HELSINKI UNIVERSITY OF TECHNOLOGY
DIGITAL SYSTEMS LABORATORY
Series **A:** Research Reports
No. **46**; May 1997
ISSN 0783–5396
ISBN 951–22–3607–9

STATELESS CONNECTIONS

Tuomas Aura and Pekka Nikander
Department of Computer Science
Helsinki University of Technology
Otaniemi, FINLAND

Helsinki University of Technology
Department of Computer Science
Digital Systems Laboratory
Otaniemi, Otakaari 1
P.O.Box 1100, FIN–02015 HUT, FINLAND

HELSINKI UNIVERSITY OF TECHNOLOGY
DIGITAL SYSTEMS LABORATORY
Series **A:** Research Reports
No. **46**; May 1997

ISSN 0783–5396
ISBN 951–22–3607–9

Stateless connections

TUOMAS AURA AND PEKKA NIKANDER

Abstract: We describe a transformation of stateful connections or parts of them into stateless ones by attaching the state information to the messages. Message authentication codes are used for checking integrity of the state data and the connections. The stateless server protocols created in this way are more robust against denial of service resulting from high loads and resource exhausting attacks than their stateful counterparts. In particular, stateless authentication resists attacks that leave connections in a half-open state. Examples of problems related to statefulness and solutions to them shown for the X.509, ISAKMP, TCP and HTTP protocols.

Keywords: stateless connections, denial of service, cryptographic protocols, robust design, SYN-flooding attack

Printing: TKK Monistamo; Otaniemi 1997

Helsinki University of Technology
Department of Computer Science
Digital Systems Laboratory
Otaniemi, Otakaari 1
B.O.Box 1100, FIN–02015 HUT, FINLAND

Phone: $\frac{(09)}{+358\ 9}$ 4511
Telex: 125 161 htkk fi
Telefax: +358 9 451 3369
E-mail: lab@saturn.hut.fi

Contents

1 Introduction

When designers of communications and operating systems want the systems to survive high loads and to continue functioning under malicious flooding with requests, they resort to the application specific engineering principles that have developed over time [8]. The computer security research community, on the other hand, has given surprisingly little attention to such principles. In theories on denial of service, the focus has been on absolute qualities expressible with formulas of logic rather than on the comparative performance of different designs under attacks.

In this paper we suggest a family of design principles that can help in building the systems to be more robust against the denial-of-service threat from both malicious attacker and masses of honest clients. Our goal is to make attacks more complex and resource demanding, to limit the number of potential attackers, and to minimize the impact of attacks by making recovery after an attack easier.

The basic idea is to combine the strengths of connection-oriented protocols with the characteristic robustness of stateless services. This is done by saving the protocol state in the client rather than in the server. At the cost of transferring state data between the client and the server, we are able to maintain connections with a stateless server. The state and connection data is then protected with secret key cryptography.

The present proliferation of electronic commerce and business that rely entirely on open networks in their operation highlights the importance of reliable services. In the open networks, it has become increasingly difficult to differentiate between malicious attacks and resource exhaustion by unexpectedly high demand for a service. It therefore seems appropriate that the same techniques are applicable to both shielding services against overload by honest clients and against request flooding attacks. We will show how the new concepts of this paper can be used to enhance the reliability of several layers in the Internet protocol suite.

The paper is structured as follows. We first discuss resource exhaustion problems that are typical of stateful services in Sec. 2. In Sec. 3 we present a transformation of stateful protocols into stateless ones, gradually augment their security, and compare the behavior of the new protocols and the original ones under denial-of-service attacks. Sec. 4 describes how statelessness can be utilized at the parts of the protocol where it is most beneficial. Sec. 5 and 6 discuss applications of the techniques in schematic security protocols and in real communication protocols. Sec. 7 summarizes the presented ideas.

2 Denial of service in stateful protocols

The need to store connection information makes stateful protocols vulnerable to two kinds of denial-of-service threats that are not present in stateless protocols. Firstly, there can be too many clients trying to open simultaneous connections to the server. When a limit on the number of connections becomes exhausted, further requests must be refused. We discuss this weakness in Sec. 2.1. Secondly, a malicious attacker can consume the resources of the server in order to deny others the service. Sec. 2.2 describes such attacks.

2.1 Running out of connection table space

In stateful protocols, there is always an upper limit on the number of clients that can connect to a server simultaneously. It exists regardless of the possibility that the clients might still be satisfied with a thinner share of the server capacity. Eventually, the restriction is caused by the limited space that is available for storing connection state information. When more and more clients attempt to connect to the server, its storage space becomes exhausted and new connections must be refused. In the worst case, the connected clients do not consume the full capacity of the server, and the server remains partially idle while the refused clients are waiting to connect. Of course, the connection table and the memory space of the server can be expanded to accommodate more clients, but at some point, the maintenance of the growing connection data will become too expensive. An indication of this is that many operating systems optimize performance by storing connection states in fixed-size tables.

An unintentional mistake by a client or a communication error may also leave a connection in an eternally open state. The client can forget to close the connection, lose its connection table data, or it may be unable to reach the server for the closing commands. In the end, the stale connections must be purged from the server table; but it is difficult do this without sometimes closing valid connections.

One way to describe the problem with stateful servers is that their behavior under stress is unideal. Fig. 1 compares the speed of stateful and stateless service when the number of clients or connections to the server increases. The service quality in both systems shows a smooth decline as more clients connect to the server. At a certain point, the stateful server stops accepting new connections. The quality of service for already connected clients remains constant, but new clients are refused completely. Thus, the stateful protocol exhibits a sudden change in its behavior that is difficult to take into account in performance models and correctness proofs. In the stateless server, the service quality continues its slow decline with increasing number of clients.

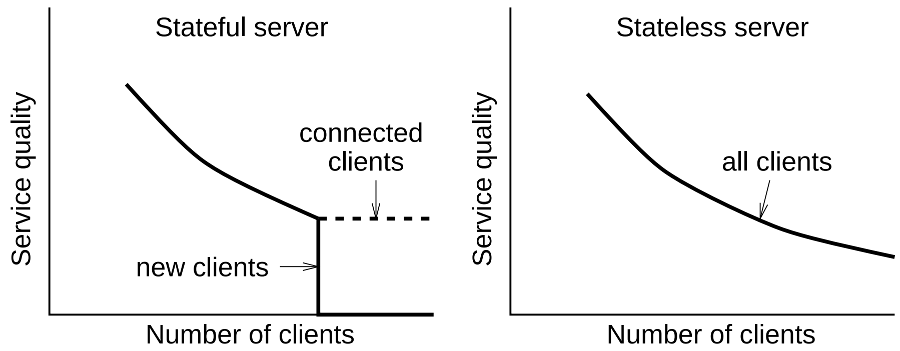

Figure 1: Service quality as a function of load in stateless and stateful servers

2.2 Attacks exhausting connection limit

The difference between stateful and stateless servers becomes highlighted when malicious attackers consume resources in an attempt to deny them to others. Attackers can exploit the limit on the number of simultaneous connections, blocking any new connections. If an attacker succeeds in opening enough connections, the server will refuse new clients. Moreover, if the open connection does not obligate the client to actively use a service, the malicious agent needs to sacrifice only relatively small computation and communication resources in order to continually block other clients' access to the service.

In the attack, the memory space for saving connection states becomes a critical resource, although it was not meant to be that. The attack is particularly disturbing, because the attacker is not necessarily utilizing the service but merely consuming a secondary resource that is needed for accessing it.

Of course, the attacker tries to keep the connections in a state where the number of simultaneous clients is as limited as possible, the effect of blocking maximally harmful, and its own efforts minimal. Furthermore, the attacker would prefer not to reveal its own identity to the server. All these conditions are best met at the transient states during connection opening, where not many connections are expected to stay for a long time, and where the client has not yet fully identified itself. Therefore, the danger of denial-of-service attacks that exploit vulnerabilities caused by statefulness is greatest at the beginning of connections.

3 Making connections stateless

Since saving the connection states creates problems, we would like agents to maintain connections without saving any data. This can be done by passing the state information between the protocol principals along the messages. In Sec. 3.1 we describe a general transformation that removes saved states from protocols. Sec. 3.2 adds an integrity check on the state information and Sec. 3.3 continues by ensuring integrity of the entire connection. Sec. 3.4 points out the main advantages of statelessness by comparing denial-of-service attacks against stateless protocols and their stateful counterparts.

There are not many precedents in the literature that would have used statelessness in a way similar to our ideas. The closest resemblant is the HTTP cookie mechanism [7] that could be used for the same purposes. Perrochon [10] suggests translation servers that function as stateless front-ends for stateful services. Some of the benefits that we gain from statelessness could also be given by the translation servers.

3.1 Transformation from stateful into stateless

We will assume for a while that all parties of communication are trustworthy and the communication channels are reliable. On this assumption, we can transform any stateful two-party protocol into a stateless protocol that achieves the same goals. In every protocol step, the principals of the new protocol avoid saving state information by sending it along the message to the other party and erasing it from their own memory. In the next protocol step, the same state information is always returned to the sender. That is, the principals pass the responsibility of saving the state data to each other and receive the data back right at the time when they need it. Since we assume that nobody takes advantage of the unprotected state information by reading or altering it, the resulting protocol is equivalent to the original one. The only difference is the new way of storing the state data and the communication costs caused by it.

The following protocol schemas demonstrate the idea of repeatedly sending and receiving the state information. Every message in the stateless variant contains the actual message to be transferred, the receiving principal's state, and the sending principal's state. After sending a message, the principals discard any state information from their memory.

A stateful protocol:

1.	$A \longrightarrow B$	Msg_1
		State of A is $State_{A1}$.
2.	$B \longrightarrow A$	Msg_2
		State of B is $State_{B1}$.
3.	$A \longrightarrow B$	Msg_3
		State of A is $State_{A2}$.
4.	$B \longrightarrow A$	Msg_4
$\vdots$	$\vdots$	$\vdots$

An equivalent stateless protocol:

1.	$A \longrightarrow B$	$Msg_1, State_{A1}$
2.	$B \longrightarrow A$	$Msg_2, State_{B1}, State_{A1}$
3.	$A \longrightarrow B$	$Msg_3, State_{A2}, State_{B1}$
4.	$B \longrightarrow A$	$Msg_4, State_{B2}, State_{A2}$
$\vdots$	$\vdots$	$\vdots$

Although any two-party communication protocol can, assuming security problems do not exist, be transformed to a stateless equivalent in this way, it is usually sufficient to make the server in a client/server system, or the responder in symmetric protocols, stateless. Again, the protocol schema below demonstrates the transformation. The server attaches its state to the messages going to the client. The client returns the state information in its next message to the server.

A stateless service protocol:

1.	$C \longrightarrow S$	Msg_1
2.	$S \longrightarrow C$	$Msg_2, State_{S1}$
3.	$C \longrightarrow S$	$Msg_3, State_{S1}$
4.	$S \longrightarrow C$	$Msg_4, State_{S2}$
$\vdots$	$\vdots$	$\vdots$

It is possible to do a similar conversion to multi-party protocols if the messages travel suitably. There one must take care that the state information is returned to the stateless principal right at the time when it is needed.

The main reason why one would want to transform protocols into stateless ones is that it makes the system behavior more ideal. A stateless server does not have an upper limit on the number of clients that can be connected to it or be in a particular phase of the protocol run simultaneously, because it does not need any tables for saving connection parameters. When there is no limit on the number of clients, the limit cannot be exploited by denial-of-service attacks. The ideal protocol properties also simplify quantitative analysis of system behavior under stress. Instead of failing unexpectedly at some threshold, the performance degrades gradually as the frequency of service requests increases.

Moreover, the stateless protocol moves the responsibility of reliably saving state information from the server to the client. Since the client has requested the service, it is better motivated to take care of the information and to recover from error conditions and data loss. This is advantageous, because a server with several clients does not want to reserve its resources for a single client for the indefinite time that may pass before the client continues the message exchange.

Another application for stateless protocols is information services that divide the server load between several identical machines. The servers can be geographically distributed or clustered in one place. When the servers are stateless, client requests can be routed to an arbitrary server without giving any consideration to where the previous messages were processed. Routing decisions and reply addresses can be changed dynamically in order to level the load on the servers and to minimize communication costs.

As a drawback, the stateless protocols require additional bandwidth for transferring the state data. If the states are large, the cost may be too high. File or document servers are therefore examples of promising applications for stateless protocols while intensely interactive sessions most often are not. Also, the communication channels should be relatively reliable, because retransmission of damaged messages is more difficult to arrange than in stateful protocols.

Although stateless protocols implemented in the above way resist denial-of-service attacks by server flooding, they are prone to much more serious attacks. We will analyze and improve the security of the protocols in the next three sections.

3.2 Integrity and confidentiality of the state data

When the state data is repeatedly transferred through insecure channels, its integrity and confidentiality become an important security concern. In this section, we will show how to protect the stateless protocols against altering and disclosure of the state data. These mechanisms will then be used for enhancing the overall security of the protocol in the later sections.

The integrity is naturally protected by signing the data. The server signs the data before sending it to the client and checks the signature on returned state data. A big advantage here is that the signature is checked exclusively by the signer itself. This makes possible the use of secret key signatures, i.e. message authentication codes, instead of much more costly public key signatures.

Also, the freshness of the state data should be checked in order to limit the number of times the data can be replayed. Timestamps can be applied liberally, because they are checked by their creator against the same clock that is used for the timestamping. (Distributed servers that accept state data packets created by each other must, however, have synchronized clocks. The accuracy does not need to be very high, because the timestamp lifetimes will be long.)

The improved transformation of a stateful service into a stateless one is illustrated by the protocol schema below. Every message leaving the server contains a timestamped state of the connection, signed with a key K_S^s known only by the server. The state is then returned to the server along the next message from the client.

1.	$C \longrightarrow S$	Msg_1
2.	$S \longrightarrow C$	$Msg_2, S_{K_S^s}(T_{S1}, State_{S1})$
3.	$C \longrightarrow S$	$Msg_3, S_{K_S^s}(T_{S1}, State_{S1})$
4.	$S \longrightarrow C$	$Msg_4, S_{K_S^s}(T_{S2}, State_{S2})$
$\vdots$	$\vdots$	$\vdots$

The secret key signature can be implemented as a message authentication code,

$$Msg, S_{K_S^s}(T_S, State) = Msg,\ T_S, State,\ MAC_{K_S^s}(T_S, State).$$

Sometimes there is redundancy in the actual message and the state information. In that case, it is not necessary to repeat the redundant data. Below, $State'$ contains only the data that is not in the message Msg.

$$Msg, S_{K_S^s}(T_S, State) = Msg,\ T_S, State',\ MAC_{K_S^s}(T_S, State).$$

On receipt of the state data, the server should check the timestamp against the same clock as is used for timestamping. Expired messages can be simply ignored. The client is responsible for taking any corrective steps after such error conditions. It is necessary to allow long lifetimes for the states so that the data does not expire before the client wants to continue the message exchange and succeeds in sending its next request. Hence, the timestamp lifetime should be longer than the expected duration of a denial-of-service attack, usually on the order of a few days. The goal is not to control immediate replays but to limit the amount and distribution of message material that is available for replay attacks. The issue of replays is discussed in more detail in the next two sections.

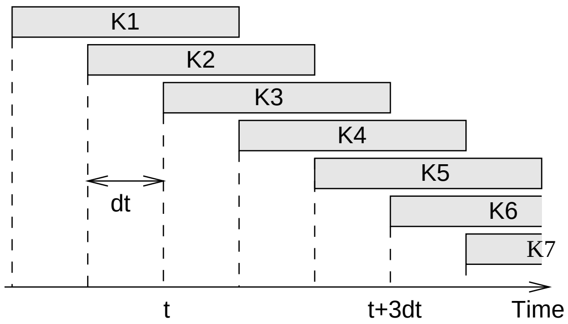

Figure 2: Key validity periods

Another method for checking freshness is to change signature keys periodically. A few of the newest keys should be kept in the server's memory for accepting fresh messages. State information signed with older keys is then discarded as invalid. The period of generating new signature keys becomes thus the resolution of message expiration times. The period of validity is the period of key generation multiplied by the number of newest keys accepted. For example, K3 in Fig. 2 is generated at time t and it remains valid until time $t + 3dt$. Keys with different lifetimes can be used for different purposes. This is feasible, because even in a complicated key arrangement, the space and computation required for maintaining the keys is a constant, i.e. does not depend on the number of clients or on the amount of traffic in the system. In any case, a key identifier should be added to the messages in order to avoid trying out several keys.

Also, any secret state data is easily concealed by encrypting it with a secret key K_S^e known only by the server. The mechanism is illustrated below.

$$\boxed{\begin{array}{rll} i. & C \longrightarrow S & Msg_i,\ T_{S,i}, E_{K_S^e}(State_{S,i}),\ MAC_i \\ i+1. & S \longrightarrow C & Msg_{i+1},\ T_{S,i}, E_{K_S^e}(State_{S,i}),\ MAC_i \end{array}}$$

As before, the authenticity of the state data is protected with a message authentication code

$$MAC_i = MAC_{K_S^s}(T_{S,i}, State_{S,i}).$$

In the rest of the paper we assume that secret parts of the server state are transferred in encrypted form, even when this is not shown explicitly.

3.3 Integrity and confidentiality of the connection

So far, the described stateless protocols have not protected the integrity or confidentiality of the messages in any way. Such protocols are vulnerable to a wide variety of attacks. In this section we will describe a technique for authenticating and encrypting stateless connections. These measures will protect the protocol against the same kind of attacks as they would protect a stateful protocol.

There is, however, an additional reason for the security enhancements. Namely, the statelessness opens a whole new range of attacks against the connection integrity: replay of connection states. The stateless principals have no means for detecting the replays, because they cannot remember which messages have already been received and processed. An integrity check that links the state data to the actual messages will limit the ways in which third parties can utilize recorded server states in attacks.

In the following protocol schema, the client and the server have a shared secret key K_{CS} for signing and encrypting connection data. (Key distribution will be discussed in Sec. 5.1.) The server naturally passes the key to the client along with all other state data.

$$\boxed{\begin{array}{rll} i. & C \longrightarrow S & Msg'_i,\ T_{S,i}, E_{K^e_S}(K_{CS}), State_{S,i},\ MAC_i \\ i+1. & S \longrightarrow C & Msg'_{i+1},\ T_{S,i}, E_{K^e_S}(K_{CS}), State_{S,i},\ MAC_i \end{array}}$$

The familiar message authentication code is

$$MAC_i = MAC_{K^s_S}(T_{S,i}, K_{CS}, State_{S,i}),$$

and the protected messages themselves take the form

$$Msg'_i = E_{K_{CS}}(Msg_i), MAC_{K_{CS}}(i, Msg_i, MAC_i).$$

It is necessary to encrypt the messages only if their contents are secret. Signatures, on the other hand, should be used in this way in all systems where replay attacks are considered a threat. The signatures alleviate the effect of replays, because they bind state data together with the corresponding messages. After these enhancements, the only harm the attacker can do with recordings is to replay messages to the stateless server. The replays result in duplicate responses to the client. The protocol designer should ensure that the client is able to detect the duplicates or is not affected by them.

When protection against replay attacks is needed, the above connection integrity check should also be made on anonymous connections. Although it is impossible to securely exchange a session key with an anonymous client, this limits the possible attacks to ones that also manipulate the key exchange process. Thus, it becomes useless to record and replay random messages without understanding and altering them.

The described measures effectively shield the system against replay attacks by third parties. A dishonest client, on the other hand, cannot be stopped from replaying state data from earlier stages of the protocol run. By replaying old states, the client can return to any previous point in the protocol run. Consequently, the client can execute parts of the protocol several times or go through several alternative branches of the protocol run. If the protocol has a point of choice for a principal, it can come back later and test how the other alternatives would have ended. In some protocols, the possibility of collecting information from several alternative execution paths is catastrophic. For example, in many zero knowledge proofs, allowing two choices for the prover or verifier could result in a false proof or disclosure of secret knowledge, respectively. Therefore, not all protocols can be securely made stateless. The transformation is not suitable for protocols where combined information from a small number of alternative runs could reveal something more to the client than a single deterministic run. Luckily, most existing protocols are deterministic enough so that the client will not be able to cause any damage by replaying the states.

3.4 Replays and denial of service

Another way to exploit the inability of a stateless server to detect replays is to exhaust the server's capacity by continuously resending old messages. The attacker must be someone with access to the communication channel between the server and its clients. This is a denial-of-service attack, as was the attack that we avoid by making the protocols stateless. The two attacks, however, exploit a different vulnerability, and we must compare the harm caused by them.

We first consider the stateless server under replay flooding attack. The best the attacker can do is to replay messages at the maximum throughput rate of the server so that no service capacity remains for real clients. (This is the worst case scenario. In most communication systems, some legitimate messages will still get through, but more optimistic estimates would require detailed knowledge of the structure of the communications system.)

Another danger is that the legitimate connections start breaking, because the time stamps on the state data expire while the attacker blocks the service. This can be avoided by making the timestamp lifetimes longer than it takes to detect the attack and to take countermeasures. After the attack, the clients can continue the connections if they still find them useful. This is the reason why the time stamps should last days rather than seconds or hours. The timestamp lifetime, however, should not be infinite, because we want to limit the amount of replayable material in circulation.

Next, we consider the behavior of a stateful server when an attacker is creating new connections and leaving them open. If the attacker opens connections at the maximum rate C allowed by the server, no other clients can access the service. The server must also purge the idle connections from its memory after a certain time to make space for new ones. If the server has enough memory to save the state of M connections, each connection will remain in the memory at most time M/C. In a typical system, this time will be much shorter than the duration of an average attack. Thus, the attacker is able to break the existing connections.

Comparing the stateless and stateful protocols under their characteristic attacks, we observe that an equal rate of replays against the stateless server and connection openings against the stateful server have approximately the same effect on the service quality during the attack. Both servers can be totally clogged by the attacks. The clients of the stateless server, however, recover much faster after an attack.

Another big advantage for the stateless server is that the described worst-case scenario is less likely to happen for it. We assumed that the attacker can record enough messages for the replay attacks. On a large network, most agents never see any such messages. Hence, the number of agents that can mount the replay attack against the stateless server is very small in comparison to the group that can open useless connections to the stateful server. For example, on the Internet, anyone in the world can open connections to almost any public server, but very few people can observe other peoples' connections to a particular server.

We conclude that stateless protocols are, in general, more robust against denial-of-service attacks than their stateful counterparts. The stateless protocol makes recovery after an attack easy and dramatically reduces the number of potential attackers. Nevertheless, the effect of replay attacks should be evaluated for each particular system before transforming the protocols into stateless equivalents.

4 Partially stateless protocols

Although we have described a general transformation of entire protocols into stateless ones, in practical systems, most benefits of statelessness are obtained at some specific parts of the protocol. It thus makes sense to have only those parts stateless. Sections 4.1 and 4.2 discuss statelessness at two different periods in the protocol run, in connection opening and during idle periods, respectively. In Sec. 4.3 we consider stateless layers in protocol stacks. Sec. 4.4 overviews protocol optimizations such as state caching and packet windowing.

4.1 Stateless handshake

Attackers usually do not want to reveal their identity. Neither do they want to pay for the services that they are blocking. Therefore, the robustness of stateless protocols is most beneficial at those parts of the run where the client remains unidentified. In many protocols, the client must be authenticated as a legitimate user of the service at the beginning of the connection. Thus, the only time when the client has not yet been reliably identified are the first few messages of the connection. Such protocols should avoid saving the state before the authentication of the client has been completed. After that, the server can move to stateful mode in order to optimize the rest of the message exchange.

A few opening messages of a connection often adequately indicate the client's real intentions. In the opening dialogue, the client can show its commitment to honest use of the service with the following actions:

- The client authenticates itself cryptographically to the server and can be held responsible for misuse of the service.
- The client proves its access rights to the service, or pays for it.
- The client does something resource-demanding that an attacker cannot afford to do repeatedly.
- The client responds to a message from the server and, thus, has not forged the return address.

The last point is important, because the communications address of the client in its opening message could be forged. If the client responds to a message from the server, the server at least knows how to reach the client, which can be construed as a level of authentication. Knowing the address of the attacker helps the server administration in resolving problems off-line. In anonymous services that are available to everyone, the return address may be the only available identifier, but it is often enough. In the following protocol schema, the server is stateless only during the first roundtrip from it to the client and back.

1.	$C \longrightarrow S$	$Msg1$
2.	$S \longrightarrow C$	$Msg2, S_{K_S^s}(T_{S1}, State_{S,1})$
3.	$C \longrightarrow S$	$Msg3, S_{K_S^s}(T_{S1}, State_{S,1})$
		The return address is valid.
		S moves to stateful mode.
4.	$S \longrightarrow C$	$Msg4$
$\vdots$	$\vdots$	$\vdots$

Anonymity is also possible if the client proves its access rights to the server without revealing its identity. This commonly occurs when the client pays for the service with an anonymous digital payment method. An attacker usually does not want to pay for the services.

4.2 Statelessness during idle periods

In addition to the first handshake with the clients, the server may benefit from statelessness during long idle periods in communication. In long-lived protocol runs, activity often ceases and is resumed later. In a stateful protocol, the server will have to maintain connection state data throughout the idle periods. The server can be relieved of this duty by sending the state information to the client for temporary storage. The client has the responsibility for saving the data and recovering from data losses. The server's memory will be freed for use by other connections.

Depending on the protocol, the server can send its state to the client with every message, after certain messages, or after the connection has been idle for a threshold time. The client can either return the state in its next messages or check first whether the server still has the state in its cache.

The main benefit from passing the state to the client on long connection is that the limit on the number of open connections is removed. The client is also better motivated to take care of storing its connection state than the server is.

4.3 Stateless layers in protocol stacks

Communication protocols are normally organized in stacks where each layer uses the services of the layer below and provides services to the layer above.

A protocol, as we have used the word, either forms a layer in itself or is a part of a layer protocol. Even if one layer is stateless, the others might not be. This is in accordance with the philosophy behind the layered design: each layer can be designed independently of the others as long as it provides the specified services to the layer above. If we, however, consider availability and security to be an essential part of the service specification, some layers may have to be implemented as stateless.

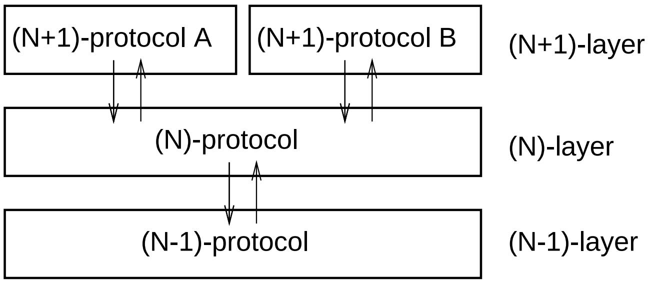

Figure 3: Protocol stack

In the OSI terminology [. ISO 7498 .], when an application layer protocol on the top of the stack is stateless, it usually makes sense to have all layers down to the network layer equally stateless. If denial-of-service attacks from the local networks are expected to be a problem, the data link layer could also be made stateless, but in that case, there are probably much more serious vulnerabilities that should be considered first.

A layer protocol can have more than one upper layer protocol accessing it (see Fig. 3). Then, regardless of whether the upper protocols are stateless or not, it is beneficial to make the lower layers stateless. The reason is that the alternative upper layer protocols should be able to operate independently of the resources consumed by each other and attacks against each other.

To accomplish this, it is not always necessary to send the state data to the other end of the connection. The lower layer could just as well pass the information to the upper layers for storage. That way the parallel upper layer protocols are still independent of each other. The lower layer protocol must somehow obtain the state data from the upper layer protocol every time it processes data for that particular protocol. Thus, there must be a mechanism for requesting the state information from the correct upper layer protocol when the lower layer receives messages over the connection.

4.4 State caching and packet windowing

Server overloads and denial-of-service attacks occur only occasionally. For the rest of the time, one would like to retain the many performance optimizations offered by stateful protocols. In this section we will see that most such optimizations can be preserved in a stateless service by caching state data in the server.

State caching means that the stateless server sends its state to the client as usual, but it also maintains the state in its memory as long as there is enough space for it. Under normal server load, a caching server can behave just as a stateful server except for the slight performance penalty for transmitting the state data. It can, for example, detect lost and duplicated messages, report the errors to the client, collect statistics on the channel characteristics and dynamically adjust the transfer rate and packet size for optimum performance. If server memory becomes scarce or the maintenance of the state data too burdensome, the server can purge the data from its memory any time. In this way, the system can have most of the performance benefits of stateful servers while being as resistant to denial-of-service attacks as any stateless protocol principal. Admittedly, it may be difficult to design state caching servers with even near-optimal performance. Nevertheless, it should be feasible to find solutions that significantly improve on the robustness of stateful protocols and on the efficiency of stateless ones.

It should be noted that unless the integrity of connections is protected, the state caching opens a possibility for a new kind of attack: connection hijacking. If the attacker sends a false message with correct state information to the server before the client does, the client's messages will be ignored as duplicates. Therefore, it is important to combine connection integrity check (see Sec. 3.3) with state caching whenever replay attacks are expected. Fully stateful protocols have, of course, the same problem.

Windowing is a common performance optimization technique for stateful communication. It means that a principal sends data in advance, without waiting for the other party to acknowledge. The maximum amount of data that a protocol principal can send in advance is called the window size. The window size estimates the amount of data the receiver can hold in its buffers plus any data fitting on the transmission channel. Windowing allows the sender to adjust its sending speed to the ability of the receiver to handle data.

A fully stateless server obviously cannot keep count of sent packets and received responses. Consequently, windowing in a stateless protocol has to be performed by the stateful client alone. The client can easily measure its own buffer capacity and estimate the round-trip time. The server may also be able to give additional timing and load statistics to the client. Based on this, the client can then adjust its transmission rate. A requirement for the server is that it must give the client a number of message authentication codes, representing a number of expected future states. Having several MACs, each representing a separate future server state, the client can send several requests before it receives replies from the server.

This way, the stateful client of a stateless server can use windowing. Still, a performance penalty in comparison to the stateful alternative is caused by the stateless server's inability to combine several acknowledgements to one message. On the other hand, if the server caches states, delayed acknowledgements and most other usual windowing techniques can be applied both on the client and the server side.

5 Stateless security protocols

If robustness under high load and denial-of-service attacks is important in all protocols, it is especially important in security protocols, such as authentication and key exchange. First, these protocols often operate in systems where reliability is a special concern. Second, a security protocol should make the system more trustworthy, not open any new easy points of attack. We discuss statelessness in authentication and key exchange protocols in Sec. 5.1 and in the middle of long key lifetimes in Sec. 5.2.

5.1 Stateless key exchange

Authentication and key exchange are typically the first phases of establishing a new connection. As we noted in Sec. 4.1, the server should be stateless up to the point where the client, or the connection initiator, has been positively identified. Otherwise, an attacker could anonymously open connections and leave them in the open state. If the server can trust its legitimate clients not to abuse the resources, it could safely become stateful after the authentication is complete. On the other hand, if attacks are expected from authenticated clients, the protocol should remain stateless and a shared key is needed for protecting the connection integrity (cf. Sec. 3.3).

We now demonstrate the importance of authenticating the client before the server becomes stateful. In the three-way X.509 authentication protocol [2, 6], an attacker can replay a large number of old copies of the first message. This could exhaust the space that B has reserved for saving the state of the protocol between sending Message 2 and receiving Message 3. (Note that the three-way X.509 protocol uses nonces for verifying freshness of the messages. Hence, principal B only knows that Message 1 is fresh after receiving Message 3.)

$$\begin{array}{|lll|}\hline 1. & A \longrightarrow B & S_A(N_A, B, E_B(K_{AB})) \\ 2. & B \longrightarrow A & S_B(N_B, A, N_A, E_A(K_{BA})), \\ 3. & A \longrightarrow B & S_A(N_B, B) \\ \hline \end{array}$$

In the following modification of the X.509 protocol, the responding principal B does not save the state of the protocol until it has positively authenticated A.

$$\begin{array}{|lll|}\hline 1. & A \longrightarrow B & S_A(N_A, B, E_B(K_{AB})) \\ 2. & B \longrightarrow A & S_B(N_B, A, N_A, E_A(K_{BA})), \\ & & S_{K_B^s}(T_B, N_B, A, E_{K_B^e}(K_{AB}, K_{BA})) \\ 3. & A \longrightarrow B & S_A(N_B, B) \\ & & S_{K_B^s}(T_B, N_B, A, E_{K_B^e}(K_{AB}, K_{BA})) \\ \hline \end{array}$$

The session keys in the state data have been encrypted with the secret key of the server. The protocol can be further optimized by removing data items that are repeated in both signed submessages.

$$\begin{array}{|lll|}\hline 1. & A \longrightarrow B & S_A(N_A, B, E_B(K_{AB})) \\ 2. & B \longrightarrow A & S_B(N_B, A, N_A, E_A(K_{BA})), \\ & & T_B, E_{K_B^e}(K_{AB}),\ MAC_{K_B^s}(T_B, N_B, A, K_{AB}, K_{BA}) \\ 3. & A \longrightarrow B & S_A(N_B, B), \\ & & K_{AB}(K_{BA}), T_B, E_{K_B^e}(K_{AB}), \\ & & MAC_{K_B^s}(T_B, N_B, A, K_{AB}, K_{BA}) \\ \hline \end{array}$$

The above protocol is deterministic in the sense that there is only one execution path. Its runs differ only in that fresh nonces and keys are generated every time. Therefore, an attacker could not possibly collect any interesting information by replaying old states and thus causing the principals to re-execute steps. Luckily, most cryptographic protocols have a similar deterministic nature. Key exchange and authentication protocols usually have a limited number of alternative execution paths that do not interfere with each other in any way. The only choices made by the principals are the values of the fresh data items. In a stateless version, an attacker could collect several different pairs of plaintext-ciphertext pairs (or plaintext-MAC pairs) by replaying an earlier state in order to re-execute nonce and key generation steps. The gathered information could aid cryptanalysis. In practice, however, cryptographic algorithms can resist attacks with a much larger number of known plaintext messages than the attacker could possibly collect by re-executing protocol steps. The attacker will gain no advantage from the replays as long as combining information from a small number of alternative runs does not reveal any secret data.

Most key exchange protocols aim at producing unique keys for every session and purpose. Uniqueness and freshness of protocol parameters are central issues in designing these protocols. Since stateless principals cannot distinguish between replays and original messages, they cannot recognize duplicated keys at the accuracy that is usually expected from security protocols. For the stateless principals, there is no difference between one and many sessions with the same parameters. Furthermore, the branching of the key exchange process can lead to the generation of several alternative end results. Thus, uniqueness of keys is partially lost in the stateless protocols. Nevertheless, freshness can be guaranteed in the sense that none of the fresh parameters has been used before the common beginning of the protocol runs.

5.2 Storing the session keys

After the key exchange, the keys may be used only occasionally over a long period of time. Nonetheless, the protocol principals must store the keys and other session parameters throughout the connection. The session parameters can have lifetimes lasting from hours to years. It is therefore a good idea to move this information from the server to the client for storage over long idle periods in the communication. This may be reasonable even in applications where one does not want to make the server completely stateless by always sending the data to the client.

The session parameters are usually known to both principals. Hence, it is not really necessary to send everything to the client. Often only the session key needs to be encrypted with the server's secret key and sent over the channel along with a message authentication code for the key and the rest of the session parameters. The client knows the parameters and can return them with the message authentication code in its next message. Although the client also possesses the session key, it has no secure means for transporting it to the server (apart from expensive key exchange involving public key algorithms and/or trusted third parties). Therefore, the session key must be sent both ways encrypted with the server's secret key. If necessary, the client can encrypt other confidential session parameters with the session key.

$$\boxed{\begin{array}{ll}
S \longrightarrow C & Msg1, T_S, E_{K_S^e}(K_{SES}), \\
 & MAC_{K_S^s}(T_S, C, \text{parameters}, K_{SES}) \\
 & \text{Possibly long idle period follows.} \\
S \longrightarrow S & Msg2, C, \text{parameters}, T_S, E_{K_S^e}(K_{SES}), \\
 & MAC_{K_S^s}(T_S, C, \text{parameters}, K_{SES})
\end{array}}$$

One more detail to be taken into account is that when the encrypted session keys have been transferred over the channel in this way, the server's secret key should only be used for a limited time and then completely erased from any media. That way the contents of the communication cannot later be recovered from recorded messages. In short, the key encryption key should be treated with as much care as a secret master key. It is better to use one key for encrypting session keys and other keys for signatures and less sensitive session parameters data.

6 Application examples

The principles described in this paper can be applied at any any level in the OSI protocol stack. It seems, however, that statelessness offers most benefits at the transport and application layers. For example, in the TCP/IP protocol stack, denial-of-service attacks are usually targeted either at the TCP protocol [3] or at the application layer. In UDP based protocols, the application layer protocols are the natural target [4].

In Sec. 6.1 we improve both robustness and performance of a key exchange protocol from the ISAKMP specification. Sec. 6.2 describes how the statefulness of the TCP protocol leads to the well-known SYN flooding attack. Finally, Sec. 6.3 discusses security enhancements to the HTTP cookie mechanism.

6.1 Stateless ISAKMP/Oakley

The Internet Security Association and Key Management Protocol (ISAKMP) [9] is a protocol framework for Internet-wide key management and authentication. In the following, we briefly describe a version of the Oakley key exchange [5] used in ISAKMP and show how its resistance to denial-of-service attacks can be increased while at the same time reducing the number of messages.

There are two principals, the initiator I and the responder R. In the beginning, nonces N_I and N_R are exchanged in order to to ensure that the initiator has given a correct reply address. Only after the nonce exchange will the server proceed with expensive public key operations.

The protocol uses public key signatures and Diffie-Hellman key exchange. The Diffie-Hellman parameters can be reused for several protocol runs. The freshness of the final session key is guaranteed by hashing the Diffie-Hellman key together with two key generation parameters N'_I and N'_R, one from each principal.

1.	$I \longrightarrow R$	N_I
2.	$R \longrightarrow I$	N_R, N_I
3.	$I \longrightarrow R$	$N_I, N_R, S_{K_I}(g^x, I, R, N'_I)$
4.	$R \longrightarrow I$	$N_R, N_I, S_{K_R}(g^y, R, I, N'_R, N'_I)$
5.	$I \longrightarrow R$	$N_I, N_R, S_{K_I}(g^x, I, R, N'_I, N'_R)$

The responder knows the initiator's key generation parameter to be fresh only after receiving Message 5. Nevertheless, the responder has to remember intermediate states with the following state information: the two nonces N_I and N_R, the network address of I, the Diffie-Hellman key of I, the key generation parameters of both principals, and the creation time of the state. This clearly leaves the protocol vulnerable to attacks where someone initiates a connection, executes Steps 1, 2 and 3 of the protocol, and then leaves the responder waiting forever.

We enhance the protocol by making the responder stateless until the receipt of the last message. The stateless protocol avoids the attack where an initiator leaves connections open before being authenticated. Furthermore, the initial cookie exchange is not needed, because the stateless responder is not affected by opening messages with forged return address. Thus, the protocol is more robust and has less messages than the original one. (Note that the signed messages $S(\cdots)$ below contain the message itself while the authentication codes $MAC(\cdots)$ are only hash values.)

1.	$I \longrightarrow R$	N_I, g^x, I, R, N_I'
2.	$R \longrightarrow I$	$N_R, N_I, S_{K_R}(g^y, g^x, R, I, N_R', N_I'),$
		$MAC_{K_R^s}(T_R, g^y, g^x, N_R, N_I, N_R', N_I'), E_{K_R^e}(y)$
3.	$I \longrightarrow R$	$N_I, N_R, S_{K_I}(g^x, g^y, I, R, N_I', N_R'),$
		$MAC_{K_R^s}(T_R, g^y, g^x, N_R, N_I, N_R', N_I'), E_{K_R^e}(y)$

6.2 TCP resistance to SYN flooding

Recently, a lot of attention has been paid to the the so called SYN flooding attack against the TCP/IP Transmission Control Protocol (TCP). In the TCP connection establishment, the parties exchange message sequence numbers. The first packet from the connection initiator is flagged as a SYN message and the consequent reply from the responder is flagged as a SYN acknowledgement. The third messages is the first one that can carry actual data.

1.	$C \longrightarrow S$	ISN_C, SYN
2.	$S \longrightarrow C$	$ISN_S, ISN_{C+1}, SYN\|ACK$
3.	$C \longrightarrow S$	$ISN_{C+1}, ISN_{S+1}, ACK$

After receiving the first message, the responder creates a state called Transmission Control Block (TCB). In the SYN flooding attack, the adversary sends large numbers Message 1s to the server. The attacker also forges its IP address addresses on the underlying IP protocol level. The attack fills up the responder's table for saving TCBs, making it impossible to open further connections.

Several people have independently suggested versions of this protocol where the server does not create a state initially. Instead, the responder can compute a message authentication code of the initiator sequence number and other session parameters. It sends the MAC to the client and receives it again in the next message. The secret key K_S for signing should be changed periodically as described in Sec. 3.2. To maintain backward compatibility, the MAC is used in place of the initial server sequence number ISN_S.

1.	$C \longrightarrow S$	ISN_C, SYN
2.	$S \longrightarrow C$	$MAC_{K_S}(ISN_C, parameters), ISN_{C+1}, SYN\|ACK$
3.	$C \longrightarrow S$	$ISN_{C+1}, MAC_{K_S}(parameters) + 1, ACK$

After these changes, the server avoids creating the TCB before it knows the initiator's IP address to be valid. Unfortunately, this does not protect against attackers who are willing to reveal their IP addresses, or ones who can alter routing. They only need to proceed to the next protocol step where the responder waits for data from the initiator then leave the connection open. Another possibility for the attacker is to leave the TCP connection in half closed state: when the other party closes the connection, it may fail to send the final acknowledgement, thus leaving the other party waiting indefinitely. Early experiments by the present authors indicate that these problems could be solved by making the TCP responder completely stateless.

6.3 Secure HTTP cookies

HTTP cookies [7] are an enhancement to the Hypertext Transfer Protocol [1]. The cookies are packages of connection state information that are saved by the client and returned to the server in a way closely resembling our basic transformation in Sec. 3. The cookie protocol does not originally provide any security protection for the cookie information, which can be altered by the clients and by outsiders. The same techniques that we have used for ensuring integrity and confidentiality of the state data can be applied to HTTP cookies. The present authors have implemented a secure cookie system according to the principles presented in this paper. It appears that signing and encrypting the cookies causes no significant degradation of WWW server performance. Since the client cannot tamper with the secure cookies, new kinds of applications become possible. These include personal price offers on the electronic markets and stateless score-keeping for interactive games.

7 Conclusion

It is a characteristic weakness of stateful servers that they always have a limit on the number of clients that can be connected simultaneously. The behavior of such servers is unideal, because the limit can be exhausted by high demand of the service or by malicious attackers that leave connections in the open state.

In order to remedy the problem, we presented a transformation of stateful protocols into stateless ones. In a two-party protocol, it is usually practicable to make the server, or the responder, stateless. The stateless server does not save any state information in its own memory, but instead, sends it to the client for storage. The client returns the state data in its next request. This way, clients are able to maintain connections with a stateless server.

Integrity and confidentiality of the state data can be protected with secret key cryptography. The state data packets include a message authentication code that is computed with a secret key known only by the server. Session keys and other confidential state information are likewise encrypted with server's secret key. The connection data can also be protected and be bound to the correct state data packets.

The new stateless protocols are more robust against denial-of-service threats than their stateful counterparts. The stateless server responds to increasing load with slowly decreasing service quality rather than with the unexpected drop typical to stateless services. Moreover, leaving connections open does not disturb a stateless service in any way. The cost of the robustness is, of course, the communications bandwidth required for transferring the state data to the client and back. Also, fully stateless servers cannot dynamically optimize their communication. These performance penalties can be minimized with state caching and client-run packet windowing.

Statelessness does open one new line of attack: the stateless principals cannot detect replayed messages. Nevertheless, most protocols are deterministic enough so that replays cannot not significantly disturb their operation. The attacker may also try to flood the server with replays. This can exhaust the service capacity, but the connections of the real clients survive suck attacks. This is in sharp contrast with stateless protocols that tend to forget legitimate connections under denial-of-service attacks. Additionally, the number of potential attackers for the stateless protocols is much smaller, because on open networks, anyone can open useless connections to any server while few people are able to record and replay messages to a particular server.

To summarize, stateless protocols behave more ideally under denial-of-service attacks than their stateful counterparts, they recover faster from the attacks, and they effectively limit the number of potential attackers.

In protocols where the clients are reliably identified, statelessness is needed at the opening steps before the authentication. This can be seen, for example, in the X.509 and ISAKMP/Oakley key exchange protocols. Usually, the removal of states is most beneficial in transport and application layer protocols.

In the future, the presented techniques should be applied to new practical protocols. The ultimate goal would be a completely stateless protocol stack. On the theoretical side, the security of the stateless protocols should be examined from two directions. First, we expect statelessness greatly simplify formal analysis of the protocols, because the state space of stateless server protocols is small. Second, the preservation of security properties in the transformation should be assessed more carefully.

References

[1] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol – HTTP/1.0. RFC 1995, IETF Network Working Group, May 1996.

[2] *Recommendation X.509, The Directory - Authentication Framework*, volume VIII of *CCITT Blue Book*, pages 48–81. CCITT, 1988.

[3] TCP SYN flooding and IP spoofing attack. CERT Advisory CA-96.21, CERT, November 1996.

[4] UDP port denial-of-service attack. CERT Advisory CA-96.01, CERT, August 1996.

[5] D. Harkins and D. Carrel. The resolution of ISAKMP with Oakley. Internet draft, IETF IPSEC Working Group, June 1996.

[6] Recommendation x.509 (11/93) - the directory: Authentication framework. ITU, November 1993.

[7] David M. Kristol and Lou Montulli. HTTP state management mechanism. Internet draft, IETF HTTP Working group, July 1996.

[8] Howard F. Lipson and Thomas A. Longstaff. Coming attractions in survivable systems. Draft technical report, Software Engineering Institute, Carnegie Mellon University, June 1996.

[9] Douglas Maughan, Mark Schertler, Mark Schneider, and Jeff Turner. Internet security association and key management protocol (ISAKMP). Internet draft, IETF IPSEC Working Group, June 1996.

[10] Louis Perrochon. *Gateways in globalen Informationssystemen.* PhD thesis, ETH Zürich, 1996. Diss. ETH Nr. 11708.