DOS-resistant Authentication with Client Puzzles

Tuomas Aura[1], Pekka Nikander[1], and Jussipekka Leiwo[2]

[1] Helsinki University of Technology
P.O.Box 5400, FIN-02015 HUT, Finland
{Tuomas.Aura,Pekka.Nikander}@hut.fi

[2] Vrije Universiteit, Division of Sciences
De Boelelaan 1081A, 1081 HV Amsterdam, The Netherlands
leiwo@cs.vu.nl

Abstract. Denial of service by server resource exhaustion has become a major security threat in open communications networks. Public-key authentication does not completely protect against the attacks because the authentication protocols often leave ways for an unauthenticated client to consume a server's memory space and computational resources by initiating a large number of protocol runs and inducing the server to perform expensive cryptographic computations. We show how stateless authentication protocols and the *client puzzles* of Juels and Brainard can be used to prevent such attacks.

1 Introduction

Denial-of-service (DOS) attacks that exhaust the server's resources are a growing concern on the Internet and other open communications systems. For example, in the SYN attack, a client floods the server with the opening messages of the TCP protocol and fills the space reserved in the server for storing half-open connections.

A solution to such threats is to authenticate the client before the server commits any resources to it. The authentication, however, creates new opportunities for DOS attacks because authentication protocols usually require the server to store session-specific state data, such as nonces, and to compute expensive public-key operations. One solution is to begin with a weak but inexpensive authentication, and to apply stronger and costlier methods only after the less expensive ones have succeeded. An example of a weak authentication is the SYN-cookie protection against the SYN attack where the return address is verified not to be fictional by sending the client a nonce that it must return in its next message. This strategy is not entirely unproblematic because the gradually strengthening authentication results in longer protocol runs with more messages and the security of the weak authentication mechanisms may be difficult to analyze.

In this paper, we advocate the design principle that *the client should always commit its resources to the authentication protocol first and the server should be able to verify the client commitment before allocating its own resources*. The rule of thumb is that, at any point before reliable authentication, the cost of the protocol run to the the client should be greater than to the server. The client's costs can be artificially increased by asking it to compute solutions to puzzles that are easy to generate and verify but whose difficulty for the solver can be adjusted to any level. The server should remain stateless and refuse to perform expensive cryptographic operations until it has verified the client's solution to a puzzle.

2 Related work

Classical models of denial of service by Gligor and Yu [6,17], Amoroso [1], and Millen [13] concentrate the specification and design of fair multi-user operating systems. They assume that all service requests are arbitrated by a trusted computing base (TCB) that enforces the policy set by a single security officer. Their ideas do not extend well to open distributed systems like the Internet where there is no central trusted administration and no global policy or means for enforcing one, and there are too many simultaneous users to theoretically guarantee the availability of any service.

Graph-theoretical models of network reliability by Cunningham [4] and Phillips [14] assess the vulnerability of a communications network to the destruction of nodes and links. These models are useful in the design of network topologies on the physical layer but their applicability does not easily extend to higher protocol layers.

The SYN attack against the TCP connection protocol on the Internet was reported e.g. in [3]. The attack and possible remedies were analyzed in detail by Schuba et al. [15]. Cookies have been previously used in the Photuris protocol by Karn and Simpson [11] and in the Internet Key Exchange (IKE) by Harkins and Carrel [7]. Criticism of the latter [16] shows that the gradually strengthening authentication is not straightforward to design and a careful analysis of the server resource usage is needed.

Meadows [12] formalized the idea of gradually strengthening authentication. The design goals of a cryptographic protocol should specify how much resources the server may allocate at each level when its assurance of the client's identity and honest purposes step by step increases. This assurance is measured by the resources the client would need to mount a successful attack.

The advantages of statelessness in the beginning of an authentication protocol were recognized by Janson & al. [9] in the KryptoKnight protocol suite. Aura and Nikander [2] generalized the cookie approach to create stateless servers that maintain connections by passing the state data to the client. The paper also gives examples of authentication protocols where the server avoids saving a state until the authentication of the client is complete. Hirose and Matsuura [8] applied these ideas to a DOS-resistant version of their KAP protocol. In addition to remaining stateless, the server in their protocol postpones expensive exponentiation operations until it has verified that the client has performed similar operations. This way, the server commits its memory and computational resources only after the client has demonstrated its sincerity.

The idea of requiring the client to commit its resources first was described early by Dwork and Naor [5]. They suggested increasing the cost of electronic junk mailing by asking the sender to solve a small cryptographic puzzle for each message. The cost would be negligible for normal users but high for mass mailers. Juels and Brainard [10] recently presented a simpler puzzle that could be sent to TCP clients during a suspected SYN attack. If the server thinks it is under a denial-of-service attack, it can ask clients to compute the reverse of a secure one-way function by brute force before they are allowed to carry on with rest of the protocol. The cost of the brute force computation is parameterized by revealing some input bits to the client and letting it find the remaining ones.

However, Juels and Brainard concentrate on the SYN attack and don't consider DOS attacks against authentication protocols. They, in fact, suggest that a certificate-based client authentication solves the DOS problem and, hence, would not benefit from the puzzles. We disagree with this and use the client puzzles to generalize the design principles of the DOS-resistant KAP to any authentication protocol. We also improve the efficiency of the client puzzles by reducing the length of the puzzle and its solution, by minimizing the number of hash operations needed in the verification of the solution (at the cost of slightly coarser puzzle difficulty levels), and by observing that the puzzles can in some networks be broadcast to the potential clients.

3 Client puzzles

The server in an authentication protocols can ask the client to solve a puzzle before the server creates a protocol state or computes expensive functions such as exponentiation.

C		S
	$\xleftarrow{\text{Puzzle}}$	S does not save state data or do expensive computations.
C commits its resources into solving the puzzle.	$\xrightarrow{\text{Solution}}$	
		S verifies the solution. **S may now commit resources to expensive parts of the authentication.**

Fig. 1. Server suspecting a DOS attack sends puzzles to new clients

A good puzzle should have the following properties, the two last of which are new in comparison to [10]:

1. Creating a puzzle and verifying the solution is inexpensive for the server.
2. The cost of solving the puzzle is easy to adjust from zero to impossible.
3. The puzzle can be solved on most types of client hardware (although it may take longer with slow hardware).
4. It is not possible to precompute solutions to the puzzles.
5. While the client is solving the puzzle, the server does not need to store the solution or other client-specific data.
6. The same puzzle may be given to several clients. Knowing the solution of one or more clients does not help a new client in solving the puzzle.
7. A client can reuse a puzzle by creating several instances of it.

The puzzle we use is the brute-force reversal of a one-way hash function such as MD5 or SHA. This is a practical choice because the hash functions are computable with a wide variety of hardware and the brute-force testing of different inputs is likely to remain the most efficient way for computing the inverse of these functions. (The difficulty of solving number-theoretic puzzles like factoring may depend heavily on the sophistication of the algorithms used by client.)

To create new puzzles, the server periodically generates a nonce N_S and sends it to the clients. To prevent the attacker from precomputing solutions, the nonce needs to be random and not predictable like, for example, time stamps. (About 64 bits of entropy is sufficient to prevent the attacker from creating a database of solutions from which it could frequently find a matching nonce. Birthday-style attacks that may result in occasional matches will not do much harm here.) The server also decides the difficulty level k of the puzzle. N_S and k together form the puzzle that is sent to the client.

To solve the puzzle, the client generates a random nonce N_C. The purpose of this nonce is twofold. First, if the client reuses a server nonce N_S, it creates a new puzzle by generating a new N_C. Second, without the client nonce an attacker could consume a specific client's puzzles by computing solutions and sending them to the server before the client does. (About 24 bits of entropy is enough to prevent an attacker from exhausting the values of N_C given that N_S changes frequently.)

The client solves X (and Y, which will be discarded) from the following equation by brute force and sends the solution X to the server.

$$h(C, N_S, N_C, X) = \overbrace{000\ldots000}^{\text{the } k \text{ first bits of the hash}} \underbrace{Y}_{\text{the rest of the hash bits}}$$

h	=	a cryptographic hash function (e.g. MD5 or SHA)
C	=	the client indentity
N_S	=	the server's nonce
N_C	=	the client's nonce
X	=	the solution of the puzzle
k	=	the puzzle difficulty level
$000\ldots000$	=	the k first bits of the hash value; must be zero
Y	=	the rest of the hash value; may be anything

The server changes the value of N_S periodically (for example, every 60 seconds) to limit the time clients have for precomputing solutions. As long as the server accepts solutions for a certain value of N_S, it must keep book of the correctly solved instances so that the solutions cannot be reused.

The above puzzle satisfies the criteria for good puzzles. The server only needs to generate a single random nonce to create a new puzzle. The only efficient way to solve the puzzle is to try values of X by brute force until a solution is found. The cost of solving the puzzle depends exponentially on the required number k of zero bits in the beginning of the hash. If $k = 0$, no work is required. If $k = 128$ (for MD5), the client must reverse the entire one-way hash function, which is computationally impossible. Reasonable values of k lie between 0 and 64. The puzzle can be solved on a wide range of hardware because the hash functions are one of the simplest cryptographic operations.

We believe the exponential scale for puzzle difficulty is sufficient for applications. That way, the server can verify the solution in a constant time with a single hash. A more accurate scale could be achieved by combining several puzzles with varying size k. This would, however, increase the cost of verification. (Combining puzzles of varying size would achieve the same granularity of puzzle difficulty as the sets of equal-size subpuzzles in [10] but at a slightly lower cost to the server.) The parameter k should normally be set to zero and increased gradually when the server resources are close to being exhausted. Later, when the server again has free capacity, it is time to decrement k. This way, the correct value is found dynamically and we do not need to know the exact cost of the brute-force computation for the range of parameter values.

The solutions cannot be precomputed because the same N_S is used only for a short time. The client identity C is used as a parameter in the puzzle so that solving the puzzle for one C does not help in finding solutions for another C. This means that it is expensive for one client to impersonate several clients because the solution must be recomputed for each client. The client may reuse the same N_S by solving the puzzle with a new N_C.

Finally, it is feasible to use the above puzzle in the stateless phase of the protocol because the same periodically generated N_S may be used for all clients. This makes it also possible to broadcast the puzzle.

4 An authentication protocol

We will now look at how the client puzzles are used to improve the DOS-resistance of an authentication protocol. In the protocol of Fig. 2, C and S authenticate each other with digital signatures and nonces. The protocol can easily be extended into a key exchange by including encrypted key material in the messages.

The protocol normally begins with a broadcast message from the server. In a non-broadcast network, this message may be sent individually to clients that greet the server with a Hello message. The server broadcast consists of a random nonce N_S and a parameter k that determines the difficulty of the puzzle. The server generates a fresh nonce periodically and sends the same values N_S, k to all clients during that period. The message may timestamped and signed to prevent an attacker from broadcasting false puzzles. The timestamp T_S and the signature can be omitted if the potential DOS attacks againts the clients are not a concern. The client then generates a nonce N_C, solves the puzzle, and returns the signed answer to the server. The client may reuse a recent puzzle by generating a new nonce N_C.

The server first checks that the same client C has not previously sent a correct solution with the same N_S, N_C. Replayed solutions are ignored. The server verifies the client's solution to the puzzle by computing the hash and, only after seeing that it is correct, verifies the signature and continues with the last message of the authentication. The server stores the values C, N_S, N_C as long as it still considers the nonce N_S recent. If an attacker wants to induce the server to store false values of this kind or to verify false signatures, it must compute a solution to a new puzzle for every stored data item and verified signature.

C		S
	$\xrightarrow[\text{OPTIONAL}]{\text{Hello}}$	
	$\xleftarrow[\text{BROADCAST}]{S_S(T_S, k, N_S)}$	S periodically decides k, generates N_S, and timestamps and signs the following message.
C verifies the timestamp T_S and signature S_S. C generates N_C and solves X by brute force from $h(C, N_S, N_C, X) = 0_1 0_2 \ldots 0_k Y$. C signs the following message.		
	$\xrightarrow{S_C(S, C, N_S, N_C, X)}$	
		S verifies that N_S is recent, C, N_S, N_C not used before, and $h(C, N_S, N_C, X) = 0_1 0_2 \ldots 0_k\, Y$. **S may now commit resources.** A stores C, N_S, N_C while N_S recent and verifies the signature S_C. **S has now authenticated C.** S signs the following message.
	$\xleftarrow{S_S(S, C, N_C)}$	
C verifies the signature S_S. **C has now authenticated S.**		

Fig. 2. DOS-resistant authentication with public-key signatures

The puzzle increases the length of the messages only minimally: one byte for k and up to about 8 bytes for the solution X. The nonces are needed in any case for the authentication. Puzzles should only be used when the server suspects it is under an attack and its capacity is becoming exhausted. Otherwise, the server can set $k = 0$. This means that there is no puzzle to solve and any value of X is ok.

5 Conclusion

We showed how the robustness of authentication protocols against denial of service attacks can be improved by asking the client to commit its computational resources to the protocol run before the server allocates its memory and processing time. The server sends to the client a puzzle whose solution requires a brute-force search for some bits of the inverse of a one-way hash function. The difficulty of the puzzle is parameterized according to the server load. The server stores the protocol state and computes expensive public-key operations only after it has verified the client's solution. The puzzles protects servers that authenticate their clients against resource exhaustion attacks during the first messages of the connection opening before the client has been reliably authenticated. It should be noted, however, other techniques are needed to protect individual clients against denial of service and to prevent exhaustion of communications bandwidth.

6 Acknowledgments

Tuomas Aura was funded by Helsinki Graduate School in Computer Science and Engineering (HeCSE) and by Academy of Finland projects #44806 and #47754.

References

1. Edward Amoroso. A policy model for denial of service. In *Proc. Computer Security Foundations Workshop III*, pages 110–114, Franconia, NH USA, June 1990. IEEE Computer Society Press.
2. Tuomas Aura and Pekka Nikander. Stateless connections. In *Proc. International Conference on Information and Communications Security (ICICS'97)*, volume 1334 of *LNCS*, pages 87–97, Beijing, China, November 1997. Springer Verlag.
3. TCP SYN flooding and IP spoofing attack. CERT Advisory CA-96.21, CERT, November 1996.
4. William H. Cunningham. Optimal attack and reinforcement of a network. *Journal of the ACM*, 32(3):549–561, July 1985.
5. Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Advances in Cryptology - Proc. CRYPTO '98*, volume 740 of *LNCS*, pages 139–147, Santa Barbara, CA USA, August 1992. Springer-Verlag.
6. Virgil D. Gligor. A note on the denial-of-service problem. In *Proc. 1983 IEEE Symposium on Research in Security and Privacy*, pages 139–149, Oakland, CA USA, April 1983. IEEE Computer Society.
7. Dan Harkins and Dave Carrel. The Internet key exchange (IKE). RFC 2409, IETF Network Working Group, November 1998.

8. Shouichi Hirose and Kanta Matsuura. Enhancing the resistance of a provably secure key agreement protocol to a denial-of-service attack. In *Proc. 2nd International Conference on Information and Communication Security (ICICS'99)*, pages 169–182, Sydney, Australia, November 1999. Springer.
9. P. Janson, G. Tsudik, and M. Yung. Scalability and flexibility in authentication services: The KryptoKnight approach. In *IEEE INFOCOM'97*, Tokyo, April 1997.
10. Ari Juels and John Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proc. 1999 Network and Distributed Systems Security Symposium (NDSS)*, pages 151–165, San Diego, CA, February 1999. Internet Society.
11. Phil Karn and William A. Simpson. Photuris: Session-key management protocol. RFC 2522, IETF Network Working Group, March 1999.
12. Catherine Meadows. A formal framework and evaluation method for network denial of service. In *Proc. 12th IEEE Computer Security Foundations Workshop*, pages 4–13, Mordano, Italy, June 1999. IEEE Computer Society.
13. Jonathan K. Millen. A resource allocation model for denial of service. In *Proc. 1992 IEEE Computer Society Symposium on Security and Privacy*, pages 137–147, Oakland, CA USA, May 1992. IEEE Computer Society Press.
14. Cynthia A. Phillips. The network inhibition problem. In *Proc. 25th Annual ACM Symposium on the Theory of Computing*, pages 776–785. ACM Press, May 1993.
15. Christoph L. Schuba, Ivan V. Krsul, Markus G. Kuhn, Eugene H. Spaffold, Aurobindo Sundaram, and Diego Zamboni. Analysis of a denial of service attack on TCP. In *Proc. 1997 IEEE Symposium on Security and Privacy*, pages 208–223, Oakland, CA USA, May 1997. IEEE Computer Society Press.
16. William A. Simpson. IKE/ISAKMP considered harmful. *;login;*, 24(6):48–58, December 1999.
17. Che-Fn Yu and Virgil D. Gligor. A formal specification and verification method for the prevention of denial of service. In *Proc. 1988 IEEE Symposium on Security and Privacy*, pages 187–202, Oakland, CA USA, April 1988. IEEE Computer Society Press.