Chained Puzzles: A Novel Framework for IP-Layer Client Puzzles

Timothy J. McNevin*, Jung-Min Park*, and Randy Marchany†
*Advanced Research in Information Assurance and Security (ARIAS) Laboratory
*Bradley Department of Electrical and Computer Engineering
†Virginia Tech Information Technology Security Office
Virginia Polytechnic Institute and State University
{tmcnevin, jungmin, marchany}@vt.edu

Abstract

Large-scale, high-profile Distributed Denial-of-Service (DDoS) attacks have become common recurring events that increasingly threaten the proper functioning and continual success of the Internet. Recently, client puzzle protocols have been proposed as a mitigation technique for DoS attacks. These protocols require a client to solve a cryptographic "puzzle" before it receives any service from a remote server. By embedding the client puzzle mechanism into the lowest layer of the Internet protocol stack that is vulnerable against network DoS attacks—the network layer—we can mitigate the most virulent form of DoS attacks: flooding-based DDoS attacks. This paper describes the framework of a novel IP-layer client puzzle protocol that we call Chained Puzzles. We describe the framework in detail and show its effectiveness using simulation results

1. Introduction

The frequency and scale of Denial-of-Service (DoS) attacks have steadily increased and now pose a considerable threat to the proper functioning and continual success of the Internet. In particular, large-scale Distributed DoS (DDoS) attacks pose the greatest threat to the Internet and e-businesses that rely on the availability of the Internet. These attacks flood a victim server or network with a large number of packets, thus bringing down the server or rendering it unreachable by legitimate clients. A DDoS attack typically consists of an attacker (or multiple attackers), several handler computers, and many attack zombies. During the DDoS setup phase, the attacker(s) infiltrate third-party machines in order to convert them into handlers or zombies. To initiate the attack, the attackers notify the handlers to invoke the zombies to begin flooding the victim with large amounts of useless data. Due to the large volume and high rate of traffic, this attack is commonly referred to as a *flooding attack*. The primary goal of the attack is to incapacitate the victim, and the secondary goal is to consume bandwidth and processing resources of the intermediate routers. It is very difficult to apply packet filtering schemes to mitigate a well-planned DDoS attack because it is virtually impossible to distinguish between legitimate traffic and attack traffic. Although ingress filtering [1] has been successful at combating attackers from using zombies with spoofed source IP addresses, the current trend of DDoS attacks indicates that more must be done to counter these attacks.

DDoS attacks are difficult to defend against because they do not exploit the vulnerabilities of a particular system or protocol. Instead, they exploit the fundamental service paradigm of the Internet: it takes more resources for routers and servers to process received packets than for clients to send those packets. Client puzzles change this paradigm: enable those providing service to push load back onto those requesting service.

To effectively counter DDoS attacks, one must understand that the location in which the countermeasure is placed is crucial to its effectiveness. As noted by Chang [2], the flow of packets caused by a DDoS attack resembles water passing through a "funnel": attack packets generated from a large number of sources are generated at the wide end of the funnel, and the victim is located at the narrow end, receiving all the attack packets coming from the wide end. It is obvious that DDoS attacks can be most readily detected at the victim network since all attack packets can be observed there. For mitigation, the opposite is true. It is most effective to filter or rate limit attack packets as close to the attack source (the zombie) as possible.

In this paper, we propose a novel network-layer client-puzzle protocol that we call *Chained Puzzles* (*CP*). CP effectively mitigates large-scale DDoS attacks by using a client puzzle mechanism at the IP layer. A client puzzle is a cryptographic or computational problem that a server presents to a client (legitimate or not) before the server provides services. Clients solve the puzzle by using a brute-force strategy that requires both time and computational resources. When a client attempts to send a large number of packets towards a victim (as in a flooding attack), it will be forced to solve a correspondingly large number of puzzles.

Integrating a puzzle protocol with IP poses several difficult challenges. A client puzzle protocol is a connection-oriented protocol that requires a three-way handshake between a puzzle solver (i.e., client) and a puzzle generator/verifier (i.e., "puzzle

0-7803-9305-8/05/$20.00 ©2005 IEEE

server"). Because IP is an inherently connectionless protocol, integrating the two protocols is a challenging task.

The main contribution of this paper is the proposal of a novel IP-layer client puzzle protocol, Chained Puzzles. CP has several advantageous features, including: (1) seamless integration of the puzzle handshake procedure with IP by using "chained" puzzle solutions, (2) inclusion of security measures to thwart protocol circumvention, (3) use of a novel puzzle algorithm based on a lightweight block cipher that enables a puzzle server to rapidly verify puzzle solutions at or near line speed, and (4) rate limiting of attack packets at the earliest entry point into the network.

2. Related Work

Client puzzles have been proposed previously in the literature to combat DoS attacks [3, 4, 5, 6, 7, 8, 9, 10, 11]. The basic idea behind a client puzzle is to have a client prove its legitimacy by devoting some its time and resources before a remote host will perform any action. Unlike other packet filtering schemes, client puzzles is a technique that does not need to distinguish between legitimate traffic and attack traffic. Instead, client puzzles rate limit *all* incoming traffic, including attack traffic, by requiring each client to solve puzzles to receive service. Recently, client puzzle schemes integrated into the IP layer have been proposed to mitigate flooding attacks [9, 10].

Congestion Puzzles by Wang and Reiter [10] is a recently proposed IP-layer puzzle scheme. When the puzzle mechanism is activated, each client is required to solve puzzles before their packets are forwarded by a congested router. Clients continuously send separate *probe packets* along with regular data packets. When a router downstream detects congestion, it relays the probe packets toward the destination by changing the ICMP code number to resemble a ping when it reaches the victim. This packet will be modified to contain the puzzle information (i.e., a nonce and the difficulty level). When a client receives the challenge, it begins to continuously solve puzzles and embeds the solutions in separate ICMP packets. The client takes the nonce it received from the router, creates its own nonce and uses both of those items to create the puzzle. Therefore, the client does not need to contact the congested router to get a new puzzle. The client sends the solutions, embedded in ICMP packets, towards the destination which are later intercepted by the router for puzzle verification. After correct verification of the puzzles, tokens are added into a *token bucket* at the congested router. When a data packet arrives, tokens are removed from the bucket. While each client is sending data packets and puzzle solution packets, it is also concurrently sending probe packets so it can receive new puzzle information. The major drawback of Congestion Puzzles is that an attacker can exploit the token bucket design by flooding the network with packets (without solving puzzles) in the hopes that this action will remove tokens that were supplied earlier by legitimate clients. The authors are aware of this problem and call it the "free-riding" problem. The authors attempt to fix this problem by introducing an IP-caching algorithm that allocates a separate token bucket for clients that are sending more data than others or for those with a common IP prefix. However, this adds much complexity to their scheme and significantly increases the memory storage overhead at the router.

The protocol presented by Feng et al., called *Network Puzzles* [9], requires a client and a congested router to constantly exchange puzzle information for each puzzle. Unlike Congestion Puzzles, each client can solve a puzzle only when it has been presented with the puzzle information from the congested router. The major drawback of this scheme is that it requires constant interaction between the client and the congested router (when puzzles are activated) which is impractical for most IP applications. A more seamless integration of the puzzle protocol into IP is needed.

3. Technical Challenges

Implementing a puzzle protocol at the IP layer is a difficult problem that requires careful consideration of several technical issues, including: seamless integration of the *puzzle-handshake* procedure into IP, granularity of puzzle generation (e.g., per-packet puzzles or per-flow puzzles), computation and storage overhead imposed on the puzzle server, communication overhead, and countering protocol circumvention.

A client puzzle protocol is a connection-oriented protocol that requires a three-way handshake between the puzzle solver (i.e., client) and the puzzle generator/verifier (i.e., "puzzle server" or router in an IP-layer puzzle scheme). A client initiates the protocol by sending the first service-request packet to a puzzle server; the puzzle server responds by sending back a puzzle challenge; the client responds by sending back the puzzle solution. Since a TCP connection is established using a somewhat similar three-way handshake, a puzzle protocol can be readily integrated with TCP. Unfortunately, the same is not true for IP. Unlike TCP, IP is an inherently connectionless protocol that transfers each packet from hop to hop until it reaches the destination. Obviously, requiring a client and router to perform a three-way handshake for every puzzle is not practical. Thus, an ideal approach to integrating the handshake procedure with IP is to enable each client to create its own puzzles (with some initial input from the puzzle generator) so that constant interaction with a puzzle generator is not needed. But at the same time, puzzles need to be unpredictable so that a client cannot pre-compute solutions ahead of time. Designing a protocol that has both features is a challenging problem.

To a large extent, the method employed for handling puzzle information (e.g., difficulty level, nonce, solution, etc.) determines the way the handshake procedure is integrated with IP. Congestion Puzzles proposed by Wang and Reiter [10] uses a *two-channel* approach: puzzle information and regular data are kept in separate packets. By using the two-channel approach, they were able to integrate a puzzle protocol within the IP layer without requiring the client and router to engage in repeated three-way handshakes. However, their technique has a critical drawback: it does not provide a secure way of associating the puzzle solutions with the clients that solved them. This drawback is the root cause of the free-riding problem discussed in Section 2. Network Puzzles proposed by Feng et al. [9] employs a *single-channel*

approach where the data and puzzle information are kept together in one packet. This approach does not suffer from the free-riding problem. Network Puzzles, however, does not integrate the handshake procedure with IP in a seamless way—it requires constant interaction between the client and the router.

The proposed scheme, Chained Puzzles, uses the single-channel approach to avoid the free-riding problem, yet it also manages to integrate the puzzle handshake procedure into IP seamlessly.

In an IP-layer client puzzle protocol, puzzle-capable routers perform puzzle generation and verification. In order to avoid negatively impacting the normal packet processing functions of the routers and to deter potential protocol exploits by attackers, it is important for a puzzle protocol to minimize the storage overhead and computational load on each router. To deter possible memory-exhaustion attacks, a puzzle protocol must not require a router to store large volumes of puzzle-related state information. An adversary may also attempt to exploit the puzzle protocol by sending large volumes of bogus solutions to a router so that it will exhaust clock cycles in verifying those solutions. To deter such an attack, the puzzle verification algorithm should be lightweight enough for a router to execute at or close to line rate.

4. Chained Puzzles

4.1. The Client Puzzle Algorithm

In the midst of a DDoS attack, the efficiency of the puzzle algorithm is very important. The cost of verifying a puzzle solution must be orders of magnitude cheaper than solving one so that minimal load is imposed on the routers. In other words, the cost of verifying puzzle solutions should be cheap enough to be done at or near line speed. For the puzzle algorithm, we propose a block-cipher algorithm that we call *XTEA6*. It is an extremely fast and lightweight algorithm that is a truncated version of the extended Tiny Encryption Algorithm (XTEA) [12]. XTEA is a Feistel routine that incorporates several rounds of S-boxes and basic bit-wise operations with a unique key in every round. The operations needed for the encryption and decryption routines include logical operations such as a bit-wise AND, a bit-wise XOR, and bit shifting; all of which can be easily performed on a router. Note that all existing puzzle algorithms are based on hash functions. We employ XTEA6 for two reasons: (1) it is considerably faster to execute than any known hash function and (2) it consumes less program space (very few lines of code) than any hash function. In some of our initial experiments with hash-based puzzles and XTEA6-based puzzles, we discovered that the execution times for MD5 and XTEA6 were 31 microseconds and 4 microseconds, respectively. Since puzzle verification is performed by executing the routine only once, an XTEA6 puzzle is much more efficient for the puzzle verifier.

XTEA6 includes three components: plaintext, ciphertext, and key. The plaintext is a 64-bit nonce generated by the router or the previous puzzle solution. The 128-bit key is composed of the 64-bit puzzle solution, hash value of the client IP address, and hash value of the destination IP address. Given a plaintext and the puzzle difficulty level d, a client must solve for the most significant 64 bits of the key that will produce a ciphertext in which the most significant d bits are zero (the bit pattern of the rest of the ciphertext is irrelevant). The difficulty level of the puzzle is defined by the value of d. For a given puzzle, a client must continuously execute the XTEA6 encryption algorithm using the same plaintext and different random selections of the key (changing only the 64 most significant bits) until XTEA6 produces an appropriate ciphertext. The verification process simply involves executing XTEA6 once to verify that a particular plaintext-key combination produces an appropriate ciphertext. The puzzle algorithm is illustrated in Figure 1.

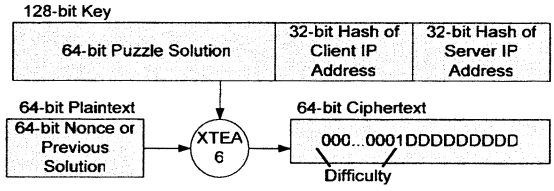

Figure 1. The client puzzle

As the difficulty level of the puzzle increases, the probability that an answer exists decreases. Using simulations, we were able to verify that a puzzle solution exists with overwhelming probability when the difficulty level is less than 20. According to our simulation results, difficulty levels below 20 are sufficient to throttle attack traffic in most instances.

4.2. An Overview of the Protocol

As stated earlier, DoS/DDoS mitigation is best performed as close to the source of the attack as possible to prevent a large amount of packets from converging onto the victim [2]. Therefore, puzzles should be generated and verified at the edge routers or border routers of an Internet Service Provider (ISP), rather than in the core of the Internet or at the routers close the victim. These routers are located closest to the source of the attack and are the optimal locations to filter attack traffic. In our scheme, we are required to modify routers at each end of the network: the border routers near the source of the attack (located upstream) and the border routers near the victim of the attack (located downstream). Thus, border routers will act as "puzzle servers". Throughout the remainder of this paper, these routers are referred to as *puzzle routers*. Border routers near the victim of the attack are simply designed to detect an ongoing attack, but can be used to generate and verify puzzles if the source of a different attack is within its own network. Methods to detect an attack are critical, but are beyond the scope of this paper. We assume that there exists a basic detection mechanism already in place that can determine if a flooding attack is underway. A puzzle router also exists near the source of the attack; this router is responsible for generating and verifying puzzles. Thus, there is a pair of puzzle routers at each end of the path that connects a client and a server. In order to handle puzzle generation and verification, the puzzle routers near the source is required to store state information. The

puzzle router maintains a *Plaintext Table* for each client's flow, where each entry contains a 20-bit hash of the concatenated string of the client and server IP addresses, and either the initial router nonce, N_R, or the client's previous puzzle solution (both 64 bits). In addition, the puzzle router also stores the level of difficulty for the puzzles in the 8-bit variable d. The clients all use N_R (and d) for the first puzzle, so initially the Plaintext Table is empty. After correct verification of the first puzzle in a given *Plaintext Chain*, a corresponding entry is created in the Plaintext Table. Thus, before the puzzle router stores state information, the client must first solve one puzzle. This will help avoid attacks on the storage capacity of the router. The Plaintext Chain is the sequence of puzzle solutions that the client creates for every puzzle with initial input from the puzzle router.

Because the number of clients that a typical puzzle router handles is not exorbitant, the amount of puzzle-related state information that needs to be stored in each puzzle router is reasonably small. For example, if each puzzle router serviced 10,000 clients with one flow each, then the size of the table would be 105 KB, which is small enough to be readily stored in the memory of any typical router. Existing IP-layer puzzle protocols require a puzzle-capable router to store a greater amount of state information. In [10], the authors use a Bloom filter stored within the router to check for duplicate puzzle solutions from a large number of clients and estimate that the size of the filter would be 1.1 MB. To introduce our protocol, we have summarized the steps taken by the client and puzzle router in the event of an attack in Table 1.

Table 1. Overview of Chained Puzzles

Puzzle Router:

1. A puzzle router downstream near the victim detects the attack and sends an *ICMP Congestion Notification* upstream to other puzzle routers, which enables Chained Puzzles.
2. When puzzles are enabled, an *ICMP Puzzle Challenge* is sent to each client with the initial router nonce, N_R, and the current difficulty level, d, in the IP options field. N_R is the same for every client and is refreshed periodically.
3. The puzzle router waits for the *Puzzle Transitional Period (PTP)* to expire or until it receives the first solution (IP Option 102) before it will verify the first puzzle solution. After puzzle verification, if it is correct, an entry is added in the Plaintext Table with the current solution. If the solution is incorrect, the packet is dropped. Subsequent packets use IP Option 101 to indicate that solutions are embedded and to indicate that these packets are part of an existing chain. When verifying solutions received from a given client, the puzzle router uses the current solution as the plaintext of the next puzzle and so on.
4. Upstream puzzle routers may receive congestion notifications from downstream puzzle routers to increase the difficulty level or to disable puzzles. Difficulty levels are updated by a new ICMP Puzzle Challenge to each client. A new N_R value can be distributed in the same fashion. Puzzles can also be disabled by sending a different ICMP message back to each client. When N_R or d is updated, a new chain is created and the router will wait until the *PTP* expires or until it sees a new solution for the new chain represented by IP Option 102.

Client:

1. If the client receives the Puzzle Challenge Packet from the puzzle router with the puzzle information for the first time, it solves the puzzle using the initial router nonce, N_R, and sends its next packet with the puzzle solution embedded into the IP options field of the header (IP Option 101).
2. For every new packet, the client uses the current puzzle solution as the plaintext for the next XTEA6 puzzle. The client solves the puzzle and embeds the solution into the IP header of the packet (IP Option 101). Doing so, the client will create a chain of puzzles.
3. A client may receive a new Puzzle Challenge Packet with a new N_R or d. It resets the chain and in the next packet supplies the next answer (IP Option 102). Every following packet has the IP Option 101 to indicate that a solution is embedded and to indicate that this packet is part of an existing chain.
4. A client may also receive an ICMP message to stop solving puzzles. Following this message, the next packet is sent without a solution.

Chained Puzzles allows the client to create a chain of puzzles which enables the seamless integration of client puzzles into IP. With the Plaintext Table, each client and puzzle router is synchronized with the same plaintext value used in the next puzzle, which is referred to as the Plaintext Chain. If the chain is broken, problems can arise. This issue is discussed in detail in Section 4.5.

During the PTP, packets are not verified by a puzzle router. Thus, this period must be kept to a minimum to avoid an attacker from having a window of opportunity to flood the rest of the network. The length of the PTP is essentially the minimum amount of time before the puzzle router will begin to see a puzzle solution or a new chain of puzzle solutions from any client. To calculate the PTP, we need to include the time it takes for the ICMP puzzle challenge packet to reach the client and the time it takes for one packet from the client to reach the puzzle router. Thus, this period should be slightly greater than the round-trip time (RTT) between the client and the puzzle router.

If a client does not receive the ICMP Puzzle Challenge, possibly because it was offline, it is possible for a client to query the puzzle router to receive the current N_R. This will allow more flexibility for various types of clients.

4.3. The Effectiveness of Chained Puzzles

A client puzzle at the IP layer is purposely designed to slow down any potential attackers in the event of an attack. In order to determine how effective Chained Puzzles would be, we need to first examine the behavior of a typical attacker. We assume that the attacker, or the zombie, will on average make a larger number

of requests than any single legitimate client. In a client puzzle protocol, the attacker has two realistic choices: to solve the puzzle or to supply a random solution and hope that it's correct. An attacker may alternate this behavior to create a more effective attack. Thus, we can define a *hybrid zombie*, which both solves and guesses the puzzles. Suppose that Q represents the probability that it solves a given puzzle. We can also define a unique attack, called a *Guessing Attack*, where the attacker deploys hybrid zombies scattered throughout the network that guess puzzle solutions instead of solving them.

When a zombie solves a puzzle per packet, its sending rate is reduced by a factor $S(k, d)$ defined in (1). In this equation, k is the average time it takes for the puzzle solver to execute the XTEA6 encryption function once and d is the difficulty level of the current puzzle. The function $S(k, d)$ is defined as the expected puzzle solve time.

$$S(k,d) = k \cdot 2^d \tag{1}$$

If we let t_p be the average processing time required to generate a regular packet, we can define the worst-case sending rate of a puzzle solver, r, using (2). In this equation, we assume that the time it takes to solve a puzzle will be much larger than the time it takes to generate a packet.

$$r = \frac{1}{t_p + S(k,d)} \cong \frac{1}{S(k,d)} \tag{2}$$

Thus, every puzzle solving client will have the sending rate as defined in (2). Thus, an attacker's sending rate is controlled by the difficulty level. Unfortunately, this includes legitimate clients as well as the puzzle-solving zombies. For a legitimate client that is sending less than the rate defined in (2), the only difference it will notice is an increase in CPU usage because of the puzzle mechanism. If the zombie does not solve a puzzle and supplies a random solution instead, its sending rate is not reduced. The probability of the packet reaching downstream depends solely on the difficulty of the puzzle. The probability that a guessing attacker's packet reaches the victim is given by (3).

$$p_D = 2^{-d} \tag{3}$$

Thus, when puzzles are enabled, the difficulty level needs to be initialized to a value high enough that minimizes this probability to combat the hybrid zombies.

4.4. Simulation Results

To determine how effective Chained Puzzles would be in the event of a DDoS attack, we simulated a DDoS attack using the Network Simulator (NS-2) [13]. We modified the simulator to have a packet wait in a queue for a determined amount of time ($S(k, d)$ seconds) before the sender sent it to the victim. Likewise, the first router in the path of the packet delayed forwarding the packet due to the puzzle verification ($S(k, 0)$ seconds). In our simulations, we created a tree network with 100 clients and 40 attackers. The attackers sent UDP packets at a rate 10 times higher than the legitimate clients. Each client began sending data at a randomly chosen time, while the attackers coordinated their transmissions (i.e., started sending packets at the same time). During the attack, we assumed that every legitimate client solved puzzles.

In Figure 2, we show the Normal Packet Survival Ratio (NPSR) [2]. NPSR is defined as the number of legitimate packets reaching the victim divided by the total number of packets sent by the legitimate clients during a DDoS attack. When IP (with no puzzles) was used, the NPSR value was equal to 0.23, which means that 77% of client packets were being dropped. In our simulations with Chained Puzzles (CP), we varied the value of Q, the probability that an attacker solves a given puzzle. We can observe that as the difficulty level increases, the NPSR value improves.

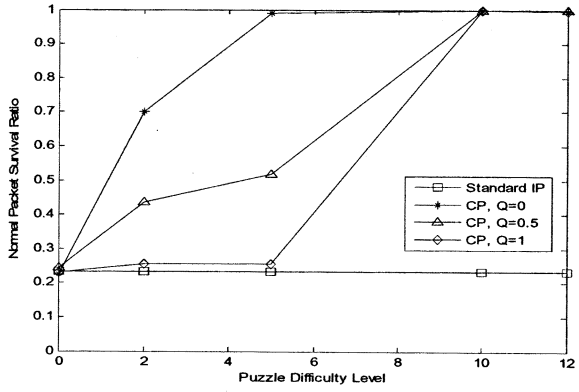

Figure 2. NPSR during a DDoS attack

From Figure 2, it appears that a difficulty level of ten is optimal to defend against the attack that was simulated. At this level, there were very few packets being dropped and the client's throughput remained fairly high. As expected, at difficulty levels higher than ten, clients' throughput dropped significantly. In fact, the sending rate of each client dropped from roughly 31 packets per second (with standard IP) to 10 packets per second (with puzzles on and a difficulty level of twelve). Therefore, there is a tradeoff between the NPSR value and the client's sending rate.

In this experiment, clients used the expected puzzle solve time when solving a puzzle. If the solve time was less than that, then it is likely that the NPSR would not reach one as quickly because the sending rates of all clients and attackers would increase. However, the NPSR would still reach one as the difficulty level increased. During an attack, this difficulty level can be adjusted if needed by the downstream puzzle routers by adjusting the number of ICMP congestion notification packets that they send to upstream puzzle routers.

4.5. Security of Chained Puzzles

A DoS attack mitigation scheme is obsolete if the scheme itself is vulnerable to attacks or if it can be circumvented by an attacker altogether. Here, we discuss possible attacks against Chained Puzzles and their countermeasures.

In our scheme, we rely on the puzzle routers downstream to be able to detect congestion or packet loss and send the congestion notifications upstream to the other puzzle routers. For our scheme to work properly, these messages need to be authenticated. An attacker could spoof a router's identity and send ICMP congestion notification messages to several puzzle routers, causing clients to solve puzzles when it is not necessary. Messages from puzzle routers to their clients also need to be authenticated to prevent an attacker from spoofing these messages. One way to do this is to use the technique described in [14]: when sending packets from a puzzle router, set the TTL value to 255. Since the packet is traveling from a first-hop router to a client, an attacker outside of the network could not spoof this message.

When puzzles are enabled, each client and puzzle router are synchronized with a plaintext value. Originally this value is N_R, but after the first puzzle is solved correctly, this value is the previous solution provided by that client. The router updates this value (in the Plaintext Table) after forwarding each packet, and the client updates this value after sending each packet. If the Plaintext Chain is broken, the legitimate client could be denied access for its subsequent packets, which causes another unique DoS attack. If the difficulty level is low, an attacker could guess a puzzle solution and spoof its IP address to a real client's address within the same subnet and use that client's destination IP address in its attack packets. This would cause an incorrect plaintext value to be inserted in the puzzle router's Plaintext Table. We call such an attack a *Spoofed Guessing Attack*. This attack is similar to the *Guessing Attack* mentioned in Section 4.3 except that it employs intelligent IP spoofing (i.e. it knows the correct range of IP addresses to spoof and the correct destination IP address). One way to combat this attack is to ensure that the difficulty level is high enough to prevent an attacker from guessing correct puzzle solutions of chains being transmitted by legitimate clients. An attacker can guess a puzzle solution with probability of 2^{-d}. For example, if the difficulty level was set to five, the probability of an attacker guessing a correct solution is 1/32.

When a Plaintext Chain has been broken, it will result in a sequence of incorrect puzzles for that client. One straightforward solution to this problem is to resynchronize that client with a new N_R by executing a handshake procedure between the client and a puzzle router. An attacker may exploit this approach by spoofing the IP address of the client and then send incorrect random solutions at the puzzle router, causing continuous resynchronizations between the client and the puzzle router. An attack such as this would be just as effective as a flooding attack because it would severely disrupt the flow of packets exchanged between the client and the server. A resynchronization technique that is robust against such attacks needs to be employed.

The attacks described above are difficult to defend against. Finding effective solutions for preventing protocol circumvention is a challenging task that requires careful consideration of the underlying architecture of the protocol itself.

5. Conclusions

In this paper, we have presented the framework of a novel client puzzle protocol that seamlessly integrates the puzzle handshake procedure with the IP layer. The integration was carried out by chaining puzzle solutions. To prevent the exploitation of the puzzle mechanism itself by attackers, we have proposed a lightweight puzzle algorithm based on a block cipher that enables a puzzle router to verify puzzle solutions at or near line speed. Chained Puzzles enables the network to rate limit attack traffic at the entry point of the network (i.e., at the border routers), which is the optimal place to mitigate a DDoS attack.

Scalability is an important issue with virtually all client puzzle protocols. As part of our future work, we intend to investigate the scalability of Chained Puzzles and evaluate its effectiveness at mitigating very large-scale DDoS flooding attacks.

6. References

[1] P. Ferguson and D. Senie. RFC 2267 – Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. January 1998.

[2] R. K. C. Chang. "Defending against flooding-based distributed denial-of-service attacks: a tutorial," in *IEEE Communications Magazine*. Volume 40, Issue 10. October 2002. Pages 42-51.

[3] A. Juels and J. Brainard. "Client Puzzles: A cryptographic defense against connection depletion attacks," in *Proceedings of NDSS '99 (Networks and Distributed Systems Security)*, pages 151-165. 1999.

[4] T. Aura, P. Nikander, and J. Leiwo. "DoS-Resistant Authentication with Client Puzzles," in *Lecture Notes in Computer Science*, Volume 2133, 2001.

[5] X. Wang and M. K. Reiter. "Defending Against Denial-of-Service Attacks with Puzzle Auctions (Extended Abstract)," in *Proceedings of the 2003 IEEE Symposium on Security and Privacy*. 2003.

[6] B. Bencsath, I. Vajda, and L. Buttyan. "A Game Based Analysis of the Client Puzzle Approach to Defend Against DoS Attacks," in *Proceedings of SoftCOM 2003. International Conference on Software, Telecommunications and Computer Networks.*

[7] D. Dean and A. Stubblefield. "Using Client Puzzles to Protect TLS," in *Proceedings of the 10th USENIX Security Symposium*. August 2001.

[8] B. Waters, J. A. Halderman, A. Juels, and E. W. Felten. "New Client Puzzle Outsourcing Techniques for DoS Resistance," in *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS'04)*. October 25-29, 2004.

[9] W. Feng, E. Kaiser, W. Feng, A. Luu, "The Design and Implementation of Network Puzzles," in *Proceedings of INFOCOM 2005*, March 2005.

[10] X. Wang and M. K. Reiter. "Mitigating Bandwidth-Exhaustion Attacks using Congestion Puzzles (Extended Abstract)," in *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS '04)*. October 25-29, 2004.

[11] W. Feng. "The Case for TCP/IP Puzzles," in *Proceedings of the ACM SIGCOMM 2003 Workshops*. August 2003.

[12] D. Wheeler and R. Needham. "TEA Extensions," Unpublished Manuscript. Available at: http://www.cl.cam.ac.uk/ftp/users/djw3/xtea.ps

[13] The network simulator – ns-2. Available at: http://www.isi.edu/nsnam/ns/

[14] J. Ioannidis and S. M. Bellovin. "Implementing Pushback: Router-Based Defense Against DDoS Attacks," in Proceedings of *NDSS'02 (Networks and Distributed Systems Security)*, February 2002.