of the client.
- It should be easy to adjust the difficulty and sizes of the puzzles.
- There must be a time-limit for the clients to solve the puzzles.
- It should not be possible to pre-compute puzzle solutions.
- Puzzles already solved should not provide help in solving a new puzzle.
- The server should not keep a record of the connections state before accepting a solution.

2.5 Other techniques

One problem with this protocol that is mentioned in [3] is that the client obviously needs software to handle this protocol as opposed to other methods to prevent connection depletion attacks such as syn-cookies [6] or random

dropping [12]. The random dropping approach drops half-opened connections at random when the connection slots are filled up to make sure that there is always resources for another client [12]. The obvious problem with this during an attack is that is as likely to drop a connection to a legitimate client as it is to prevent the attackers since it has no way to detect whether a client is an attacker or not. The advantage with the protocol on the other hand is that the connection slots never gets completely depleted and does not force neither the clients or server to make additional computations.

3 Method

We have studied the strengths and weaknesses of the client puzzles protocol. To help ourselves in that task, we have implemented a Java program that generates puzzles, solves puzzles and verifies that a solution is correct. We have also found articles and scientific reports that is relevant for analysis of the client puzzles protocol in a general computer security context.

3.1 Implementation

Our implementation is somewhat simplified since some parts are just not relevant to be able to analyze the result. First of all, we have only implemented the puzzles with it's generator, solver and verifier without any underlying protocol or clients. Another simplification is that for the x value that is used to generate a puzzle we will only consist of the server secret s and the index bitstring i since the time stamp and the message from the client will not be necessary for the analysis since there are no time limits or clients in our implementation. The program measures the duration of each step and we use that to present statistics on how much work the server need to do to generate a puzzle and verify the solution and how much work the client has to do to solve the puzzle. The ratio between the two are an important aspect since it might be possible to exhaust the processor of the server if a client can solve the puzzles to quickly. The workload of the server alone is important because it must be able to handle a lot of puzzle requests at the same time. We will try different sizes of the puzzles by modifying the number of sub-puzzles and the number of bits left out and try to analyze what might be a convenient puzzle size in a real client puzzles implementation and also how the server can vary the puzzles to make them more time consuming for the client if necessary.

The program measures the wall clock time in every step using the Java api method System.nanoTime() which returns the current time in nano seconds, that precision is needed since the duration of the steps is very small so we need a precise time to be able to measure the duration. The reason we measure wall clock time is that the built in methods for measuring cpu time of a thread has different accuracy on different Java virtual machines [7] and

some virtual machines do not even support cpu time measurement. We concluded that this fact is a bigger problem than using the wall clock time.

To calculate the hashes, the built-in class MessageDigest [9] was used and the hash function used is sha-256 since that is the hash function recommended by NIST and it meets all requirements specified in section 2.3. Juels and Brainard [3] mention the MD4 algorithm as a good algorithm and it might still work. We have not used that since it is an old algorithm that is not used any more. Also using the sha-256 algorithm our puzzles will perform more like a modern implementation of the client puzzles protocol would.

All executions have been done on a computer with an *Intel(R) Core(TM)2 Quad CPU Q9550 @ 2.83GHz* processor. Operating system is Ubuntu 10.04 LTS and the Java version used is *1.6.0_20* on virtual machine *OpenJDK Runtime Environment (IcedTea6 1.9.13) (6b20-1.9.13-0ubuntu1 10.04.1)*.

3.2 General analysis

The statistics have been combined with an analysis in a more general computer security context. Does the overall protocol do its job in an efficient way? What are the weaknesses and do the client puzzles protocol open any other security vulnerabilities that Juels and Brainard did not consider?

3.3 Literature review

When looking for references we have been trying to find the most credible reports and articles on the subject. We have been looking around for many reports and have chosen the ones that are most relevant and cited by many other authors.

This reports analyzes the protocol announced in a report by Juels and Brainard [3]. At the time of the report they worked at RSA Laboratories, both this report and many other reports they have written have been cited in a lot of reports and they are recognized researchers within the field.

All of our other reports is also written by well known and respected names so we consider them credible.

We have also made a few references to for example the Java api and the reason for that is mostly to clarify what is being written about.

4 Result

Here we will state the strengths and weaknesses that we discovered of the client puzzles protocol as well as the results from our own implementation of the protocol. Below are our results in some specific and important aspects of the client puzzles protocol.

4.1 Implementation

In this section we will show the results from the implementation described in section 3.1. The statistics will focus on how the performance scale when increasing the size of the puzzles and how big the workload on the server is compared to the client.

4.1.1 Example output

First of all, we will show how an output from the program might look like to make it clearer how the program works and looks like.

The puzzle generator produces a number of sub-puzzles which looks like the one in figure 4. $GenerValue$ is the server secret concatenated with the index bitstring and $yFull$ is the first hash-value $h(generValue)$. The sub-puzzle consists of y and z which means the same as in section 2.3. y is yFull with, in this example, 16 bits left out from the beginning and z is the hash value of $yFull$.

```
generValue: 426305d23afb6789d840e47ffdbbf300000000
yFull: bb85bcab07647fcbe40e8bebd8e1c9ccacd9569a132cca948b7aef39b8adbc29
puzzle: y: bcab07647fcbe40e8bebd8e1c9ccacd9569a132cca948b7aef39b8adbc29
z: a9b2ffb04a481ffa3dffb55ce7ee1d691784f7beb3cbd52de80d9fac1d340fce
```

Figure 4: A sub-puzzle from the puzzle generator

The solver's output looks like in figure 5. It is simply the solution to the sub-puzzle with index 0. By comparing figure 4 and 5 we can see that the solution is actually the correct one.

```
solution 0: bb85
```

Figure 5: A sub-puzzle from the puzzle generator

The output from the verifier is not that interesting since it just prints whether the solution was correct or not.

4.1.2 Results from implementation

In this section we will present the results from the executions in graphs and tables.

Figure 6 and 7 shows the server workload and the client workload in milliseconds as the number of sub-puzzles increases. 16 bits are left out for the solver to solve. As we can see, both the server and client workload grows almost linearly. That is an expected result since the time needed for each sub-puzzle is expected to be constant. In both cases we have done a linearly

regression which shows us that the gradient on the server is approximately 0.02 and on the client 21.8 ms per sub puzzle. This means that the workload of the client increases about 1000 times more on the client than the server. That is a very good ratio according to the requirements on puzzles stated in section 2.3 that the workload on the server has to be significantly less than the client. It also tells us that it would be meaningful to increase the number of puzzles to a client if the server is under a heavy attack to make the workload on the client even bigger.

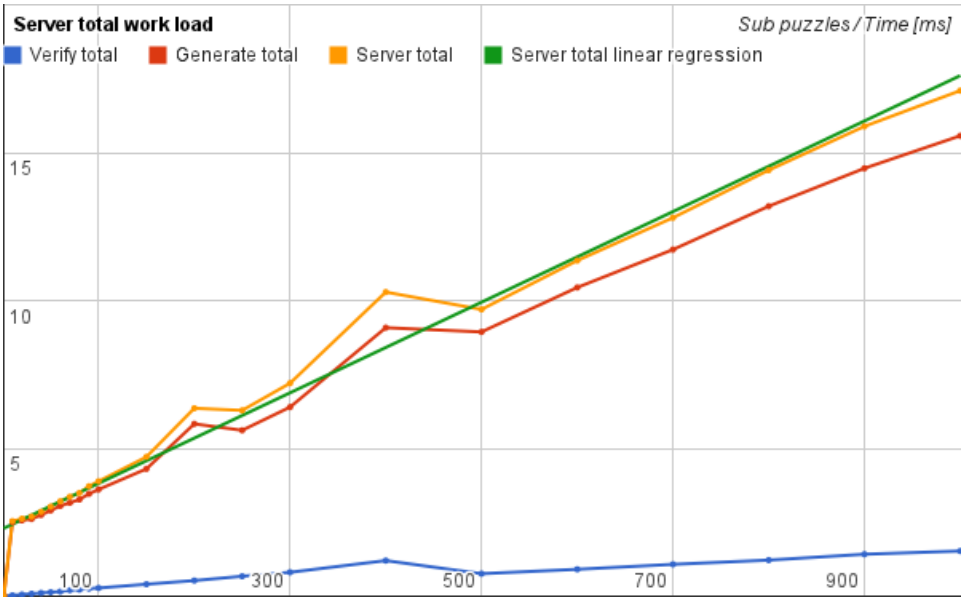

Figure 6: Workload on server when the number of sub-puzzles increases

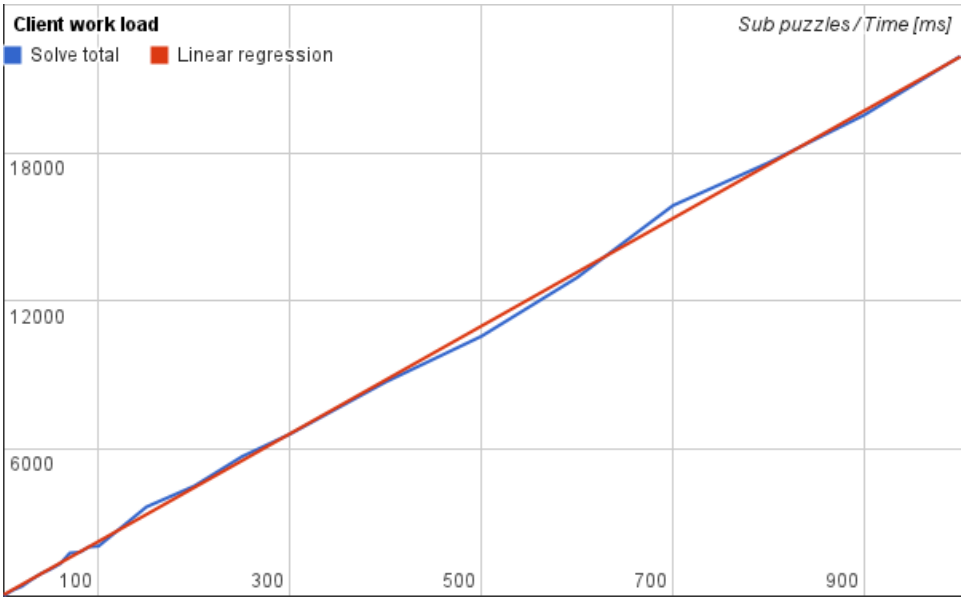

Figure 7: Workload on client when the number of sub-puzzles increases

The ratio above tells us that when the puzzle size is growing, the clients workload increases a lot quicker than for the server. Although that is very good, it does not give us a complete picture of the performance. If we take a quick look in figure 7 we can see that the time taken to solve the puzzles are far too long even with a hundred sub-puzzles which takes about two seconds. As stated in section 2.1, a crucial point with any proof-of-work protocol is that a legitimate user should not notice that any work has been proved and that would not be the case if it takes as long as two seconds. A reasonable time to accomplish that could be around half a second maybe less, maybe more depending on the magnitude of the attack the server is under. In our implementation, this is the case when a puzzle consists of 10 to 30 puzzles which therefore is a reasonable size.

There is another way of modifying the workload of the client; to change the number of bits left out. Table 1 shows the workload of client and server when increasing the number of bits left out using 10 sub-puzzles. To meet the requirements in section 2.1 it seems reasonably to leave out about 16 bits since eight bits makes the workload on the client too small compared to the server and leaving 24 bits makes it unreasonably big. That the time increases so rapidly is expected since the number of possible solutions is 2^n where n is the number of bits left out [3]. Increasing the size of the solution with one bit doubles the expected workload on the client which makes it more difficult to control than changing the number of sub-puzzles, hence that is a better approach. More detail on why that is better can be found in [3]

Number of bits left out	Server total [ms]	Client total [ms]
8	2.90	8.74
16	2.37	298.41
24	2.43	53808.62
32	Could not be solved within reasonable time	

Table 1: The workload on the server and the client when increasing the number of bits left out

4.2 Analysis on client puzzles protocol

So does the puzzles meet the requirements specified in section 2.3? We have already stated that the workload on the client compared to the server is very good. It does meet the requirements well, it is very easy to adjust the size of a puzzle by just changing the number of sub-puzzles, the server does not store any data about the state of the connection and there is a time limit. The server secret makes sure that no puzzles can be precomputed if we assume that no one but the server knows the server secret. That is a reasonable assumption to make since the only way to find the server secret is by brute force which is infeasible to do. The security of the hash function makes sure that solved puzzles does not help solving new puzzles. From this point of view this protocol do provide a good protection against connection depletion attacks.

We are looking at this protocol in a more general computer security context and there are actually other kinds of attacks than connection depletion attacks. We have found that this protocol not only do not provide any security against such attacks but in some cases it even makes these attacks easier. We are going to discuss this below.

Even if the ratio between the server and the client workload is quite good, the server still need to perform an amount of work to generate puzzles which is significantly bigger than to just accept a connection without puzzles, we consider this a problem. Since this protocol is stateless, the server can not distinguish a client that already has received a puzzle from one that has not. That makes it possible for a client to send a lot of puzzle request forcing the server to produce a lot of puzzles that will never be used and the client do not even have to solve them. This would in fact be a connection depletion attack itself and is similar to the attack on the TLS protocol since it tries to exhaust the processor. In figure 6 we can see that it takes about two point five milliseconds to produce 10 sub-puzzles, that is not very much work so this vulnerability is probably smaller than the one in TLS, but it still exists. To produce a solution to this problem would be very difficult since the protocol do not store data about the state of the connections. Even if we could do that it would not be much of a protection since IP-spoofing attacks would make it useless.

The Distributed Denial of Service (DDoS) attacks could pose a real threat to the client puzzles protocol. When an attack is distributed amongst several impersonated computers (zombies), there is an actual possibility for the attacker to be able to exploit the vulnerability of the protocol under protection even though the client puzzles is in effect. Although one client has a hard time filling for example the TCP buffer for half open connections, thousands or ten thousands of clients could easily solve all the puzzles needed within the time limit and still be able to fill the buffer. This is of course a major problem since this is what the client puzzles should be good at.

A countermeasure against this would be to send increasingly difficult puzzles to the client, as the connection buffer is getting more saturated. But as the server can not determine whether a client is a genuine user or an attacker, the puzzles to all the clients will get increasingly difficult which eventually could lead to the clients not being able to solve the puzzles which means that genuine users would get a denial of service [4] which of course also is not a very good result.

Another big problem that actually concerns proof-of-work protocols in general is the question, how much work should be proved? Laurie and Clayton [1] writes about this problem with proof-of-work in an e-mail spam context but it is highly relevant for us as well. We have done our measurements and it is quite easy to decide what is a good size of a puzzle for that computer but computation power could vary a lot between different computers. Cell phones and maybe other devices should also be able to connect to a server and they have a lot less computation power. Some devices may not be able to solve a puzzle within the time limit and some may be able to solve it too quick. So how to decide this? Laurie and Clayton claims that it is impossible and although we have tried to find a solution we have not been able to do so.

One big problem with the client puzzles protocol is that it needs software on the client and would have to be implemented everywhere the client puzzles protocol is used. Juels and Brainard [3] mention this in their report but they do not consider it as a big problem. However, we do since it would require a big commitment to make the client puzzle protocol usable.

5 Conclusions

Having studied the client puzzles protocol and implementing our own version of the puzzles we have found a lot of strengths and weaknesses with the protocol. Its main strength is that it actually do provide a level of protection against connection depletion attacks which is what it is designed for. One big problem though, is that it is so focused on connection depletion attacks that many other computer security aspects are forgotten. For example, the protocol opens up other vulnerabilities and it does not even provide a good

protection for a distributed connection depletion attack.

The question concerning how much work needs to be proven is still relevant and unanswered. This is a major problem that can deny the service for legitimate users.

All clients would need to have software support for this protocol and this software would have to be implemented everywhere a TCP connection, for example, is used. To implement this would of course be a very big effort. That fact together with the fact that the problem with connection depletion attacks is very limited and would by no means provide a DoS resistant server makes it unreasonable to think that the client puzzles protocol will be widely used in the future unless some major improvements are made. Also there are other solutions that provides somewhat good protection that requires no client software and it is possible that they are to be preferred over client puzzles.

All in all, we have come to the conclusion that the client puzzles just is not worth it considering the kind of limited protection it provides.

References

[1] Laurie, Ben; Clayton, Richard (2004) *"Proof-of-Work" Proves Not to Work*
URL: http://cl.cam.ac.uk/~rnc1/proofwork.pdf (Fetched 2012-04-09)

[2] Dean, Drew; Stubblefield, Adam *Using Client Puzzles to Protect TLS*
URL: http://static.usenix.org/events/sec2001/full_papers/dean/dean.pdf (Fetched 2012-04-11)

[3] Juels, Ari; Brainard, John (1999) *Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks*
URL: http://www.hashcash.org/papers/client-puzzles.pdf (Fetched 2012-04-09)

[4] Laurens, Vicky; El Saddik, Abdulmotaleb; Nayak, Amiya (2006) *Requirements for Cient Puzzles to Defeat the Denial of Service and the Distributed Denial of Service attacks*
URL: http://www.ccis2k.org/iajit/PDF/vol.3,no.4/8-abdulmotaleb.pdf (Fetched 2012-04-10)

[5] Stallings, William; Brown, Lawrie *Computer Security: Principles and Practice*

[6] Bernstien, D.J *SYN cookies*
URL: http://cr.yp.to/syncookies.html (Fetched 2012-04-09)

[7] *Java Api ThreadMXBean class*
URL: http://docs.oracle.com/javase/6/docs/api/java/lang/management/ThreadMXBean.html#getCurrentThreadCpuTime() (Fetched 2012-04-09)

[8] Oracle *Java Api System.nanoTime()*
URL: http://docs.oracle.com/javase/6/docs/api/java/lang/System.html#nanoTime() (Fetched 2012-04-09)

[9] Oracle *Java Api MessageDigest*
URL: http://docs.oracle.com/javase/6/docs/api/java/security/MessageDigest.html (Fetched 2012-04-09)

[10] NIST (2006)*Nist's policy on hash functions*
URL: http://csrc.nist.gov/groups/ST/hash/policy.html (Fetched 2012-04-09)

[11] The international telegraph and telephone consultative committee (1991) *X.800: Security Architecture for Open Systems Interconnection for CCITT Applications* URL: http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.800-199103-I!!PDF-E&type=items (Fetched 2012-04-11)

[12] Karig, David; Lee, Ruby (2001) *Remote Denial of Service Attacks and Countermeasures*
URL: http://palms.ee.princeton.edu/.../karig01remote.pdf (Fetched 2012-04-11)