A Cost-Based Framework for Analysis of Denial of Service in Networks

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 computer systems communicate more and more with others that they 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 resources and disable servers. This paper shows how some principles that have already been used to make cryptographic protocols more resistant to denial of service by trading off the cost to defender against the cost to the attacker can be formalized based on a modification of the Gong-Syverson fail-stop model of cryptographic protocols, and indicates the ways in which existing cryptographic protocol analysis tools could be modified to operate within this formal framework. We also indicate how this framework could be extended to protocols that do not make use of strong authentication.

1 Introduction

Denial of service is becoming a growing concern. As computer systems communicate more and more with others that they 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, it is not possible to rely upon enforcement of the appropriate access control policy for protection (as is recommended in the classic work of Gligor and Millen in [7, 18, 19]). Instead the defenses, as much as possible, must be built into the protocols themselves.

One of the most common and devastating types of denial of service attack is the resource exhaustion attack, in which an attacker, by initiating a large number of instances of a protocol, causes a victim to waste resources. Although the victim can thwart this attack by refusing the communicate with the attacker once the source of its message is known, this defense can be circumvented, or at least made more difficult, by disguising the origin of the requests. For example, the SYN attack on TCP/IP is a classic example of this type of attack. The SYN attack works by having an attacker initiate a number of instances of the protocol but fail to complete them. The victim exhausts its resources maintaining state information until timeout. Since the verification of the origin of messages in TCP/IP is based on sequence numbers that are easily forged, the victim cannot easily identify the attacker even when it is aware it is under attack.

There are a number of defenses against attacks like this. One is to reduce the cost to the potential victim (from now on called the defender) of engaging in the protocol. Another is to increase the resources of the defender. A third is to introduce some sort of authentication so that a defender could at least tell where an attack is coming from.

However, using authentication introduces denial of service risks of its own. Suppose, for example, that a protocol requires the defender to verify messages signed with digital signatures. Verification of such signatures is relatively resource-consuming, and an attacker could introduce denial of service by launching a large number of messages with bogus digital signatures, which the defender would then waste its resources verifying. Since the signatures were bogus, there would offer no assistance to the defender in verifying the origin of the attack.

This problem has not been ignored by protocol designers, and there have been a number of proposals to reduce the denial-of-service risks involved in participating in a protocol that uses authentication, in particular strong authentication. These usually involve using weak authentication when the protocol is initiated, but stronger authentication as it completes. This, for example, is the approach followed by Kent et al. in [11], which applies these principals in a systematic way to design security enhancements for several internet routing protocols. The idea is that, even when it is within an attacker's capacities to break the weak authentication, it will still cost it a certain amount of effort

which may make it infeasible, or at least more difficult, for it to practice the multiple spoofing required to mount an effective denial of service attack. Thus the protocol provides protection against an attacker spending minimal resources 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 an attacker willing to spend greater resources. This does not leave the protocol completely invulnerable against a denial of service attack by a strong opponent, but it increases the cost to an attacker. For example, Photuris' use of cookies [10], in which a light-weight authenticator is attached to the beginning of each message and must be verified before any further message processing is done, is a classic example of this kind of approach. Cookies have since become a popular defense against denial of service in a number of protocols, including the version of TCP/IP used in Linux [2], where they are used to defend against the SYN attack that we described earlier.

Techniques like these assume an implicit model of the interaction between the attacker and defender. The attacker's goals are two fold: first to cause the defender to waste its resources by interacting with the attacker, and secondarily to keep the defender from learning its identity, since if it could, it could protect itself against further attacks by refusing to to engage with further communication with the defender. The defender is assumed to have bounded resources that could possibly be exhausted by a clever attacker. The attacker may or may not have bounded resources, and may or may not have resources greater than the defender, but there is assumed to be some level of resource expenditure beyond which the attack is considered to be difficult enough so that at least the likelihood of widespread attack is reduced.

Formal methods are one good way of addressing problems involving a lot of complex information that must be carefully evaluated. And indeed, formal methods have already been applied with great success in evaluating whether or not cryptographic protocols satisfy their authentication and secrecy requirements. Moreover, the underlying process, showing that certain authentication goals are achieved in the face of attack by an intruder expending a certain amount of effort, 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 involving resource exhaustion in a way that is intended to allow us to make maximal use of these tools. Although it is most applicable to cryptographic protocols, which use the most expensive form of authentication, it can be applied to any protocol that uses authentication, weak or strong, to protect against denial of service.

This framework extends our earlier model described in [16] by expanding the model of an intruder's capabilities is to include the cost of an action to the intruder as well as a list of actions of which we assume it to be capable. This allows us to model, not only the tradeoff between the cost to the defender

and the abilities of an attacker to implement different subsets of the Dolev-Yao model, as was done in the earlier paper, but also to compare the cost to the attacker directly with the cost of the defender, whenever this is possible.

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

2 The Framework

2.1 Motivation

In this section we describe the basic reasoning that underlies our framework, and the motivation for making the choices that we did.

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. However, it is not enough for the attacker simply to have the ability to pass off a bogus message as genuine. It also must be the case that the cost to the defender of accepting the message is non-trivial, and that the cost to the attacker is small enough in relation to its total resources that it finds the attack worth its effort. This means, first of all, that we need the capability of measuring cost to the attacker and cost to the defender.

There are a number of problems we must address when we model costs to attacker and defender. First of all, there is the problem of assigning and comparing costs of engaging in individual actions such as computing digital signatures, storing data, impersonating principals, etc. Different events may have costs that are not directly comparable. For example, the cost of computing a digital signature may not be directly comparable to the cost of storing a portion of a message for later reference, since they may use different resources. Moreover, what we may really be interested is not the raw cost in terms of memory or cycles used, but the ratio of cost to available resources. Finally, the actions available to an intruder may be different from those available to the defender; for example, the intruder, besides being able to perform the actions necessary to execute the protocol may also have the capability of impersonating principals, intercepting messages, and, in some instances, breaking cryptographic algorithms.

We may also need a way of computing the cost of several actions over time. When a principal participates in a protocol or accepts a message, it generally performs a sequence of actions instead of just one. What is the best way of computing the total cost of these actions when different actions may be using different resources?

Since we are merely developing a framework for possible models, not trying

to develop any specific model, so we do not need to solve this problem right away. However, we do not want to rule out any possible solutions, so we choose the most general structure that appears to be compatible with our needs. We will assume that there is a mapping from the set of possible protocol actions to a monoid with operator + which enjoys a partial order so that the sum of any two elements a and b dominates a and b. This monoid will be the set of possible costs, or *cost set*.

Once we have decided on a cost set and mapping, we need to decide over what sequence of operations we want to compute the cost. In order to do this, we will assume that there are two ways in which a defender could be caused to waste resources. One way would be by participating in a bogus instance of the protocol, up to the point at which an attack is detected. In this case, the cost of participating in the protocol for both attacker and defender must be the cost of engaging in the entire protocol up to the point at which the defender detects an error or the attacker stops participating. Another way would be in processing a bogus message inserted by the attacker into an ongoing protocol execution, which may or may not be legitimate. Assuming that the message contains an error, it could be replayed several times, until the defender times out. In this case the cost to the attacker is the cost of creating and inserting the message, while the cost to the defender is the cost of processing the bogus message up to the point at which it detects the error.

Once we have decided how to assign and compute costs, we still need to be able the characterize an attack on a protocol. This is not a trivial problem. There have been a number of different proposals for such characterizations. They generally fall into two camps: conservative models that require the intruder to have essentially no effect on a successful protocol execution, and more flexible models that allow the protocol analyst to set specific security goals. It is clear that in our case we are interested in the most conservative model possible, since it is possible for an intruder to use any deviation as a hook for a denial of service attack, even if it does not lead to violation of a protocol goal such as secrecy or authentication of keys. As a matter of fact, it is possible for any deviation to be used as a hook for denial of service even if it does not lead to a successful protocol execution. For this reason we need a definition of security that governs the behavior of protocols as they execute, not only after they terminate. Fortunately, such a model already exists: Gong and Syverson's fail-stop model [9].

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, by modifying our notion of an intruder's capability and introducing the notion of the cost to an intruder of its performing various activities.

We do this in two stages. First of all, we modify the notion of fail-stop by extending it to any action taken by a principal, not just the acceptance of a message. We can now define a function , from actions in a protocol to costs. We can then say that a protocol is fail-stop with respect to , , if a principal cannot be tricked into engaging in a protocol up to and including action A unless the attacker expends an effort of more than , (A). We are now in a state in which we can evaluate a protocol's resitance to denial of service. This proceeds roughly as follows (for the sake of clarity we omit some of the details).

We estimate the cost to the principal of engaging in all actions in a protocol up to and including A by adding their costs, and we compare this cost to , (A). If , (A) is trivial (from the point of view of the attacker) in comparison with the cost of engaging in the events up to and including A (from the point of view of the defender), then we can judge the protocol insecure against denial of service attacks. If the reverse is true for all actions A that a defender may engage in, then we can judge the protocol secure. Therey may also be gray areas in which neither , (A) nor the cost of engaging in all actions up to A are trivial in relation to the other, and a closer look may be necessary.

The rest of this section will be organized as follows. In Section 2.2 we introduce the protocol that we will use as an illustrative example throughout this section: the Station-to-Station protocol of Diffie, van Oorschot, and Wiener. In Section 2.3 we introduce our notation for specifying protocols such that each action is made explicit, and we show this notation can be used to describe the desired behavior of a protocol. In Section 2.4 we introduce the notion of cost and show how costs of protocol executions and are computed. In Section 2.5 we introduce an informal intruder model and show how costs of intruder actions are computed. In Section 2.6 we introduce the notion of a fail-stop protocol, and show how we modify it to meet our needs. In that section we also introduce the notion of tolerance relation, which can be used to model our estimate of the risk to a defender of being tricked into engaging in the protocol, and show how the tolerance relation and the modified fail-stop model can be used to analyze resistance to resource exhaustion attacks.

2.2 The Station to Station Protocol

The Station to Station protocol [4] is a protocol that was makes use of the Diffie-Hellman protocol together with digital signatures in order to exchange and authenticate keys between two principals. It was designed with message efficiency rather than resistance to denial of service in mind, and thus makes an interesting case study for our analysis.

We assume that two principals share a common modulus $\mathbf{P}$ and a generator α of the multiplicative group of $\mathbf{GF}(\mathbf{P})$. The protocol consists of three messages:

1. $A \to B \ : \ \alpha^{X_A}$

   A raises α to a secret value X_A and sends it to B.

2. $B \to A \ : \ \alpha^{X_B}, \ E_K(S_B(\alpha^{X_B}, \alpha^{X_A}))$

   B computes the key $K = \alpha^{X_A\,X_B} = \alpha^{X_A \cdot X_B}$. It then digitally signs α^{X_B} and α^{X_A}, and encrypts the result with K. It then sends this, together with the key half α^{X_B} to A.

3. $A \to B \ : \ E_K(S_A(\alpha^{X_A}, \alpha^{X_B}))$

   A computes $K = \alpha^{X_B\,X_A} == \alpha^{X_A \cdot X_B}$. It uses that key to decrypt the message. It verifies that that the decrypted message contains B's signature on α^{X_B} and α^{X_A}. It then computes a similar signed and encrypted message and sends it to B. Since B also knows K, it can decrypt the message and verify the signature as well.

As we can see, there are some obvious places where this protocol is open to resource exhaustion attacks. For example, when B receives the first message from A, it starts to expend resources generating a Diffie-Hellman key and computing signatures, even though it cannot yet be sure that the message came from A. As we shall see, our analysis will uncover some other vulnerabilities as well, along with some places where the protocol is not as open to denial of service attacks as might first appear.

2.3 Alice-and-Bob Specifications

In this section we introduce our notation for cryptographic protocols, which we will also use to specify correctness properties. 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 [8]. But in this case, a description of what does happen that 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* $A \ \to \ B : M$.

Since we are interested in producing a framework in which existing models can be integrated rather than a model itself, we will not attempt to introduce a syntax or semantics here, except to note that A and B are variables corresponding to names of communicating principals and M corresponds to a message sent between them.

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* $A \rightarrow B : T_1, ..., T_k \parallel M \parallel O_1, ..., O_n$. *We refer to the* T_i *as the* operations performed by A *and the* O_j *as the* operations performed by B.

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* N *from* A', $O_1, ..., O_n$ occur at B. *There are three types of events:* normal events *(which include the send and receive events),* verification events *(also called* verification operations*), and* accept events. *Normal events can occur at either sender or receiver. Verification events occur only at the receiver. The event* O_n *is a reserved event called the* accept event.

Note that the M sent by A is not necessarily the N received by B. This is intentional, since we will assume that the message N may have been created by a hostile intruder, and so will not necessarily be the same as M. As a matter of fact, if may be possible for B to receive N from A without A every having sent any M to B at all.

Briefly, the events, in sequence, correspond to creating a message, sending a message, receiving a message, processing a message, and accepting a message. The motivation for the different types of events is as follows. A normal event represents the operations performed that can have only one outcome, success. After completing a normal event, a principal always moves on to the next event, if any. A verification event has two possible outcomes, success or failure. If the event succeed, the principal performing it moves onto the next event. If the event fails, the principal halts. An accept event describes a principal's deciding, after, successfully completing all the message process events, that the message is genuine and it is ready to move on to the next stage.

We do not prescribe any particular notation for events, but for the purposes of this paper we will denote them by atomic symbols. Events that describe the same operation (e.g. two digital signature verifications) will be differentiated by subscripts.

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 show how this notation could be used to specify the Station-to-Station protocol. Some of the obvious operations used are exponentiation (which we will denote by exp), digital signatures (sign), verification of digital signatures (checksig), encryption (encrypt) and decryption (decrypt). Some of the less obvious are storing information (storename, storenonce), looking up information (retrievenonce, checkname). We should also include the use of pre-calculated exponentiated values, which represent a different use of resources than exponentiation in real time.

Once we have determined the various possible events, the specification appears as follows:

1. $A \rightarrow B$: $preexp_1, storename_1 \parallel$
   $\alpha^{X_A} \parallel$
   $storenonce_1, storename_2, accept_1.$

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

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

Although we do not attempt to give a formal semantics for Alice-and-Bob specifications, we do need to be precise about the intended and actual order in which events occur. For this, we use the notion of a *desirably-precedes* relation, which is similar to the notion used in Gong and Syverson's fail-stop model, and like theirs, is based on Lamport's causally-precedes relation [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 [6]. 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 in the specification 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 E_1 desirably-precedes E_2 and E_2 desirably-precedes E_3 then E_1 desirably-precedes E_3.*

Note that that in the definition of desirably-precedes, unlike our definition of events, we do require that the message received by B in the receive event corresponding to $A \rightarrow B : M$ be the same as the M sent by A. This is because

we are now describing what should happen when the protocol does proceed according to plan.

An Alice-and-Bob specification can be used together with the desirably-precedes relation to specify both the desired and actual behavior of a protocol. To understand how to do this, we need some notion of a semantics for such specifications. In the interest of generality, we avoid giving a formal semantics, but we provide the following informal semantics. This will be not only be used to give us a better idea of how to apply annotated Alice-and-Bob specifications, but we will also find it helpful in the definition of fail-stop protocols.

We begin by assuming that an annotated Alice-and-Bob specification corresponds to a set of programs, one for each principal variable described in the specification. Multiple copies of the programs may exist, corresponding to different instantiation of the principal variables, and these copies may be running concurrently. We may also assume that more than one copy may exist for the same instantiations of the principal variables, and these copies may also be running concurrently, allowing us to model the same principals taking part in multiple sessions. Each program corresponding to to a principal A can be broken down into a sequence of program fragments corresponding to the events engaged in by A. Program fragments corresponding to events engaged in by the same principal in the same copy of a program are assumed to execute in an order according to the desirably-precedes relation imposed by the specification, but no other assumption is made about the order of execution. In particular, there is no assumed ordering relation between the fragments corresponding to the sending and the receiving of a message.

We also assume that each program fragment has an output which can be either "success" or "failure". If the output is "success", the program proceeds to the next fragment. If the outcome is "failure", the program halts. Normal and accept events always output "success", while verification events can output "success" or "failure."

We model communication between programs as follows. The event "A sends M to B" corresponds to A placing M on a communication channel between A and B. When this happens, on of two things can occur. Either M is sent to B along the channel, or it is removed by an attacker. Likewise, the event "B receives M' from A" corresponding to A's send event describes B receiving a message along the channel between A and B. This could either be the genuine message M sent by A, or another message place on the channel by the attacker. We leave the discussion of our assumptions about how the attacker creates and inserts messages for Section 2.5.

Thus, in the case of the Station-to-Station protocol above, we assume that the program corresponding to A performs $preexp_1$ and $storename_1$, and sends a message to B. It then receives a message, ostensibly from B, and performs the actions $checkname_1$, $retrievenonce_1$, exp_2, $decrypt_1$, $checksig_1$, $accept_2$. If $checkname_1$ fails, it halts. Otherwise it continues until it hits the next verification event $checksig_1$. If that fails, it halts. If not, it proceeds to $accept_2$.

After $accept_2$, it executes the actions $sign_2$ and $encrypt_2$, and sends the message resulting from these actions to B.

2.4 Cost Sets and Protocol Cost Functions

Now that we have a notation for participation in a protocol, we can use it to compute the cost of participating in it. This is done by computing the cost of individual events, and summing up over the costs in an appropriate manner.

Definition 6 *A* cost set **C** *is a monoid with monoid operation* $+$ *with partial order* $<$ *such* $x + y \geq x$ *and* $x + y \geq x$*, for all* x *and* y *in* **C**.

Note in particular that $x = x + 0 \geq 0$ for all x.

Examples of cost could be things like time or money, in which case $+$ is simple addition and the partial order is total. Or, it could be a vector corresponding to different limited resources, such as computation time, storage space, etc., in which case we might want the partial order to be a lattice. 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. Or, it might represent the ratio of resources available to resources expended. Note that we do not even require that the $+$ operation be commutative, so that the cost of expending resources x before resources y could be greater than expending y before x.

Next, we need to specify a mapping from protocol events to costs.

Definition 7 *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.

For our analysis of the Station-to-Station protocol we do not have a very precise notion of cost available. Thus, we will use the simplest possible cost set, consisting of four members: $expensive > medium > cheap > 0$. We assume that cryptographic operations involving exponentiation during the protocol (*exp*, *checksig*, *sign*) are expensive. All other cryptographic operations (*encrypt*, *decrypt*, *preexp*) are medium, and all other events are cheap.

We now define two functions based on event 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. These correspond to the two ways in which we assume that an attacker can cause a defender to waste resources; first by processing a bogus instance of a message inserted by the attacker into a protocol run, and secondly by participating in a bogus instance of the protocol with the attacker.

Definition 8 *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

processing cost function associated with δ *to be the function* δ' *on verification 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 verification event* V_j*:*

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

The message processing cost describes the cost of processing a message up to an including a failed verification event.

Definition 9 *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*,.*

Note that the protocol engagement cost reflects not only the cost of all events up to and including the processing of the last message received, but the cost of composing any message sent as the result of sending that last message.

For the Station-to-Station protocol, we 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, as well as engage in all the previous events up to that point.

For the message processing cost functions of the Station-to-Station protocol, 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.

2.5 Attackers and Attacker Cost Functions

Up to this point we have said very little about attackers. The attacker occupies the no-man's land between the messages sent and the messages received. Basically, we assume that when a principal receives a message, it may be a legitimate

message as described by the Alice-and-Bob specification, or it may be a message concocted by the attacker. We do not attempt to give any kind of formal definition of an attacker, except to note that the attacker, like the legitimate principals, concocts and sends its messages using a combination of events that have cost. Thus we need to specify a set of attacker events and attacker costs.

Most of the protocol verification literature depends upon a standard set of assumptions about the attacker's capabilities. In our case, however, we leave it up to the protocol analyst to specify what these capabilities are. All that we will require is that each possible intruder action be assigned a cost as follows.

We use the following intruder cost function to compute the amount of effort that the intruder has to expend to interfere with a protocol.

Definition 10 *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. Let* $\mathbf{C}$ *be a cost set. We define* ϕ *to be a function from the set of intruder actions to* $\mathbf{C}$*. We extend* ϕ *to a function* Φ *from an intruder capability to* $\mathbf{C}$ *by defining* $\Phi(\{x_1, ..., x_n\}) = \phi(x_1) + ... + \phi(x_n)$*. We call* Φ *an* intruder cost function.

Note that we do not require the cost set used for the attacker to be the cost set used for the defender. In order to make this distinction clear, we refer to the two cost sets as the *attacker* and *defender cost sets*.

For the Station-to-Station protocol, we will let the attacker cost set be the defender cost set augmented by two costs, *very expensive* and *maximal*, obeying the ordering implied by their names. Very expensive is used for actions that may be difficult to implement (such as man-in-the-middle attacks), but which may be worth it if the rewards are large enough. Maximal is reserved for events such as cryptanalysis which are in most cases assumed to be infeasible but should not be left out of the reckoning altogether.

We thus assume that the intruder capabilities and costs are as follows:

1. sending a legitimate message (Cost = the cost of computing the message);
2. forging a return address (Cost = cheap);
3. reading messages (Cost = medium);
4. creating a new message out of old ones (Cost = cost of deconstructing the old messages + cost of creating the new ones);
5. disabling of a legitimate principal (Cost = medium: this requires another denial-of-service attack, but since a disabled principal can be used more than once, the cost can be amortized over a number of attacks);
6. substituting bogus messages for genuine ones in real time, as would be done in a man-in-the-middle attack (Cost = very expensive);

7. breaking cryptosystems (Cost = maximal), and;

8. inducing a principal to initiate communication with a bogus, disabled, or dishonest principal (Cost = expensive to very expensive: it is not hard to induce a principal to do this a few times, but it is probably difficult to induce this the number of times required to launch a denial-of-service attack).

2.6 Fail-Stop Protocols

Now that we have a notion of protocol specifications, attackers, defenders, and costs, we are able to show how we can modify Gong and Syverson's fail-stop model to specify desired levels of resistance to denial of service. We will begin by presenting the fail-stop model in terms of annotated Alice-and-Bob specifications. We will show how it can be modified to reflect cost to attacker and defender, and how it can be used in the analyze a protocol's vulnerability to denial-of-service attacks.

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 desired behavior. We have shown how an Alice-and-Bob specification can be used to described desired behavior. Now what we need is a way to describe deviations from that behavior.

Recall that we assumed that there exists some mapping of Alice-and-Bob specifications to programs associated principals such that each event maps to a program fragment, and that fragments belonging to the same program execute in the order specified by the desirably-precedes order induced by the specification, while we make no assumptions about the order of execution of fragments corresponding to different programs. We use this idea for the following definition.

Definition 11 *We say that an event as occurred if the program fragment corresponding to it has executed. We say that M has been* interfered with *if either the event B receives M from A occurs but some event desirably-preceding it has not, or the event B receives M from A has already occurred.*

Note that, as we have said previously, the fact that B receives M from A occurs does not mean the B actually received M or any other message from A. It simply means that B received a message at the point at which it was expecting the message M from A, and it is ready to perform a set of tests to determine whether the message was genuinely M and was sent by A. Thus although we use the term "interfered with" to be consistent with the language in [9], we note that it covers not only messages that were tampered with, but fake messages generated by an intruder.

We now give the definition of fail-stop from [9], modified slightly to take into account our slightly different definition of an event.

Definition 12 *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 now present our modified version of fail-stop. Since we need it to incorporate the notion of the effort involved in interfering with messages, we use the following attack cost function to compute the amount of effort that we want the intruder to have to expend to interfere with a protocol.

Definition 13 *Let Θ be a function from the set of events defined by an annotated Alice-and-Bob specification P to a cost set C. We refer to Θ as the* attack cost function. *We say that P is* fail-stop with respect to Θ *if, for each event E in the system, if an intruder interferes with any message desirably-preceding E, then neither E nor any events desirably-after E will occur, unless the cost of the capabilities the intruder uses in interfering with the message or messages is at least $\Theta(E)$.*

If we assume the cost of exercising the usual Dolev-Yao set of intruder capabilities is some value M, and we let $\Theta(E) > M$ whenever E is an accept event, and let it be zero for all other events, then the reader can verify that our definition of fail-stop is equivalent to Gong and Syverson's.

The question we are left with, of course, is which Θ a protocol analyst would want to use. Clearly, it should bear some relation to the defender cost functions. We make this idea more precise below.

Definition 14 *Let C be a defender cost set, and let G be an attacker cost set. We define a* tolerance relation *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 attacker cannot force a defender to expend resources of cost c or greater without revealing its identify or expending resources of cost g or 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 $g' \geq g$.*

In other words, the tolerance relation reflects the result of performing a risk analysis in which an estimate of the attacker's resources and its willingness to expend them is traded off against an estimate of the defender's available resources.

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

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. Determine the minimal attack cost functions with respect to which the protocol is fail-stop.

4. For each attack cost function Θ defined in Step 3, determine that:

   a) If E_1 is an event immediately preceding a verification event E_2, in a line of a protocol, then $(\delta'(E_2),\Theta(E_1))$ is within 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 any intruder will have to expend cost $\Theta(E_1)$ in order to successfully interfere with M after E_1 has finished executing.

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

Note that Step 4a allows us to reason about the ability of a protocol to thwart an intruder who is only willing to expend a certain cost and who tries to mount 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 forged by such an intruder. Step 4b allows us to reason about the ability of a protocol to thwart an intruder who is only willing to expend a certain cost and who tried to mount 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 forged by such an intruder.

We show how this procedure would be applied to the Station-to-Station Protocol as follows.

We first compute the attack cost functions and the protocol engagement cost functions for the accept events.

Clearly, since the first message is not authenticated at all, it protects only against a very weak intruder, so at best we can take $\Theta(accept_1)$ to be is the cost of creating and sending the message. This at first appears to be medium, but in order to get this message accepted, the intruder does not have to include an exponentiated value. All it needs to do is substitute a field taking up the appropriate amount of space, although it may have to take some care to be able to pass checks for obviously bogus values such as are recommended in [4]. Thus the cost is actually cheap. On the other hand, $\Delta(accept_1)$ is expensive, since B is required to perform expensive exponentiation and computation of digital signatures as a result of accepting the message.

We can take $\Theta(accept_3)$ to be maximal, in the sense that we have verified using the NRL Protocol Analyzer that an intruder expending less than maximal cost according to our model cannot break the protocol. (Note, however, that

this is not the same as a full-scale proof that breaking the protocol is equivalent to breaking the underlying crypto-systems.)

What is surprising, however, is $\Theta(accept_2)$. We would expect this to also be maximal, but as a matter of fact it is only very expensive, as it involves a man-in-the middle attack. As was shown by Lowe in [14], the event $accept_2$ is vulnerable against the following attack, where I is the intruder, and I_Z is the intruder impersonating Z:

1. $A \rightarrow I_B \ : \ \alpha^{X_A}$
2. $I_C \rightarrow B \ : \ \alpha^{X_A}$
3. $B \rightarrow I_C \ : \alpha^{X_B}, \ E_K(S_B(\alpha^{X_B}, \alpha^{X_A}))$
4. $I_B \rightarrow A : \alpha^{X_B}, \ E_K(S_B(\alpha^{X_B}, \alpha^{X_A}))$
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. On the other hand, $\Theta(accept_2)$ is merely expensive.

We now compute the attack cost functions and the message processing cost functions for each verification event.

The first verification event is $checkname_1$. Since no verification is done before the name is checked, the associated Θ is cheap. However, since no event except the reception of the message precedes $checkname_1$ at that line, and the cost of $checkname_1$ is itself cheap, the pair (c,g) computed should be well within any tolerance relation.

The next verification event is $checksig_1$. The message processing cost $\delta'(checksig_1)$ is expensive. The event appearing immediately before $checksig_1$ is $decrypt_1$. In order to spoof $decrypt_1$, the intruder has at least to be able to induce A to initiate contact with a bogus, dishonest, or disabled B. The intruder can then construct a bogus message; for this the only capability required is the ability to read A's message, and possibly to fake B's return address. Thus $\Theta(decrypt_1)$ is at least expensive to very expensive. This may or may not be within a protocol designer's tolerance relation.

The next verification event is $checkname_2$. The event immediately preceding it is B's receipt of the third message in the protocol. Again, the message acceptance cost $\delta'(checkname_2)$ is cheap, so no matter what the value of $\Theta(A \rightarrow B : M)$, it should be within most tolerance relations.

Finally, the last verification event is $checksig_2$. Again, the message acceptance cost of $checksig_2$ is expensive. In this case, however, the attacker has two choices for tricking the B into applying $checksig_2$ to a bogus message. First of all, it could attempt to hijack an existing session between A and B in which A sent a legitimate first message. The cost of this would be expensive, and thus

the resulting pair (c,g) would be within most tolerance relations. The attacker's other option, however, would be to impersonate A from the beginning. Since no verification is made on A's first message to B, this would only require the attacker to be able to spoof $checkname_2$, and possibly to disable A prior to executing the protocol. Thus $\Theta(decrypt_2)$ is at most medium, and the resulting (c,g) = (expensive, medium) is probably not within most tolerance relations.

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 leading up to the final expensive check on the last message are weak, an intruder who is capable of faking return addresses and diverting messages could cause either the responder to waste resources in processing a bogus message. Finally, though less seriously since it is the most expensive of the attacks, an 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 IKE [5], 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 by Lowe in [14].

3 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 [3] 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 expending various amount of effort. 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 [15] 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. In [16], in which we modeled denial-of-service in terms of intruders of different strengths, we recommended replacing this implementation of the intruder model with implementations of a number of models representing intruders of different strengths, and to verify different goals of the protocol with respect to the appropriate model. However, with the cost-based framework described in this paper, it may make more sense to use a single intruder model, but to keep a running tally of the cost involved as an attack is constructed. Attacks that exceed the recommended costs would be ignored. A similar tally might also be appropriate for use in theorem-proving approaches such as [23] which also rely upon an explicit model of the intruder, although they do not compute attacks directly. Note that in some cases computing the cost could be a little tricky, especially in the case of such intruder actions as man-in-the-middle attacks which describe intruder behavior over several different states. However, this should still be easier than implementing a number of different intruder models.

The fact that we are modeling the security of a protocol in terms of an intruder of limited capacity also invites comparison with the vast amount of work that has been done using limited-capacity models in the cryptographic literature. In these models a protocol or algorithm is proven secure in face of an intruder who has limited computational capacity, for example, an intruder who is only able to solve polynomial-time problems. The main difference between these models and ours is that these models generally only concern restrictions on the *computational* capacity of the intruder and use one set of assumptions to reason about a given protocol, while our framework includes restrictions on the capacity to perform various network operations and uses different sets of restrictions to reason about different points in a protocol's execution. However, a closer study of the ways in which assumptions about the computational capacity are used to reason about protocol security, particularly in work such as that of Bellare and Rogaway [1] and Lincoln et al., [13], which analyzes protocols similar to the ones we are considering in our work, would probably yield results that could be used to enhance our framework.

Finally, we consider the applicability of high-level protocol description languages, such as CAPSL [21] and Casper [15]. 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. A protocol designer might also want to specify the desired attack cost function for various protocol subgoals; this could be built into the specification as an annotation.

4 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, in particular in the development of more realistic and sensitive cost functions. We used a rather crude and ad hoc cost function in our analysis of the Station to Station protocol, simply in order to illustrate how our framework could be applied. However, enough is known about the difficulty of the various operations used in executing and attacking a cryptographic protocol that it should be possible to refine it considerably. We expect that the main challenge will be in comparing the difficulty of different sorts of operations, e.g. how do we compare the difficulty of diverting a message with the difficulty of computing a digital signature? This will probably be implementation-dependent and require careful analysis, breaking each action down into its component parts. Once this work is done, however, it should easily fit into our framework.

There are several other issues that remain to be explored. One is the use of variable levels of defenses. In many cases, it is becoming the practice to use one level of verification when the environment is believed to be benign, and another more stringent, level of verification when a system is believed to be other attack. For example, in the benign case authentication checks may be made on only a portion of the system, while in the case that the system is under attack more rigorous authentication checks may be used. We can model this by using two different tolerance relation, one for the benign situation (in which we require security only against weak attackers), and one in the attack situation (in which we require security against strong attackers). It should then be possible to evaluate the two different authentication policies using the two different attack scenarios.

We have a final comment to make about applying our framework to protocols that do not use strong authentication. In this paper we have mainly looked at protocols that use some form of strong authentication to achieve their goals, and the framework we present is derived from models used to reason about protocols that use strong authentication. However, authentication is a useful method for protection against denial of service attacks even for protocols for which it is not practical against cryptography. For example, one of the most common form of attack is simply to flood a channel with messages. This is can actually be seen as a denial of service attack on the router, and thus could be defended against, at least in part, by requiring the router to perform lightweight authentication. But even the simplest checks can have a nontrivial cost to a router. Thus it is conceivable that a framework like ours could be useful in analyzing the effect of using different types of authentication on even low-level protocols that use light-weight authentication measures. We plan to investigate these issues in our further work.

5 Acknowledgments

We would like to thank the anonymous reviewers for helpful comments on earlier versions of this paper. We are also grateful to Stephen Brackin, David Opitz, John McHugh, and Chad Dougherty for helpful discussions.

This work was supported by ONR. Portions of this paper appeared previously in [16].

References

[1] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology - CRYPTO '95*. Springer-Verlag, LLNCS 773, 1995.

[2] D. J. Bernstein. SYN cookies. `http://cr.yp.to/syncookies.html`. See also The Linux Documentation Project, `http://metalab.unc.edu/mdw/index.html`.

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

[4] W. Diffie, P. C. van Oorschot, and M. J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, pages 107–125, 1992.

[5] N. Doraswamy and D. Harkins. *IPSec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks*. Prentice Hall, 2000.

[6] 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.

[7] 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.

[8] 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.

[9] 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 79–99. IEEE Computer Society, 1998.

[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] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, San Francisco, CA, November 1998. ACM.

[14] 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.

[15] 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.

[16] C. Meadows. A formal framework and evaluation method for network denial of service. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pages 4–13. IEEE Computer Society Press, June 1999.

[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 analysis of the Internet protocol TLS. Report 440, Cambridge University Computer Science Laboratory, 1998.