A Formal Framework and Evaluation Method for Network Denial of Service

Catherine Meadows
Code 5543
Naval Research Laboratory
Washington, DC 20375
meadows@itd.nrl.navy.mil

Abstract

Denial of service is becoming a growing concern. As our systems communicate more and more with others that we know less and less, they become increasingly vulnerable to hostile intruders who may take advantage of the very protocols intended for the establishment and authentication of communication to tie up our resources and disable our servers. Since these attacks occur before parties are authenticated to each other, we cannot rely upon enforcement of the appropriate access control policy to protect us (as is recommended in the classic work of Gligor and Millen in [5, 18, 19]). Instead we must build our defenses, as much as possible, into the protocols themselves. This paper shows how some principles that have already been used to make protocols more resistant to denial of service can be formalized, and indicates the ways in which existing cryptographic protocol analysis tools could be modified to operate within this formal framework.

1 Introduction

Denial of service is becoming a growing concern. As our systems communicate more and more with others that we know less and less, they become increasingly vulnerable to hostile intruders who may take advantage of the very protocols intended for the establishment and authentication of communication to tie up our resources and disable our servers. Since these attacks occur before parties are authenticated to each other, we cannot rely upon enforcement of the appropriate access control policy to protect us (as is recommended in the classic work of Gligor and Millen in [5, 18, 19]). Instead we must build our defenses, as much as possible, into the protocols themselves.

The SYN attack on TCP (see [24] for a more complete discussion) is a classic example of this type of attack. In TCP, a source host initiates the protocol by sending a SYN (synchronization/start) message to its destination. The destination host responds with a SYN ACK, and waits for am ACK of the SYN ACK. When it receives it, the connection is established. The destination host keeps track of unacknowledged SYN ACKs in a connection queue. When a SYN ACK is acknowledged, it is removed from the queue.

The SYN attack is simple. The attacker sends SYN messages with fake source addresses to its victim, which then sends back SYN ACK messages which are never acknowledged. As a result, the victim's connection queue fills up, and TCP services are denied to legitimate users.

Although the SYN attack makes use of the features of a particular protocol, it is easy to see how it could be generalized. An attacker initiates an authentication protocol with its intended victim, using a fake identity. At some point in the protocol, the attacker simply stops participating. The victim is left with an unmanageably large number of uncompleted protocols on its hands, and shuts down operations. The victim cannot defend against this attack by refusing connections from the attacker, since the attacker uses a fake identity, and sometimes a number of different fake identities.

Attacks like this can be foiled by the use of good authentication protocols. But two things need to be kept in mind. First of all, such a protocol must provide authentication from the very beginning. It is not enough that the destination host should be able to verify the origin of the SYN ACK ACK message. It must be able to verify the origin of the SYN message as well. This is in contrast to such definitions of protocol security as are found in [3], in which correctness is defined in terms of the guarantees that must hold if the entire pro-

tocol completes. Thus the requirements for protection against denial of service appear to be much stronger than usual authentication requirements.

On the other hand, strong authentication itself can be a hook for denial of service attacks, since the mechanisms used to provide it are computationally intense. For example, suppose that the SYN message was protected by a digital signature. An attacker could send a number of bogus SYN messages with incorrect signatures, and the victim would use up its resources in verifying the signatures instead of in storing the SYN messages in the connection queue. The results, however, would be similar.

The approach that has been taken to resolve these conflicting requirements has been to use weak authentication when the protocol is initiated, but stronger authentication as it completes. Thus the protocol provides protection against a weak attacker in its initial stages without leaving itself vulnerable to denial of service attacks that take advantage of strong authentication, while still ultimately protecting the protocol against spoofing by a strong attacker. This does not leave the protocol completely invulnerable against a denial of service attack by a strong opponent, but it means that that opponent will have to work much harder, having first to break the weak authentication mechanisms.

A classic example of this approach is offered by Photuris [10]. A Photuris exchange begins with an interchange of cookies. A cookie is a unique nonce computed from the names of the sending and receiving parties and local secret information available only to the sender. Thus only the sender can origniate a cookie that it will accept, and only the originator of a cookie can verify that is genuine. Also, the cookie generation and verification methods are required to be fast, to make them less vulnerable to denial of service attacks. After initiator and responder exchange cookies, both cookies are included in all further messages. When either party receives a message, it first verifies the cookie. With this it gets the assurance that the message must have been sent after the cookie was generated, and by someone who was able to see the cookie. Thus the receiver knows that, if no intruder with the capability to generate new messages out of recent messages is present, then the message is genuine. Once this level of assurance is attained, the receiver performs the rest of the verification to achieve a higher degree of assurance.

Cookies provide an add-on that can be used to make an existing protocol more resistant to denial of service. But it is also possible to use these same principals as the basis for the design of an entire protocol. This is what is done by Kent et al. in [11], in which a protocol is developed that uses signed Lamport hash chains [13] to play the role of cookies in providing the weak authentication, while digital signatures are used to provide the strong authentication. Since the hash chains are precomputed and pre-authenticated, a member of the chain can be inserted in a signed message without having to include an initial cookie exchange in the protocol. Thus the participants gain the weak assurance without having to include a pair of cookie exchange messages in each instance of the protocol. The strong assurance can then be gained within the same message by verifying the digital signature. A separate protocol is needed to set up the hash chains, but this needs to executed much less often.

As we can see, the assurance offered by these protocols can increase in two ways. First of all, it can increase monotonically as each message is sent. In the cookie exchange, the receiver of a cookie has no way of verifying that the cookie it receives is genuine, so no assurance is offered at this stage. As the remaining messages are received and verified, however, the amount of assurance increases. The other way in which the assurance can increase is as a message is verified, as in [11], in which all messages are both signed and include a chained hash. When a party receives a message, it can verify the chained hash to determine its freshness; then it can verify the digital signature to determine that the message is not only recent but authentic.

The level of difficulty of the protocol execution also increases with the level of assurance. The cookie exchanges in Photuris proceed with a very low degree of assurance; however, cookie generation is intended to be cheap, so the risk to denial of service is lessened. But the protection against denial of service provided by including cookies in the later messages is somewhat higher, and it gives the receivers of the messages the confidence necessary to proceed to the next, more expensive stage of the authentication. Likewise for the protocol in [11], verification of chained hashes is cheap compared to signature verification.

We see that the type of assurance we must attempt to gain here is somewhat different from that the type of results that we attempt to prove to show that a protocol satisfies its authentication goals. For an authentication protocol, we generally only need to prove that its requirements are satisfied when the protocol completes. Moreover, we attempt to prove that the protocol is sound against the attacks of a uniformly strong intruder. But in order to prove that a protocol is secure against denial of service attacks, we must show that it satisfies the necessary properties at each step of the way. Moreover, the assumptions about the intruder's strengths that we consider may vary as the

protocol proceeds.

Thus the process of assuring that a protocol is secure against denial of service is potentially much more complex than the problem of assuring that a protocol satisfies its authentication requirements. However, the underlying process, showing that certain authentication goals are achieved in the face of attack by an intruder who possesses certain capabilities, is similar to that used to show that a protocol satisfies its authentication requirements. Thus it should be possible to apply many of the tools and techniques that have been developed for the verification of authentication properties to the analysis of denial of service. In this paper we provide a framework for evaluating a protocol for resistance to denial of service attacks in a way that is intended to allow us to make maximal use of these tools.

The remainder of the paper is organized as follows. In Section 2 we motivate and present the framework. In Section 3 we apply the framework to an example. In Section 4 we discuss the ways in which existing security protocol analysis tools and methods could be modified to verify protocols within this framework.

2 The Framework

Our construction of the framework begins with the observation that any point at which during a protocol execution at which a responder may accept a bogus message as genuine could be used to launch a denial of service attack. If a bogus initial message could be accepted as genuine, an attack such as the SYN attack would be possible, in which the target wastes its resources storing and responding to bogus messages. If a bogus intermediate message is accepted, this could cause the target to waste even more of its resources to get to that point in the protocol. Finally, if a bogus final message is accepted as genuine, this could give rise to a particularly insidious denial of service attack in which a principal believes that it has been provided a service (e.g. a key shared with another party) when in fact this service was not provided. We will refer to these types of attacks as *service spoofing* attacks. Similar concerns exist for the case of the initiator, although these are not as strong here, since a denial of service attack would in most cases require the initiator to initiate a large number of protocols, which the attacker then subverts. This is not impossible, but it is not certainly not as likely as the attacker initiating a large number of protocols.

The fact that a certain degree of guarantee must be achieved at each step of the way naturally leads us to Gong and Syverson's concept of *fail-stop* protocols [7]. Briefly, a protocol is fail-stop if any bogus message (that is, a replay or a message manufactured by the intruder) can be detected, and the protocol halts upon detection.

Fail-stop protocols have some of the desirable features of a denial-of-service-resistant protocol. However, in order to achieve the fail-stop property, they must make use of strong authentication right from the beginning. This makes fail-stop protocols potentially vulnerable to denial of service attacks in which the target is forced to use up resources verifying bogus messages. Thus we need to modify our concept of fail-stop in order to make it applicable to our needs. However, we will do so in the manner we suggested earlier in this paper, by modifying our notion of an intruder's capability.

2.1 Alice-and-Bob Specifications and Causal Sequencing

In this paper we will be making use of the popular "Alice-and-Bob" specification style. This has been criticized as confusing the description of what should happen with what actually does happen [6]. But in this case, a description of what does happen which can be made to correspond to a description of what should happen is exactly what we want, so much so that we are led to a formal definition of what we will call annotated Alice-and-Bob specifications.

Definition 1 *An* Alice-and-Bob *specification is a sequence of statements of the form* $A \rightarrow B : M$ *where* A *and* B *are processes and* M *is a message.*

Since we are interested in how messages are processed, as well as what messages are sent, we need to annotate our Alice-and-Bob specifications to include message processing steps.

Definition 2 *An* annotated *Alice-and-Bob specification is a sequence of statements of the form* $A \rightarrow B : T_1, ..., T_k \parallel M \parallel O_1, ..., O_n$ *where* A *and* B *are processes,* M *is a message, and* $T_1, ..., T_k$ *and* O_1, ..., O_n *are sequences of operations performed by* A *and* B*, respectively.*

The sequence $T_1, ..., T_k$ preceding the message in an annotated Alice-and-Bob specification represents the sequence of operations performed by A in producing M, while the sequence $O_1, ..., O_n$ represents the sequence of operations performed by B in processing and verifying M. We now look at the make-up of a line in an Alice-and-Bob protocol more closely.

Definition 3 *Let* $\mathbf{L} = A \rightarrow B : T_1, ..., T_k \parallel M \parallel O_1, ..., O_n$ *be a line in an annotated Alice-and-Bob specification. We say that* X *is an* event occurring in **L** *if*

1. X *is one of the* T_i *or* O_j*, or;*
2. X *is 'A sends* M *to B' or 'B receives* M *from A'.*

We say that the events $T_1, ..., T_k$ *and 'A sends* M *to B'* occur at A *and the events 'B receives* M *from A',* $O_1, ..., O_n$ occur at B. *There are two types of events:* normal events *and* verification events *(also called* verification operations*). Normal events can occur at either sender or receiver, and have only one outcome : success. Verification events occur only at the receiver. A verification operation can have two outcomes, success or failure. If the operation succeeds, then B engages in the next event in* **L***; if the operation fails, B stops does not engage in the next event, and stops participating in the protocol. We also attach at the end of each line a special event called an* accept event *which describes B's deciding to proceed with the protocol after successfully verifying a message.*

For example, consider the case in which A computes a digital signature over B's name and a nonce, and sends the result to B, together with its own name and B's. The resulting annotated specification would look like this:

$A \rightarrow B$:
$computenonce_1$, $storenonce_1$, $storename_1$, $sign_1 \parallel A, B, S_A(B, N_A)) \parallel$
$checkname_2$, $storenonce_2$, $storename_2$, $checksig_1$, $accept_1$

where *computenonce* denotes A's computing a nonce, *storenonce* and *storename* denote A (respectively, B)s storing its nonce and B's (respectively, A's), name for future reference, *sign* denotes A's computing a digital signature, *checkname* denotes the checking for the presence of B's name, and *checksig* denotes the checking of A's digital signature.

We now want to use our annotated Alice-and-Bob specifications to help us give our requirements for a cryptographic protocol. For this, as in [7], we use a notion very similar to Lamport's notion of causally-precedes [12]. The only difference between our notion and Lamport's is that Lamport was dealing with a situation in which, although messages could be delayed or lost, they were not spoofed, redirected, or altered by a hostile intruder. Thus, Lamport used the notion of causally-before to describe what actually happened in his environment, while we will merely use it to describe what we would like to happen in our environment.

For motivation, we first give Lamport's definition:

Definition 4 *Let* S *be a system of distributed processes that communicate by sending messages. Let* E *be a temporally ordered sequence of events in* S*, where* E *consists of internal events, sending of messages, and receiving of messages. If* a *and* b *are two events from* E*, then* b *is causally-after* a *if:*

1. a *and* b *occur at the same process, and* a *precedes* b*;*
2. a *is the sending of a message by one process, and* b *is the receiving of the same message by another process, or;*
3. *there is an event* c *such that* c *is causally-after* a *and* b *is causally-after* c*.*

We say that E_1 is *causally-after* E_2 if E_2 causally-precedes E_1.

We note that notions very similar to Lamport's have found fruitful application in the analysis of cryptographic protocols, most notably in the definition of strand spaces [4]. However, for our purposes, we will need something a little different. An Alice-and-Bob specification of a cryptographic protocol can be thought of as giving of a requirements specification in terms of what events *should* causally-precede others. We thus define *desirably-precedes* (or *desirably-after*), as follows:

Definition 5
1. *If* $A \rightarrow B : R_1, ..., R_m \parallel M \parallel O_1, ..., O_n$ *appears then the event in which* B *receives* M *desirably-precedes any of the* O_i*, and* O_i *desirably-precedes any of the* O_j *for which* $i < j$*;*
2. *If* $A \rightarrow B : R_1, ..., R_m \parallel M \parallel O_1, ..., O_n$ *appears then any* R_i *desirably-precedes the event in which* A *sends* M *and* R_i *desirably-precedes any of the* R_j *whenever* $i < j$*;*
3. *If* $A \rightarrow B : R_1, ..., R_m \parallel M \parallel O_1, ..., O_n$ *precedes* $B \rightarrow Y : S_1, ..., S_p \parallel N \parallel T_1, ..., T_k$ *then* O_n *desirably-precedes* S_1*;*
4. *If* $A \rightarrow B : R_1, ..., R_m \parallel M \parallel O_1, ..., O_n$ *appears then the event in which* A *sends* M *to* B *desirably-precedes the event in which* B *receives* M *from* A*, and;*
5. *If* $\mathbf{E_1}$ *desirably-precedes* $\mathbf{E_2}$ *and* $\mathbf{E_2}$ *desirably-precedes* $\mathbf{E_3}$ *then* $\mathbf{E_1}$ *desirably-precedes* $\mathbf{E_3}$*.*

Note that our definition of desirably-precedes assumes that a principal will not start creating a message it wants to send until it has finished processing all relevant messages it has received. This assumption is

reasonable when we are attempting to avoid denial of service attacks : a principal should not risk wasting resources on creating a message until it has verified that the messages it has received are genuine.

2.2 Fail-Stop Protocols

A fail-stop protocol is one that provides a certain degree of security against attack. Thus, in order to define a fail-stop protocol, we need first to say what an attack is. But an attack describes a behavior of the implemented protocol that deviates from its requirements. An Alice-and-Bob specification describes the desired behavior very well; what we need to supply as well the derivation of an implementation of a protocol from an Alice-and-Bob specification as follows. This will allow us to make clear the difference between actual and required behavior.

Definition 6 *An implementation of an Alice-and-Bob specification is a set of programs, one for each participant in the protocol. We say that the program corresponding to* A implements A *and refer to it at* Pr(A)*. Multiple copies of this program can exist and can be running on behalf of different principals.* **Pr**(A) *has the following properties:*

1. *For each instance of* $A \rightarrow B : R_1, ..., R_m \parallel M \parallel O_1, ..., O_n$ *in the protocol there are corresponding portion of* **Pr**(A) *describing A performing the operations* $R_1, ..., R_m$ *and sending the resulting message to B. When we can avoid confusion, we will refer to these as* **Pr**(A)*[*$A \rightarrow B : M$*]* *and* **Pr**(A)*[*R_i*].*
2. *For each instance of* $C \rightarrow A : R_1, ..., R_m \parallel M \parallel O_1, ..., O_n$*, there are corresponding portions of* **Pr**(A) *describing A receiving a message. performing the operations* $O_1, ..., O_n$ *on a received message, and halting if any of the verification operations fail. When we can avoid confusion, we will refer to these as* **Pr**(A)*[*$B \rightarrow A : M$*]* *and* **Pr**(A)*[*O_i*].*
3. *If both* E_1 *and* E_2 *occur at A, and* E_1 *desirably-precedes* E_2*, then* **Pr**(A)*[*E_1*]* *always executes before* **Pr**(A)*[*E_2*].*

We do not go any further into describing a procedure for deriving implementations from annotated Alice-and-Bob specifications, but we note that a number of tools, such as Casper [16] and the CAPSL-to-NRL translator [1] already exist that provide such a functionality by translating high-level Alice-and-Bob-style specifications into process algebra and state machine specifications.

Definition 7 *We say that the* event B receives M from A *in a cryptographic protocol has occurred if the program fragment* **Pr**(B)*[*$A \rightarrow B : M$*]* *has executed successfully. We say that M has been* interfered with *if the event B receives M from A occurs but some event desirably-preceding it did not.*

Note that the fact that B receives M from A occurs does not mean the B actually received M from A or anyone else. It simply means that B received a message at the point at which it was expecting the message M, and it is ready to perform a set of tests to determine whether the message was genuinely M. Thus although we use the term "interfered with" to be consistent with the language in [7], we note that it covers not only messages that were tampered with, but fake messages generated by an intruder.

We are now ready for the definition of fail-stop from [7], modified slightly to take into account our slightly different definition of an event.

Definition 8 *An Alice-and-Bob specification of a cryptographic protocol is* fail-stop *if, whenever a message is interfered with, then no accept event desirably-after the receiving of that message will occur.*

We have as yet said nothing yet about our assumptions about an intruder's capabilities. In most of the literature on cryptographic protocol analysis, the assumptions that are made about what an intruder can do are the same. An intruder is assumed to be able to read all traffic, delete and modify traffic, have access to system operations such as the encryption functions used, to be in collusion with some legitimate users of the system, and possibly have access to compromised session keys and other secrets from previous executions of the protocol. In our case, of course, we are interested in intruders with different capabilities:

Definition 9 *We define an* intruder action *to be an event engaged in by an intruder that affects messages received by legitimate participants in a protocol. We define an* intruder capability *to be a set of actions available to an intruder, partially ordered by set inclusion.*

We do not list the types of capabilities available to the intruder, since at this point we do not want to limit ourselves. However, examples would include such cases as an intruder who could send messages but not read messages that were not addressed to it, an intruder who could send and read messages but not block messages, an intruder who could send, read, and block messages, but could not replace the blocked messages with other messages in real time, and so forth. We might

also want to include actions outside the usual intruder model, such as the ability to break certain classes of cryptosystems.

Definition 10 *Let , be a function from the set of events defined by an annotated Alice-and-Bob specification P to a set of intruder capabilities. We refer to , as an* intruder capability function. *We say that P is* fail-stop with respect to , *if, for each event E in the system, if an intruder of capability , (E) interferes with any message desirably-causally-preceding E, then neither E nor any events desirably-after E will occur.*

The idea behind an intruder capability function is that, for each accept event E, an intruder of capability , (E) should not be able to cause E to occur by using its capabilities to manipulate the system.

If we set , (E) to be the usual intruder capability whenever E is an accept event, and the null capability for all other events, then the reader can verify that our definition of fail-stop is equivalent to Gong and Syverson's. In the case of denial of service, though, we may want , to be gradually increasing with respect to final verification events. For example, in the Photuris protocol, the intruder capability associated with the first accept events, which only concern the cookie exchange, would be the ability to read and block messages, but nothing else, while the intruder capability associated with the later events is greater. We would also want , to be gradually increasing with respect to any sequence of verification events on a single line of an annotated Alice-and-Bob specification, since the amount of assurance gained after each check done on a single message should increase the confidence in that message.

Intruder capability functions tell us only half the story, however. We also need a way of calculating the expense of executing each protocol step. In order to do this, we need to define a cost function.

Definition 11 *A* cost set **C** *is a partially ordered set with partial order $<$ together with a function $+$ from* **C** $\times$ **C** *to* **C** *such that $+$ is associative and commutative, and $x + y \geq max(x,y)$, along with an zero element 0 such that $x = 0 + x = x + 0$, for all x in* **C**.

Examples of cost could be things like time or money, in which case $+$ is simple addition. Or it could be a rough estimate such as a division into "easy" or "hard," in which case $x + y$ is simply the maximum of x and y. In general, we would expect the cost of a verification event to express the expense of performing the verification, the cost of sending a message to express the expense involved in preparing that message, and the cost of accepting a message (that is, the cost of a final accept event) to be the cost of performing all the verification steps on that message, as well as the cost of sending the next message, if any.

Definition 12 *A function δ from the set of events defined by an annotated Alice-and-Bob specification P to a cost set C which is 0 on the accept events is called an* event cost function.

We now define two functions based on cost functions. One describes the cost of processing a single message in a protocol. The other describes the cost of a principal's reaching a given point in the protocol.

Definition 13 *Let P be an annotated Alice-and-Bob protocol, let C be a cost set, and let δ be an event cost function defined on P and C. We define the* message acceptance cost function associated with δ *to be the function δ' on events following the receipt of a message as follows:*

If the line $A \rightarrow B : O_1, ...O_k \parallel M \parallel V_1, ..., V_n$ appears in P, then for each event V_j:

$\delta'(V_j) = \delta(V_1) + ... + \delta(V_j)$.

We define the protocol engagement cost function associated with δ *to be the function Δ defined on accept events as follows:*

If the line $A \rightarrow B : O_1, ...O_k \parallel M \parallel V_1, ..., V_n$ appears in the protocol, where V_n is the accept event, then:

1. *If there are no lines $B \rightarrow X : O'_1, ...O'_k \parallel M' \parallel V'_1, ..., V'_n$ such that V_n immediately desirably-precedes O'_1, then $\Delta(V_n)$ is the sum of all the costs of all operations occurring at B desirably-preceding V_n;*

2. *If there is a line $B \rightarrow X : O'_1, ...O'_k \parallel M' \parallel V'_1, ..., V'_n$ such that V_n immediately desirably-precedes O'_1, then $\Delta(V_n)$ is the sum of the costs of all operations occurring at B desirably-preceding V_n, plus the sum of the costs of the O'_i,.*

Thus the message acceptance cost represents the cost of reaching a certain point in accepting a message, while the protocol engagement cost represents the total cost of accepting a message in terms of both the cost of processing it and the costs of any events that are necessarily engaged in as a result of accepting the message.

Our construction of message acceptance and protocol engagement costs reflects two ways in which denial of service attacks can proceed. One is by sending a principal a bogus message at any point in the protocol, and having the victim waste resources processing

it. This is protected against by performing the relatively cheap verification events first upon receiving a message, and should be reflected in the message acceptance costs. In particular, we would like the costs to start out cheap, and only become more expensive towards the end of the message. The other is to persuade a principal to waste resources participating in a bogus instance of the protocol; this can be defended against by performing cheap verification and computation events early in the protocol, and should be reflected in the protocol engagement costs.

We are now ready to relate cost and intruder capability functions.

Definition 14 *Let C be a cost set and G an intruder capability set. We define a* tolerance relation defined by a protocol designer *to be the subset of $C \times G$ consisting of all pairs (c,g) such that the protocol designer is willing to tolerate a situation in which an effort of cost c that provides security against an intruder of capability g but no greater. We say that (c',g'), is* within the tolerance relation *if there is a (c,g) in the relation such that $c' \leq c$ and $g' \geq g$.*

We do not put any constraints on the tolerance relation, since we believe it may vary from situation to situation.

We can now describe the procedure for evaluating whether or not a protocol is secure against denial of service using the following steps:

1. Decide what your cost function is, and what you assume the various capabilities of the intruder can be.

2. Decide what your tolerance relation is: how much are you willing to spend to provide a certain level of security, and how much insecurity are you willing to put up with given a certain amount of cost?

3. Calculate an intruder capability function , such that:

   a) If E_1 is an event immediately preceding a verification event E_2, in a line of a protocol, then $(\delta'(E_2),, (E_1))$ is in the tolerance relation. This means, that, if M is the message received in the line in which E_1 and E_2 appear, the cost of getting to the point where E_2 succeeds or fails is $\delta'(E_2)$, and no intruder of capability , (E_1) will have been able to successfully interfere with M after E_1 has finished executing.

   b) If E is an accept event, then $(\Delta(E),, (E))$ is in the tolerance relation.

4. Verify that the protocol is fail-stop with respect to , .

Note that Step 3a) allows us to reason about the ability of a protocol to prevent an intruder from mounting a denial-of-service attack by causing a legitimate principal to waste resources processing a message before it has been able to verify that it could not have been spoofed by an intruder of a certain capability. Step 3b) allows us to reason about the ability of a protocol to prevent an intruder from mounting a denial-of-service attack by causing a legitimate principal to waste resources participating in a protocol up to receiving a particular message and responding to it, before it has been able to verify that that message could not have been spoofed by an intruder of a certain capability.

3 Example: The Station-to-Station Protocol

In this section we show how we can apply our framework to the Station to Station protocol of Diffie, van Oorschot, and Wiener. This protocol was designed with message economy rather than denial of service in mind, so, not surprisingly, it turns out not to meet the demands of our framework. However, the framework can help us identify areas of weakness and suggest possible improvements from the denial of service perspective.

The protocol uses Diffie-Hellman for key generation and digital signatures for authentication. It involves a number of different operations, to which we assign costs as follows: exponentiation over a finite field during the protocol execution (expensive), represented by *exp*, exponentiation over a finite field which can be precomputed (medium), represented by *preexp*, computation and verification of digital signatures (expensive), represented by *sign* and *checksig*, single key encryption and decryption (medium), represented by *encrypt* and *decrypt*, checking for the presence of names and retrieving nonces (cheap), represented by *checkname* and *retrievenonce*, and storing of names and nonces (medium), represented by *storename* and *storenonce*. We define a + operation on costs by $X+Y = \max(X,Y)$, where expensive > medium > cheap > 0.

Next, let $\mathbf{Z_P}$ be the integers modulo P for some P, let α be a generator of the multiplicative group of $\mathbf{Z_P}$, let α^x denote α raised to the x'th power mod P, let $K = \alpha^{X_B \cdot X_A} = \alpha^{X_A \cdot X_B}$, let E_K represent single key encryption with key K, and S_B denote a digital signature obtained using B's private key. We can now use our notation to describe the protocol as follows:

1. $A \rightarrow B$: $preexp_1$ ||
   α^{X_A} ||
   $storenonce_1, storename_1, accept_1$.

2. $B \rightarrow A$:
   $preexp_1, sign_1, exp_1, encrypt_1$ ||
   α^{X_B}, $E_K(S_B(\alpha^{X_B}, \alpha^{X_A}))$ ||
   $checkname_1$, $retrievenonce_1$, exp_2, $decrypt_1$,
   $checksig_1$, $accept_2$.

3. $A \rightarrow B$:
   $retrievenonce_2, sign_2$, $encrypt_2$ ||
   $E_K(S_A(\alpha^{X_A}, \alpha^{X_B}))$ ||
   $checkname_2$, $retrievenonce_2$, $decrypt_2$,
   $checksig_2$, $accept_4$.

Since we are dealing with a protocol that has already been developed, rather than designing one from scratch, we will proceed in an order somewhat different than that recommended in the last section. First, we will determine the values of the cost functions. Secondly, we determine the intruder capability function with respect to which the protocol is fail-stop. We will use this to determine a likely set of tolerance relations, and discuss their significance to the protocol.

We first note that protocol engagement cost function is *expensive* for all accept events. For the first, the responder B must compute the Diffie-Hellman key and sign a message. For the second, the initiator A must check a signature, compute a Diffie-Hellman key and sign a message. For the third, B must check a signature. For the message acceptance cost functions, none is defined for the first message, since it contains no verification events. For the second the cost of B's checking the name is cheap, while the cost of B's performing the signature is expensive. Likewise for the third message: the cost of A's checking a name is cheap, while the cost of its checking a signature is expensive.

How does the intruder capability function fare for the accept events? Clearly, since the first message is not authenticated at all, it protects only against a very weak intruder, so best we can take , ($accept_1$) to be is the intruder with the powers of a non-malicious network, or an intruder with the capability of sending a message. Any intruder who could create an acceptable-looking return address should be able to defeat the protocol at this point. We can take , ($accept_3$) to the intruder with full powers (we have verified this using the NRL Protocol Analyzer). What is surprising, however, is , ($accept_2$). We would expect this to also be an intruder with full powers, but as a matter of fact it is somewhat weaker. As was shown by Lowe in [15], the event $accept_2$ is vulnerable against the following attack, where I is the intruder, and I_Z is the intruder impersonating Z:

1. $A \rightarrow I_B$: α^{X_A}
2. $I_C \rightarrow B$: α^{X_A}
3. $B \rightarrow I_C$: α^{X_B}, $E_K(S_B(\alpha^{X_B}, \alpha^{X_A}))$
4. $I_B \rightarrow A$: α^{X_B}, $E_K(S_B(\alpha^{X_B}, \alpha^{X_A}))$
5. $A \rightarrow I_B$: $K(S_A(\alpha^{X_A}, \alpha^{X_B}))$

At this point A is convinced that it is sharing a key with B, although B knows nothing about this; if B received A's final message it would reject it, since it is expecting a response from C. Note that all this attack requires is an intruder with the ability to intercept messages intended for B, and to forward them to B as messages from C. It does not need to be capable of performing any cryptographic operations such as exponentiation or the checking of a digital signature.

Since this is an attack against the initiator, not the responder, it is potentially less useful for tying up resources; the victim must decide to initiate an instance of the protocol before the attack can be mounted. (Note that a successful denial-of-service attack using this vulnerability would require the initiator to start the protocol not once, but many times.) On the other hand, it can be thought of as a service spoofing attack, since A thinks it shares a key with B, when in fact it does not.

We now discuss what the values of , for the verification events. The first verification event is $checkname_1$. Since no verification is done before the name is checked, the associated , is the intruder with the powers of a non-malicious network. However, since no event except the reception of the message precedes $checkname_1$ at that line, and the cost of $checkname_1$ itself is quite low, the pair (c,g) computed should be well within any tolerance relation. The next verification event is $checksig_1$. The message acceptance cost of $checksig_1$ is expensive; however , ($decrypt_1$), the immediately preceding event, is an intruder who is not able to forge B's return address and prevent messages from reaching their destinations, in other words a relatively weak intruder. This pair would probably not be within most tolerance relations. The next verification event is $checkname_2$. Again, the message acceptance cost of $checkname_2$ is cheap, while the assurance provided before $checkname_2$ is executed is against an intruder with the powers of a non-malicious network. Again, the associated pair (c,g) should be well within any tolerance relation. Finally, the last verification event is $checksig_2$. Again, the message acceptance cost of

$checksig_2$ is expensive, but the assurance provided until $checksig_2$ has been executed successfully is relatively weak: it is only strong against an intruder who cannot forge A's address and prevent messages from reaching their destinations.

Thus the Station-to-Station protocol, as it stands, is vulnerable to denial of service attacks in several places. In the first message, an intruder who is capable of doing nothing more than sending messages could send a bogus message and cause the responder to waste resources responding to it. Likewise, since the only cheap interim checks on the last two messages are weak, an intruder who is capable of faking return addresses could cause either initiator or responder to waste resources in processing a bogus message. Finally, though less seriously, a somewhat more capable intruder could mount Lowe's attack and convince an initiator that it is sharing a key with a responder when it does not, and when the responder is not even expecting a message from the initiator.

There are a number of ways in which this protocol could be strengthened against denial of service attacks. First, a cookie exchange could be done initially, to introduce some weak authentication before the major message exchange starts, as is done in the IKE protocol [8], which uses a protocol based on the station-to-station protocol. Secondly, if cookies are included in second and third messages, checking them could provide weak authentication for these as well (as is also done by IKE). Weak authentication could also be provided by including both α^{X_A} and α^{X_B} in the second and third messages, so that either party Y could check for the presence of α^{X_Y} before proceeding with the more expensive verification steps. Finally, Lowe's attack could be prevented by including the identity of the intended receiver in the signed part of the message, as is recommended in [15].

4 Applicability of Existing Tools and Models

In this section, we consider how some of the existing tools and methods could be applied within our framework.

We begin with belief logics. To look at them, we would not expect belief logics such as BAN [2] to be very useful within our framework. They use an implicit model of the intruder, and guarantee properties such as freshness and authentication, not immunity against intruders of various strengths. However, like our framework, BAN and similar logics are used to analyze protocols incrementally; one sees what degree of security is provided by each message as it is processed. And, although the properties guaranteed by these logics are currently cast in terms of beliefs in the properties of keys and messages, not properties of the intruder, there does not seem to be any inherent reason why they could not be recast as statements about what capabilities the intruder is supposed to have. For example, consider a message that contains a fresh sub-element, such as a cookie. That message is authentic if we assume the the intruder is not able to read and modify messages in real time. On the other hand, a signed message containing a fresh sub-element is authenticated in the face of a stronger intruder.

We next consider tools that make use of state exploration techniques in some form or the other. These include model checkers such as FDR/Casper [16] or Murϕ [22], specialized tools such as the Interrogator [20] that provide much of the same capability but are fine-tuned for cryptographic protocol analysis, and tools such as the NRL Protocol Analyzer [17] that combine state exploration with a limited theorem-proving capability. What all of these tools have in common is that at some point their designers implemented the standard intruder model as part of the tool. Thus, in order to make use of our framework, it would be necessary to have the option to replace the standard intruder with intruders of different strengths. This should not be too difficult, since all these tools model the intruder as an independent state machine; all that is needed is to replace this state machine with another. Similarly, most uses of theorem provers to analyze cryptographic protocols (e.g. Paulson [23]), include an independent specification of an intruder; thus it should be straightforward to deal with intruders of various strengths for theorem provers as well as state exploration tools.

Another issue remains to be considered, however. Usually, when we attempt to prove security of cryptographic protocols, we examine only a handful or properties, e.g., that two parties always agree on a key, that the key is not a replay, that the key remains secret, and so forth. In the case of denial of service, we are attempting to prove at least one goal for each accept event (and thus for each message sent), as well as one goal for each verification event. Thus the work involved could be multiplied several times. The model checkers have an advantage here; since model checkers verify whether or not a program satisfies its specification, all the user has to do is write a specification in terms of all the security goals that are relevant to a particular intruder strength, and run the protocol together with that intruder. Thus it may be possible to proceed so that the amount of work really only increases by a factor equal to the number of intruder strengths involved.

In the case of tools such as theorem provers and the

NRL Protocol Analyzer, in which goals are verified separately, the problem is a little more difficult. In this case it would probably be necessary to develop some standard lemmas involving different goals and intruders of different strengths to allow us to use the weaker results that we find to aid us in proving the stronger ones. This is an open area of research.

Finally, we consider the applicability of high-level protocol description languages, such as CAPSL [21] and Casper [16]. These languages are based on the popular Alice-and-Bob notation, so the most straightforward thing would appear to be to include the annotations that we have used in building our framework. But, as a matter of fact, this may not be necessary. Translators for these languages commonly infer the necessary operations directly from the specification; there is no reason that they should not also be able to derive a sequence of such operations that is optimal with respect to increasing cost, assuming that they are given the cost of each type of operation. Thus, all that would be needed to be added to the high-level specification would be an estimate of the cost of each type of operation; these could even be built into the translators when they are well understood.

5 Conclusion

We have developed a framework for reasoning about network denial of service, and indicated how existing tools and methods could be modified for reasoning within this framework. But there is still much work that remains to be done. Denial of service, unlike authentication, is a matter of degree. No protocol is completely immune against denial of service attacks. Moreover certain types of mechanisms, such as strong authentication, may make a protocol more resistant to one kind of attack but more vulnerable to other kinds. Things are made even more complicated by the fact that certain types of denial of service attacks may take advantage of other, previous, denial of service attacks. For example, an attack that requires an intruder to impersonate another principal may be easier to implement if that principal was disabled by a denial of service attack. This may include extending our model to cases in which intruder strengths may vary for different executions of a protocol. Thus, we need ways of measuring the degree to which a protocol is vulnerable to denial of service, and ways of using our measurements to reason about the different ways denial of service requirements can interact.

Another issue arises from the possible interaction of our framework with other classes of defenses against denial of service. For example, the SYN attack depends upon the necessary fact that principals must store state information when they participate in a protocol. There is no way to make this requirement go away, but it is possible to mitigate the bad effects by paying careful attention to such matters as how much state information is held, and for how long. We have not addressed this problem here, but others have begun applying formal techniques to its analysis [14]. It may turn out to be useful to pursue this matter further and see if our framework can be adapted and/or expanded to take into account countermeasures such as these. Another possibility for further investigation is the use of cryptographic "puzzles" to increase the difficulty of a clients' making any connection whether genuine or fake, thus making it more difficult for a malicious client to flood a server [9]. Here, the concept of increasing the client's work is still used, but the cryptographic functions are used directly to increase a client's workload instead of to provide authentication. It may be that, with some modification, our framework could be extended to handle this type of defense.

6 Acknowledgments

We would like to thank Stephen Brackin and the anonymous referees for their helpful comments on earlier versions of this paper. This work was supported by ONR.

References

[1] Stephen Brackin, Catherine Meadows, and Jonathan Millen. CAPSL interface for the NRL Protocol Analyze. In *Proceedings of ASSET'99*. IEEE Computer Society Press, March 1999.

[2] Michael Burrows, Martín Abadi, and Roger Needham. A Logic of Authentication. *ACM Transactions in Computer Systems*, 8(1):18–36, Feb. 1990.

[3] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, 1992.

[4] F. Javier Thayer Fabrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pages 160–171. IEEE Computer Society Press, May 1998.

[5] V. Gligor. A note on the denial-of-service problem. In *Proceedings of the 1983 Symposium on Security and Privacy*, pages 139–149. IEEE Computer Society Press, 1983.

[6] Dieter Gollmann. What do we mean by entity authentication? In *Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy*, pages 46–54. IEEE Computer Society Press, 1996.

[7] Li Gong and Paul Syverson. Fail-stop protocols: An approach to designing secure protocols. In R. K. Iyer, M. Morganti, Fuchs W. K, and V. Gligor, editors, *Dependable Computing for Critical Applications 5*, pages 79–100. IEEE Computer Society, 1998.

[8] D. Harkins and D. Carrel. The Internet Key Exchange (IKE), version 8. draft-ietf-ipsec-isakmp-oakley-08.txt, June 1998.

[9] Ari Juels and John Brainard. Client puzzles: A cryptographic countermeasure against connection depeletion attacks. In *Proceedings of the 1999 Network and Distributed System Security Symposium (NDSS'99)*. Internet Society, March 1999. Available at http://www.isoc.org/ndss99/-proceedings/.

[10] P. Karn and W. Simpson. The Photuris session key management protocol. Internet draft: draft-simpson-photuris-17.txt, November 1997.

[11] S. T. Kent, D. Ellis, P. Helinek, K. Sirois, and N. Yuan. Internet routing infrastructure security countermeasures. BBN Report 8173, BBN, January 1996.

[12] L. Lamport. Times, clocks, and the ordering of events in a distributed system. *CACM*, 21(7):558–565, July 1978.

[13] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, pages 770–772, November 1981.

[14] Patrick Lincoln. personal communication, Sept. 1998.

[15] Gavin Lowe. Some new attacks upon security protocols. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop*, pages 162–169. IEEE Computer Society, June 1996.

[16] Gavin Lowe. Casper, a compiler for the analysis of security protocols. In *Proceedings of 10th IEEE Computer Security Foundations Workshop*, pages 18–30. IEEE Computer Society Press, Jun 1997.

[17] Catherine Meadows. The NRL Protocol Analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[18] Jonathan Millen. A resource allocation model for denial of service. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, pages 137–147. IEEE Computer Society Press, 1992.

[19] Jonathan Millen. Denial of service : A perspective. In F. Cristian, G. Le Lann, and T. Lunt, editors, *Dependable Computing for Critical Applications 4*, pages 93–108. Springer-Verlag, 1995.

[20] Jonathan Millen. The Interrogator model. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, pages 251–260. IEEE Computer Society Press, May 1995.

[21] Jonathan K. Millen. CAPSL: Common Authentication Protocol Specification Language. Technical Report MP 97B48, The MITRE Corporation, 1997. See http://www.csl.sri.com/ millen/capsl.

[22] J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Murϕ. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 141–151. IEEE Computer Society Press, May 1997.

[23] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[24] C. Schuba, I. Krsul, M. Kuhn, G. Spafford, A. Sundaram, and D. Zamboni. Analysis of a denial of service attack on TCP. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 208–223. IEEE Computer Society Press, May 1997.