Secure Client Puzzles based on Random Beacons

Yves Igor Jerschow and Martin Mauve
{jerschow, mauve}@cs.uni-duesseldorf.de

Institute of Computer Science, Heinrich Heine University, 40225 Düsseldorf, Germany

Abstract. Denial of Service (DoS) attacks pose a fast-growing threat to network services in the Internet, but also corporate Intranets and public local area networks like Wi-Fi hotspots may be affected. Especially protocols that perform authentication and key exchange relying on expensive public key cryptography are likely to be preferred targets. A well-known countermeasure against resource depletion attacks are client puzzles. Most existing client puzzle schemes are interactive. Upon receiving a request the server constructs a puzzle and asks the client to solve this challenge before processing its request. But the packet with the puzzle parameters sent from server to client lacks authentication. The attacker might mount a counterattack on the clients by injecting faked packets with bogus puzzle parameters bearing the server's sender address. A client receiving a plethora of bogus challenges may become overloaded and probably will not be able to solve the genuine challenge issued by the authentic server. Thus, its request remains unanswered. In this paper we introduce a secure client puzzle architecture that overcomes the described authentication issue. In our scheme client puzzles are employed non-interactively and constructed by the client from a periodically changing, secure random beacon. A special beacon server broadcasts beacon messages which can be easily verified by matching their hash values against a list of beacon fingerprints that has been obtained in advance. We develop sophisticated techniques to provide a robust beacon service. This involves synchronization aspects and especially the secure deployment of beacon fingerprints.

Keywords: network security, Denial of Service (DoS), client puzzles, authentication, public key cryptography

1 Introduction

Denial of Service (DoS) attacks aiming to exhaust the resources of a server by overwhelming it with bogus requests have become a serious threat to network services not only in the Internet. Corporate Intranets and public local area networks like Wi-Fi hotspots also pose promising targets for an effective DoS attack. Especially protocols and services that involve complex database queries or perform authentication and key exchange relying on expensive public key cryptography are likely vulnerable to DoS. By flooding valid-looking requests, for example SSL / TLS or IPsec authentication handshakes, an attacker seeks to overload his victim. A well-known countermeasure against resource exhaustion are *client puzzles* [2,3,11]. A server being under attack processes requests only from those clients that themselves spend resources in solving a

cryptographic puzzle and submit the right solution. Puzzle verification must be cheap, while the puzzle difficulty can be tuned from easy to hard. By imposing a computational task on the client the victimized server dramatically cuts down the number of valid requests that the attacker can emit. However, benign hosts having only one or a few requests are hardly penalized.

The majority of existing client puzzle protocols is interactive. The server constructs the puzzle upon receiving a request and demands from the client to solve this challenge before providing service. But the packet with the puzzle parameters sent from server to client lacks authentication. An attacker can mount a second DoS attack against prospective clients by flooding faked packets that pretend to come from the defending server and contain bogus puzzle parameters. The feasibility of such a counterattack depends on the network environment and the attacker's location. Forging the sender address is especially easy in wired and wireless LANs while it is more difficult in the Internet. The capability to eavesdrop on the traffic, which is a simple matter in WiFi networks, facilitates the attack but is not a necessary condition. An attacker who cannot overhear the client's request may continuously inject faked puzzle challenges. This proactive counterattack takes effect when the client actually issues a request. A client receiving a plethora of bogus challenges that were possibly chosen to be even more difficult than the puzzle of the genuine server may easily become overwhelmed. Most likely, it will not be able to solve the authentic challenge and thus its request will not be processed by the server. Depending on the chosen puzzle strength, even a modest puzzle packet rate may be sufficient for the attacker to succeed. Authenticating the challenge packet by means of a digital signature is not an option, since its generation and even verification are too expensive to be performed for all incoming requests[1]. For that very reason network protocols that employ public key cryptography may be vulnerable to DoS and should be protected by means of client puzzles.

In this paper we tackle the problem of authentication for client puzzles by introducing a secure architecture where clients construct and solve *non-interactive* puzzles from a *random beacon*. The main idea is to employ client puzzles non-interactively, which eliminates authentication issues with the server's challenge message, and to prevent precomputation of puzzle solutions by deriving puzzles from a periodically changing, secure random beacon. The beacons are generated in advance for a longer time span and broadcasted in the LAN by a special beacon server. All hosts obtain a signed fingerprint package consisting of cryptographic digests of these beacons. Verifying a beacon is very easy—it takes only a single hash operation, which can be performed at line speed by all hosts. Thus, DoS attacks on the beacon service are virtually impossible. If a server becomes overloaded due to a DoS attack, it asks all clients to solve and submit a puzzle prior to processing their requests. A client constructs a non-interactive puzzle by taking its request and the current beacon as input for a cost function. This can be, e. g., the reversal of a one-way hash function by brute force or the computation of a modular

[1] Example calculation: A current desktop machine can verify roughly 1000 – 3000 DSA-1024 signatures or 10 000 – 35 000 RSA-1024 signatures per second while on a 1 GBit link an attacker can flood up to 83 333 full MTU (1500 bytes) packets containing bogus signatures. If we assume smaller packet sizes the load induced by the attacker would be even higher.

square root. Having solved the puzzle, the client attaches the puzzle parameters and the solution to the pending request and retransmits it.

Besides the general approach of employing non-interactive puzzles our contribution lies in the development of sophisticated techniques to provide a robust and secure beacon service. We address synchronization aspects and especially elaborate the deployment of beacon fingerprints. Even if hosts were not able to obtain the signed fingerprint package using one of the regular distribution channels, they can acquire it on the fly from the beacon server and verify its signature despite of possible DoS flooding attacks. Our client puzzle architecture is primarily designed for LANs. But we show how to adopt the beacon service to operate with a single beacon server in Intranets or even in the Internet.

The remainder of this paper is organized as follows. In the next section, we discuss existing DoS countermeasures and focus in particular on various client puzzle schemes. Section 3 presents our secure client puzzle architecture, describes the construction of puzzles from a periodically changing random beacon and details how to deploy and verify these beacons. In Section 4 we extend our scheme by providing techniques to deliver beacons across LAN boundaries and by introducing a service which enables emergency deployment of signed beacon fingerprints. Finally, we conclude the paper with a summary in Section 5.

2 Related Work

A comprehensive survey on DoS attacks and proposed defense mechanisms can be found in [14]. The authors classify four categories of defense: (1) attack prevention, (2) attack detection, (3) attack source identification, and (4) attack reaction. In [11] Juels and Brainard introduced *client puzzles* to protect servers from TCP SYN flooding attacks. This countermeasure falls into the last category and constitutes a currency-based approach where clients have to pay before getting served. Being under attack, a server distributes to its clients cryptographic puzzles in a stateless manner asking them to reverse a one-way hash function by brute force. The difficulty of the puzzle is chosen depending on the attack strength. Only after receiving a correct solution from the client the server allocates resources for the dangling TCP connection. The idea of CPU-bound client puzzles has been applied to authentication protocols in general by Aura et al. in [2]. An implementation of client puzzles to protect the TLS handshake against DoS is described in [5]. Hash-reversal puzzles can be used both interactively and non-interactively [3].

Wang and Reiter proposed a multi-layer framework for puzzle-based DoS protection [20], which embeds puzzle techniques into both IP-layer and end-to-end services. The authors have presented two mechanisms: *Congestion puzzles* address bandwidth-exhaustion attacks in routers by cooperatively imposing puzzles to clients whose traffic is traversing a congested link. A traffic flow must be accompanied by a corresponding computation flow of puzzle solutions. The second mechanism called *puzzle auctions* protects an end-to-end service like TCP against protocol-specific DoS attacks. Clients bid for server resources by tuning the difficulty of the hash-reversal puzzle that they solve and the server allocates its limited resources to the highest bidder first.

Waters et al. suggested a client puzzle scheme based on the Diffie-Hellman key exchange where puzzle construction and distribution are outsourced to a secure entity called *bastion* [21]. The bastion periodically issues puzzles for a specific number of virtual channels that are valid during the next time slot. Puzzle construction is quite expensive since it requires a modular exponentiation, but many servers can rely on puzzles distributed by the same bastion. A client solves a puzzle by computing the discrete logarithm through brute force testing. On the server side, verifying a puzzle involves a table lookup and another costly modular exponentiation, which, however, is performed in advance during the previous time slot. As with Juels' client puzzles, the secure distribution of puzzle challenges to the clients remains an open issue also in Waters' scheme. The authors touch on the possibility of deriving puzzles from the emissions of a random beacon and state hashes of financial-market data or Internet news as candidates for a mutual source of randomness. But authentication of the input data is again an unsolved problem, especially in environments that do not enforce address authenticity. By injecting packets with faked data or beacons an attacker might render the DoS protection useless. With Waters we share the general idea of constructing puzzles from a random beacon and develop an architecture for a secure, real-world random beacon service. Our main contribution is a solid solution to the authentication problem that we tackle from scratch and thus rule out counterattacks on the puzzle distribution.

In [13] Martinovic et al. addressed DoS attacks in IEEE 802.11 networks aiming to exhaust the access point's (AP) resources by flooding it with faked authentication requests. The authors introduced *wireless client puzzles* that are distributed by a defending AP to joining stations. To support highly heterogeneous stations these puzzles are not CPU-bound. Instead of inverting a one-way function, a station has to measure the signal strength of the links to its neighbors and to find out those neighbors, whose link reaches a certain *Neighborhood Signal Threshold (NST)*. The NST is randomly chosen and frequently changed by the AP. A station replying with a wrong solution is detected by its neighbors, which thereupon issue a warning to the AP. However, similarly to the attack on classic client puzzles, an adversary might impersonate the AP and announce many different NST values thus sabotaging the verification.

Feng et al. implemented *network puzzles* at the "weakest link"—the IP layer—to make them universally usable [6]. By introducing *hint-based* hash-reversal puzzles the authors achieved linear granularity for interactive hash-reversal puzzles. However, their protocol is based on the assumption that the attacker cannot read or modify any packets sent between the client and the server. In contrast, we assume that the attacker is able to eavesdrop on the traffic.

In [4] Chen et al. gave a formal model for the security of client puzzles. Further client puzzle architectures are, e. g., [7, 15–17]. Puzzle-based DoS defense mechanisms can also rely on other payment schemes than CPU cycles, for example on memory [1], bandwidth [10, 19], or human interaction [18].

3 Secure Client Puzzle Architecture

Our attack model assumes an adversary (or a group of adversaries) that can inject arbitrary packets and, in particular, spoof the sender's IP and MAC address. The attacker

may also be capable to eavesdrop on some or even all packets sent by the legitimate hosts. However, he has only a very limited capability to modify packets or to destroy them by causing packet losses in switches or in the medium. Otherwise the attacker could render communication impossible simply by corrupting the data or through destruction of whole packets. Against such a threat puzzles would be of no avail.

3.1 Non-Interactive Client Puzzles

We suggest employing client puzzles in a *non-interactive* way where the client constructs the puzzle, solves it and attaches the solution to its request. To avoid the waste of time and CPU resources during normal operation when the server is not suffering from a DoS attack the client first sends its request without a puzzle solution. If the server replies in the regular manner everything is fine. In case of a DoS attack the server responds with a DoS alert message and drops the client's request without processing it further. The DoS alert message is an indication to the client that it must solve a puzzle prior to being served. Of course this message might be also a fake and currently there is no overload condition at the server. However, an unnecessarily solved puzzle is harmless and the client can cope with wrong alerts by introducing a timeout. A DoS alert message is considered authentic if no regular response has been received from the server during a certain time period. Now the client constructs a puzzle, solves it and retransmits its request along with the puzzle parameters and solution in a single message. The first time the client chooses for its puzzle the default level of difficulty, which has to be specified for the protocol or service that is safeguarded from DoS by client puzzles. A required solution time of $50-200$ milliseconds on a single CPU core of an off-the-shelf desktop machine may be a reasonable value. If the server does not respond the DoS attack may be stronger than expected. The client should retry after a timeout by doubling the initial puzzle difficulty, solving a more complex puzzle and retransmitting its request in combination with the new proof of work. Several connection attempts with an exponentially growing puzzle difficulty should be carried out prior to giving up.

During an overload condition the server must parse all incoming requests, answer with a DoS alert message and verify all submitted puzzle solutions. Its computing power must be chosen high enough to perform this puzzle preprocessing at full bandwidth and to serve requests at an ordinary rate without becoming overburdened. Only requests from clients that have solved a puzzle and submitted a correct solution have a chance of being processed. A priority queue can be used to manage requests carrying puzzles with different levels of difficulty. The request from the client that has solved the most difficult puzzle is served fist. To limit the queue size a periodic cleanup should purge requests that have stayed in the queue longer than a predefined time interval.

3.2 Client Puzzles from a Random Beacon

We should prevent the reuse of a single puzzle solution by multiple different requests without demanding from the server to log spent puzzles solutions. This can be achieved

by binding the puzzle to the request so that a different request requires solving a completely new puzzle. In our client puzzle architecture a cryptographic digest of the request must flow into the puzzle construction. Nevertheless the protocol or service running on the server must provide some mechanism to recognize identical requests originating from the same client so that resources (e. g., database lookup or signature verification / computation) required to complete such requests are committed only once.

A serious issue with non-interactive client puzzles may pose precomputation attacks where the attacker prepares a huge pile of requests and corresponding puzzle solutions in advance. He might engage dozens of machines, e. g., from a botnet, to solve thousands of puzzles which enables him to overwhelm a server by flooding his prepared requests at some point in the future. We address this threat by constructing client puzzles from a periodically changing random beacon. The beacon is broadcasted in the whole network at regular intervals so that both client and server have access to a mutual source of randomness. This renders precomputation attacks virtually impossible since the beacon is unpredictable and puzzles derived from it are valid only for a short period of time.

Combining these two ideas we create our client puzzles from the cryptographic digest of the request r and the current random beacon b. Let H be a cryptographic hash function (e. g., SHA-1 or RIPEMD-160), then the input for the puzzle construction is the d-bit digest

$$s = H(r \,||\, b) \tag{1}$$

where $||$ denotes the concatenation of two bit strings.

3.3 Puzzle Construction

Our client puzzle architecture does not depend on a specific cost function. The only requirement is that the puzzle can be derived *by the client* from an arbitrary number, which is the digest s in our scenario. In case of the well-known hash-reversal cost function [2, 3, 11] the puzzle is to find by brute force a bit string x so that

$$H(s \,||\, x) = \underbrace{000\ldots000}_{\substack{\text{first } q \text{ bits} \\ \text{are zero}}} \underbrace{Z.}_{\substack{\text{remaining} \\ d-q \text{ bits}}} \tag{2}$$

To simplify the implementation x should be a fixed-length integer (e. g., 64 bits), which is initialized with zero and incremented by one for each new try. The number of leading zero bits q in the output of H determines the puzzle difficulty. Increasing q by one doubles each time the expected number of tries to find a suitable x. Thus, the granularity of the hash-reversal puzzle is exponential.

In [9] we have introduced a novel non-interactive client puzzle scheme that is based on the computation of square roots modulo a prime. Solving a modular square root puzzle involves several modular exponentiations whereas verification requires performing only a single modular squaring operation. While a hash-reversal puzzle can be solved in parallel by multiple machines or CPU cores and has only exponential granularity, a modular square root puzzle is non-parallelizable to a high degree and provides polynomial granularity. Moreover, the solution time of a hash-reversal puzzle is highly non-deterministic, while a modular square root puzzle has only a negligible probabilistic

component which can be even eliminated by taking different primes and slightly relaxing the puzzle complexity. A minor drawback of modular square root puzzles is that the level of difficulty cannot be chosen arbitrarily high without rendering verification too expensive. The size of the solution also grows with increasing puzzle difficulty. But for solution times which are usually chosen in the order of milliseconds modular square root puzzles can be verified at line speed and thus are fully viable for DoS prevention in practice. For our secure client puzzle architecture they might thus be even better candidates than hash-reversal puzzles.

3.4 Random Beacon Server

The random beacon server B is ideally a dedicated machine in the LAN that periodically broadcasts a beacon packet containing a n-bit random number b. Depending on the layer at which client puzzles are employed, the beacon message is encapsulated in a raw Ethernet frame, an IP datagram or in a UDP segment. To render any network-based attacks on the beacon server impossible, we suggest to disable the receiver unit of B's network interface or simply to drop all incoming packets without inspecting them. Only outgoing packets to provide the beacon service should be permitted. An isolated beacon server that does not receive any input is DoS-resistant by design. The requirement of setting up a dedicated machine may be of course relaxed at the expense of security. Basically, any existing server in the LAN can run the beacon service. Since the computational burden is minimal, even an off-the-shelf desktop machine would suffice for this task. Thus, setting up a beacon server does not constitute a demanding infrastructure requirement.

The random numbers to be included in the beacons are generated in advance for a time span of several days, weeks, or even months. In practice, this task can be accomplished by a cryptographically secure pseudorandom number generator that runs on the beacon server. For the generation of a set of random numbers three parameters have to be provided: the bit length n of each number, the time span t covered by the set, and the beacon period p, i. e., the time between the emission of the current and the next random number. In practice, t and p will be measured in seconds. The set consists of $k = \frac{t}{p}$ random numbers requiring $k \cdot n$ bits of output from the random number generator.

Next, for each random number b_i, $1 \leq i \leq k$, we compute a d-bit digest $H(b_i)$ by applying the cryptographic hash function H. These are the *fingerprints* of the random beacons. Now a *fingerprint package* $<T_{Start}, t, p, H(b_1), ..., H(b_k)>$ is created and digitally signed using the private key of the beacon server B. T_{Start} is a timestamp that denotes the time when the emission of the associated beacons starts. We expect that the beacon server has obtained a public key certificate from a well-known Certificate Authority (CA) and that everyone can verify its signature on the fingerprint package if B's certificate is attached. The final step is the deployment of the signed fingerprint package to all hosts in the network that will either solve or verify client puzzles in case of a DoS attack. The preferable method is to publish the signed fingerprint package along with B's certificate on the institution's website, where it can be downloaded and verified by all users / hosts. A manual deployment by sending the fingerprints via e-mail or obtaining them on a USB flash drive from the network administrator may be also conceivable in some scenarios. Instead of contacting the network administrator one

could also imagine to install a physically secured terminal somewhere in the building where users can store the fingerprint package on their USB flash drive by themselves. The size of the fingerprint package depends on the covered time span t and the beacon period p, but is reasonably small even for long time spans and short intervals. For example, for $t = 30$ *days* and $p = 60$ *sec* we need $k = 43\,200$ fingerprints, which occupy about 844 KB if using SHA-1 with a digest length d of 160 bits.

At time T_{Start} the beacon server switches to the new beacon set by emitting the random number b_1 which is valid until $T_{Start} + p$. Every p seconds the current number b_i is replaced by releasing its successor b_{i+1}. Since broadcast transmissions are not reliable, a beacon packet may get lost. Therefore we propose to periodically retransmit the current beacon during its lifetime, e. g., to broadcast it once a second. This ensures that all hosts in the network, even those that have joined recently, will receive the current beacon without noticeable delay. An appropriate bit length n for random numbers to generate client puzzles that are unpredictable is in the order of a cryptographic hash, e. g., 160 – 256 bits. Hence, beacon packets are very small, no more than 60 – 70 bytes including all protocol headers (e. g., UDP, IP, and Ethernet).

3.5 Receiving and Verifying the Beacons

All clients and servers (in the following just called hosts) in the network obtain the fingerprint package in advance using one of the deployment techniques described in the previous subsection. We assume that the clocks of all hosts and the beacon server are loosely synchronized. The allowable time skew δ may be in order of minutes. This requirement can be easily achieved even without a time synchronization protocol like NTP, just by letting the users manually adjust their computer's clock occasionally. To synchronize with the beacon server a host begins at time $T_{Start} - \delta$ to verify all incoming beacon packets by computing the beacon's digest and matching it against $H(b_1)$ from the fingerprint package. Having received a beacon b with $H(b) = H(b_1)$ the host records the beginning of the new beacon period and sets b_1 a the current beacon. This synchronization will succeed at the latest at time $T_{Start} + \delta$. Subsequent beacons that the host receives are matched against $H(b_2)$, or to generalize, after having verified and set b_i the host matches new beacons against $H(b_{i+1})$ and switches to b_{i+1} if the comparison succeeds.

Hosts that join the network during a beacon period can also synchronize with the beacon server in a straightforward manner. A host joining at time $T_{Start} + h$ (according to its clock) matches incoming beacons against a list L of fingerprints, namely $L = \langle H(b_{v-r}), ..., H(b_{v+r+2}) \rangle$ with $v = \lceil \frac{h}{p} \rceil$ and $r = \lceil \frac{\delta}{p} \rceil$. In case of a match with one of the fingerprints from the list the beacon b is set as the tentative beacon and all fingerprints preceding it in the list are removed. The host continues to verify subsequent beacons for $2p$ seconds. This ensures that it definitely hits and observes a complete beacon period. If a subsequent beacon corresponds to a newer fingerprint from the list, then it becomes the tentative beacon and old fingerprints are once again purged from L. This is done to prevent replay attacks with outdated beacons. After $2p$ seconds the synchronization is completed. The tentative beacon becomes the current beacon—now it has definitely been identified.

An attacker may try to interfere with the beacon service by flooding thousands of faked beacon packets bearing the beacon server's sender address. However, computing the cryptographic hash of a packet and matching this digest against a stored value or a small set of values is a cheap task that in general can be performed at full link speed in Gigabit networks. Table 1 shows benchmark results of four cryptographic hash functions that we have measured on an Intel Core 2 Quad Q9400 2.66 GHz CPU using a 64-bit Linux distribution, *GCC 4.4* and the cryptographic library *Botan* [12]. A single CPU core achieves a throughput of 227 – 426 MB/sec while a Gigabit link has a transfer rate of 119 MB/sec. Thus, by flooding bogus beacons the attacker is only able to raise the CPU load on the hosts, but cannot prevent the identification of the authentic beacon.

Though beacon packets are periodically retransmitted during a beacon period, a host should not except that it will receive all consecutive beacons. Due to abnormal operation it might sometimes miss some beacons. To recover from this condition we introduce a lookahead of a few fingerprints. Having failed to replace the current beacon b_i by its successor for more than p seconds, the host matches incoming beacons against the next l fingerprints $H(b_{i+1}), ..., H(b_{i+l})$. If the verification still fails for several beacon periods, the host should increase l and, even if this is of no avail, it should adjust i according to the time that has passed since the last beacon update.

Table 1. Benchmark: Throughput of cryptographic hash functions on Intel Core 2 Quad Q9400 2.66 GHz (one core active).

hash function	*block size*	*digest length*	*speed*
MD5	512 bits	128 bits	**426.4 MB/s**
RIPEMD-160	512 bits	160 bits	**260.5 MB/s**
SHA-1	512 bits	512 bits	**327.0 MB/s**
SHA-384	1024 bits	384 bits	**227.4 MB/s**

3.6 Puzzle Submission and Verification

In case of a DoS attack on the server the client submits along with its request r the puzzle solution and the beacon b from which the puzzle has been derived. Instead of transmitting the beacon it can also indicate its index in the fingerprint package. While the client was solving the puzzle or while it stayed in the server's input queue the current beacon may already have changed. Therefore the server must accept also puzzle solutions that were derived from older beacons within reasonable bounds. Considering the proposed puzzle solution time of about 50 – 200 milliseconds and a beacon period in the order of some seconds we recommend to tolerate only puzzles constructed from the current or the previous beacon. This keeps the protocol simple and effectively prevents precomputation attacks. Requests bearing a puzzle from an outdated beacon are dropped without verification. In networks encountering large delays the beacon period should be chosen accordingly. In case of a valid beacon the server first computes the digest $s = H(r \mid\mid b)$ and then verifies the solution of the puzzle constructed from s.

4 Protocol Extensions

4.1 Beacon Distribution across LAN Boundaries

Our secure client puzzle architecture primarily focuses on LANs where counterattacks on interactive client puzzle protocols through injection of bogus challenges are especially easy and thus very promising. But depending on the attacker's power and resources a counterattack with faked puzzle challenges may succeed also in large-scale networks like corporate Intranets or even in the Internet. Especially hosts in the edge network might be vulnerable to puzzle counterattacks. Thus, it can make sense to employ non-interactive client puzzles that are derived from a random beacon also in these settings. However, broadcasting beacons works only within a LAN. A beacon server that shall supply hosts spread across LAN boundaries with beacons must resort to a different distribution technique. A well-known solution for this task is multicast. Hosts employing the secure client puzzle architecture could subscribe to the multicast group to which the beacon server addresses its periodic beacons. But a major issue with beacon dissemination through multicast is that many ISPs do not route multicast traffic which breaks traditional input-rate-based billing models. Thus, while multicast may be an option for corporate networks administered by a single entity, we must resort to a different approach to provide the beacon service over the Internet.

We propose to deploy beacons across LAN boundaries via unicast and pay particular attention to DoS resilience of the beacon server. Hosts receive the current beacon from the beacon server *on demand* after having issued a corresponding request. Unicast deployment of beacons on a subscription basis where a host issues a single request and hereon periodically receives beacons from the server until it cancels this subscription would be prone to a DoS attack. The attacker could take on many different identities and spawn a multitude of faked subscriptions that might quickly exhaust the bandwidth of the beacon server. Therefore each new beacon that a host receives must be triggered by a separate request. This is a kind of tit-for-tat strategy. The server supplies only those hosts with beacons that themselves spend bandwidth and continuously send corresponding requests. The size of the request packet (usually, a UDP datagram encapsulated in IP) must be at least as large as the beacon packet. To enforce its resistance to DoS the beacon server may even demand that valid beacon requests have to be padded with zeros to have full MTU size, which usually is 1500 bytes (20 – 25 times larger than the beacon packet). This will raise the costs on the attacker's side and make his attempts to exhaust the server's resources quite useless. On the other side, legitimate hosts requesting every few seconds a new beacon will perfectly cope with this small *bandwidth-based payment* for the beacon service. The processing time for a beacon request is minimal. The server performs virtually no computation—it only crafts and sends a reply packet containing the current beacon. Nevertheless, the server capacity, especially its processor and network link, has to be carefully chosen to withstand a fluctuating number of requests including potential attackers. In contrast to the Internet scenario, the broadcast service in a LAN can be provided by any off-the-shelf desktop machine.

Requesting a beacon from a beacon server in the Internet is in some respects comparable to a DNS lookup. Indeed, another approach to deploy beacons is to rely on DNS. The beacon server becomes the authoritative name server for a particular domain. Hosts

receive the current beacon by requesting a TXT resource record. In its reply the beacon server must set the TTL to a value smaller than the beacon period p. Choosing $\frac{p}{2}$ for the TTL seems to be appropriate to guarantee freshness and at the same time to distribute load. Owing to DNS caching the number of requests going end-to-end from host to beacon server will be significantly cut down which results in a smaller traffic footprint.

4.2 Emergency Deployment of Beacon Fingerprints

Obtaining the signed fingerprint package is a crucial step in the setup of our secure client puzzle architecture. In the previous section we have proposed several deployment techniques (download from a website, manual distribution via e-mail or USB flash drive, secure terminal) to achieve this goal. However, an attacker may try to sabotage the download of the fingerprints by mounting a DoS attack against the web server or through injection of spoofed packets, e. g, TCP resets, aiming to impede the connection. Secure transmission via SSL or IPSec does not protect from DoS attacks, since these protocols rely on expensive public key cryptography and themselves may require protection from DoS by means of client puzzles. Manual distribution of the fingerprint package can be too expensive in large networks while some institutions might not be able to afford the installation of a secure terminal. Therefore we introduce a further deployment method for the fingerprints as a fallback option for emergency situations, where the other distribution channels fail. It is designed to work within a LAN.

Resorting to the Beacon Server The beacon server can periodically broadcast the current fingerprint package by dividing it into several packets. If the current fingerprint package covers a very long time span resulting in a large number of packets, the beacon server builds a smaller one which contains only the beacons for the next few hours or days. Assume that it takes g packets to deliver the fingerprint package which must be also digitally signed. To enable an efficient verification of each of the g packets for the receiver the beacon server computes the cryptographic hash of each packet and signs a list consisting of these g digests plus the timestamp T_{Start}. The digest list along with the timestamp and the signature must fit into a single packet—the header of the fingerprint package. Thus, g is bounded by the MTU, the signature size and the digest length d. Assuming 1500 bytes for the first, 1024 bits for the second and 160 bits for the third factor we obtain $g \leq 68$. The beacon server periodically broadcasts the header packet followed by the g numbered fingerprint packets. A host requiring the fingerprint package first waits for the header packet, verifies its signature and timestamp and stores the g digests of the fingerprint packets. Now it is ready to receive and quickly validate the fingerprint packets by computing their digest and matching this digest against the list. The order of the received fingerprint packets is irrelevant since each packet has a sequence number and can be independently verified and stored. Having collected all g parts of the fingerprint package the host finally needs to synchronize with the beacon server to identify the current beacon.

Fending off Flooding Attacks with Faked Signatures The deployment of beacon fingerprints by the beacon server is very robust to DoS attacks since the beacon server

does not receive any requests and thus cannot be compromised or even influenced from outside. Spoofed fingerprint packets are also harmless—they can be easily detected by checking their digest. The only sticking point is the expensive verification of the signature in the header packet. But we introduce two measures to cope with a potential flooding attack of faked header packets.

The first measure is an *observe-then-verify strategy*. The genuine header packet is periodically retransmitted by the beacon server. Hence only those header packets that a host receives over and over again are potentially authentic and need to be taken into account for verification. Instead of trying to verify all incoming header packets a host first observes the header packets that it receives for some consecutive periods and records them (or their hash values to save memory). After this observation phase only those header packets are selected for verification that have been received repeatedly during multiple periods. Now this pile of header packets gets verified until the genuine signature is found. Checking the included timestamp safeguards against replay attacks. New header packets arriving during this phase are ignored. If all packets from the pile turn out to be faked, a host retries by initiating a new observation phase. The shorter we choose the retransmission period for the header packet, the smaller will be the pile of collected packets that need to be validated and the faster a host will identify the genuine header packet. A retransmission period of 50 msec may be reasonable for the header packet while fingerprint packets are retransmitted, e. g., only every 5 seconds. Assuming a 1 GBit link, full MTU packets (1500 bytes, this can be enforced by policy), an observation phase taking 1 second (20 periods) and a quota of 0.5 packets per period on average (i. e., at least 10 copies), there will be at most 8333 candidates that must be verified. In case of an RSA-1024 signature having a verification throughput of 10 000 – 35 000 operations per second on current desktop machines it will take less than a second to validate the whole pile of header packets. This sample calculation confirms that the observe-then-verify strategy provides a viable way to quickly filter out the genuine header packet and to obtain the fingerprint package. An alternative, more basic approach which does not require to count duplicates is to collect all header packets arriving during 2 – 4 periods (at most 8333 – 16 666 packets in our example) and then to verify all them. If not too many bursty packet losses occur, at least one genuine header packet will be among this capture with very high probability.

The second measure is optional and aims to significantly cut down the number of valid-looking header packets that the attacker can emit by including a hash-reversal puzzle. Since the beacon server is ideally a dedicated machine which fulfills no other tasks besides broadcasting beacons and fingerprint packages, it has plenty of idle CPU time. This time can be used to solve a hash-reversal puzzle (see Section 3.3) for the header packet that will be broadcasted when the next fingerprint package takes effect. The puzzle is derived from the digest of the header packet. The beacon server continues to solve the puzzle by finding new solutions x that yield a larger number q of leading zero bits in the output of H than the previous solution until it is time to deploy the corresponding fingerprint package. For example, if fingerprint packages are issued for 24 hours, the beacon server has 24 hours to solve the puzzle for the corresponding header packet. Due to the nondeterministic nature of the hash-reversal puzzle the puzzle difficulty determined by q will slightly vary from run to run. Hosts waiting for the

header packet can drop all packets that have no puzzle attached, carry a wrong solution, or whose puzzle difficulty falls below a predefined threshold. Header packets that have passed this filter are inserted into a priority queue. The packet with the puzzle that has the highest level of difficulty is verified first.

To verify a signature issued by the beacon server B a host requires B's certificate. If it has not cached this certificate in the past when obtaining the fingerprint package along with B's certificate through regular distribution channels, we must provide a way to acquire it on the fly. This can be accomplished in the same manner as the deployment of the signed header packet. The beacon server periodically broadcasts its certificate in a special certificate packet. To withstand a DoS flooding attack with forged certificate packets a host applies the observe-then-verify strategy, which enables to quickly identify and verify the genuine certificate. In addition, the authentic certificate packet may also be protected by a hash-reversal puzzle.

5 Conclusion

In this paper we have introduced a secure client puzzle architecture where puzzles are constructed by the client from a periodic random beacon. By employing client puzzles non-interactively we bypass authentication issues with the challenge message sent from server to client in interactive client puzzle schemes. To rule out precomputation attacks valid puzzles must be derived from the current beacon which is broadcasted by the beacon server. Hosts obtain in advance a signed fingerprint package with cryptographic digests of the beacons which enables them to instantly authenticate all incoming beacon packets. We have proposed several regular distribution channels for the fingerprint package and introduced an emergency deployment technique to acquire the beacon fingerprints on the fly from the beacon server. Our beacon service is by design robust against DoS counterattacks. It can operate not only in LANs but also across LAN boundaries by distributing beacons via multicast, unicast or through DNS. For future work, we envision an implementation of the secure client puzzle architecture to protect the public key handshake of our cryptographic link layer protocol [8].

References

[1] Martin Abadi, Mike Burrows, Mark Manasse, and Ted Wobber. Moderately Hard, Memory-bound Functions. *ACM Transactions on Internet Technology*, 5:299–327, May 2005.

[2] Tuomas Aura, Pekka Nikander, and Jussipekka Leiwo. DOS-Resistant Authentication with Client Puzzles. In *Revised Papers from the 8th International Workshop on Security Protocols*, pages 170–177, April 2001.

[3] Adam Back. Hashcash - A Denial of Service Counter-Measure, August 2002. `http://www.hashcash.org/papers/hashcash.pdf`.

[4] Liqun Chen, Paul Morrissey, Nigel P. Smart, and Bogdan Warinschi. Security Notions and Generic Constructions for Client Puzzles. In *ASIACRYPT '09: Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security*, pages 505–523, December 2009.

[5] Drew Dean and Adam Stubblefield. Using Client Puzzles to Protect TLS. In *SSYM'01: Proceedings of the 10th USENIX Security Symposium*, August 2001.

[6] Wu-chang Feng, Ed Kaiser, Wu-chi Feng, and Antoine Luu. The Design and Implementation of Network Puzzles. In *INFOCOM 2005: Proceedings of the 24th IEEE Conference on Computer Communications*, pages 2372–2382, March 2005.

[7] Helmut Hlavacs, Wilfried N. Gansterer, Hannes Schabauer, Joachim Zottl, Martin Petraschek, Thomas Hoeher, and Oliver Jung. Enhancing ZRTP by using Computational Puzzles. *Journal of Universal Computer Science*, 14(5):693–716, 2008.

[8] Yves Igor Jerschow, Christian Lochert, Björn Scheuermann, and Martin Mauve. CLL: A Cryptographic Link Layer for Local Area Networks. In *SCN 2008: Proceedings of the 6th Conference on Security and Cryptography for Networks*, pages 21–38, September 2008.

[9] Yves Igor Jerschow and Martin Mauve. Non-Parallelizable and Non-Interactive Client Puzzles from Modular Square Roots. In *ARES 2011: Proceedings of the 6th International Conference on Availability, Reliability and Security*, pages 135–142, August 2011.

[10] Yves Igor Jerschow, Björn Scheuermann, and Martin Mauve. Counter-Flooding: DoS Protection for Public Key Handshakes in LANs. In *ICNS 2009: Proceedings of the 5th International Conference on Networking and Services*, pages 376–382, April 2009.

[11] Ari Juels and John G. Brainard. Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks. In *NDSS '99: Proceedings of the Network and Distributed System Security Symposium*, February 1999.

[12] Jack Lloyd. Botan: a BSD-licensed crypto library for C++. `http://botan.randombit.net`.

[13] Ivan Martinovic, Frank A. Zdarsky, Matthias Wilhelm, Christian Wegmann, and Jens B. Schmitt. Wireless Client Puzzles in IEEE 802.11 Networks: Security by Wireless. In *WiSec 2008: Proceedings of the ACM Conference on Wireless Network Security*, March 2008.

[14] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. Survey of Network-Based Defense Mechanisms Countering the DoS and DDoS Problems. *ACM Computing Surveys*, 39(1):3, 2007.

[15] Patrick Schaller, Srdjan Čapkun, and David Basin. BAP: Broadcast Authentication Using Cryptographic Puzzles. In *ACNS '07: Proceedings of the 5th International Conference on Applied Cryptography and Network Security*, pages 401–419, June 2007.

[16] Qiang Tang and Arjan Jeckmans. On Non-Parallelizable Deterministic Client Puzzle Scheme with Batch Verification Modes. Centre for Telematics and Information Technology, University of Twente, January 2010. `http://doc.utwente.nl/69557/`.

[17] Suratose Tritilanunt, Colin Boyd, Ernest Foo, and Juan Manuel González Nieto. Toward Non-Parallelizable Client Puzzles. In *CANS 2007: Proceedings of the 6th International Conference on Cryptology & Network Security*, pages 247–264, December 2007.

[18] Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. CAPTCHA: Using Hard AI Problems For Security. In *EUROCRYPT '03: Proceedings of the 22nd International Conference on Theory and Applications of Cryptographic Techniques*, pages 294–311, May 2003.

[19] Michael Walfish, Mythili Vutukuru, Hari Balakrishnan, David Karger, and Scott Shenker. DDoS Defense by Offense. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 303–314, September 2006.

[20] XiaoFeng Wang and Michael K. Reiter. A multi-layer framework for puzzle-based denial-of-service defense. *International Journal of Information Security*, 7:243–263, July 2008.

[21] Brent Waters, Ari Juels, J. Alex Halderman, and Edward W. Felten. New Client Puzzle Outsourcing Techniques for DoS Resistance. In *CCS '04: Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 246–256, October 2004.