Yale University
Department of Computer Science

Analyzing DoS-Resistance of Protocols Using a Cost-Based Framework

Vijay Ramachandran

YALEU/DCS/TR-1239
July 2002

The author was supported in part by a 2001 U.S. Department of Defense NDSEG Fellowship and ONR Grant N00014-01-1-0447.

Analyzing DoS-Resistance of Protocols Using a Cost-Based Framework

Vijay Ramachandran
Department of Computer Science
Yale University
vijayr@cs.yale.edu

Abstract

This paper addresses protocol susceptibility to denial-of-service attacks. We analyze protocol fragments using Meadows's cost-based framework [9] to identify sequences of actions that render servers disabled, either due to memory or CPU exhaustion. In particular, we show that the JFK protocol [1] is DoS-resistant when bogus messages are handled in an appropriate way. We also discuss the relationship between the protocol properties *fail-stop* [7] and *non-interference* [5] and DoS-susceptibility.

1 Introduction

Designing protocols to be resistant to denial-of-service (DoS) attacks involves finding a careful balance between authentication and resource use. Strong authentication can often prevent attackers from generating bogus messages or proceeding far in a protocol execution without providing their identities, but its use can require extensive computation that can exhaust the resources of the server. Weak authentication methods prevent attackers from overwhelming the server's computational resources but may allow attackers to spoof messages without much difficulty. Therefore, both of these can lead to protocols that are susceptible to DoS attacks. Other standard practices, such as storing state during a protocol execution, can also cause susceptibility to denial of service.

Meadows [9] introduced a novel method of evaluating protocols' susceptibility to DoS. By assigning costs to actions in a protocol execution, a protocol analyst can determine the resources needed for an attacker and a legitimate agent to participate in a protocol up to a given point. If an attacker requires little work to exhaust the resources of a legitimate participant, the protocol is DoS-susceptible. On the other hand, if the relationship between the attacker's amount of work and the legitimate agents' amount of work is within

The author was supported in part by a 2001 U.S. Department of Defense NDSEG Fellowship and ONR Grant N00014-01-1-0447.

some reasonable threshold, the protocol is practically DoS-resistant. The costs and thresholds can be set to model various situations; thus, a protocol theoretically susceptible to a DoS attack could be found useful in practice, *e.g.*, because the attack might require extensive computation that would be too expensive in the short amount of time available.

We apply this cost-based framework to some protocol fragments and, as a result, reveal some standard DoS flaws, including some found in the Internet Key Exchange (IKE) protocol [8]. The analysis suggests several practices that should be avoided when designing protocols to be DoS-resistant. A careful review of the Just Fast Keying (JFK) protocol [1] shows that, when error situations are handled properly, the protocol manages to resist common DoS attacks.

Meadows's cost-based framework extends a protocol property earlier studied by Gong and Syverson [7] called *fail-stop*. A fail-stop protocol is essentially one that stops when a message that is "interfered with" (one that is tampered with, out-of-sequence, or bogus) is detected. Clearly, there is a relationship between protocols with this property and those that are DoS-resistant, because cheap detection of bogus messages could prevent attacks. *Non-interference*, a property studied by Gorrieri [5], is also related to denial of service. A protocol satisfying non-interference does not allow completion of protocol steps when they are initiated by an illegitimate agent, such as an attacker. A non-interfering protocol, then, should prevent the introduction of bogus or tampered messages and so seems to be DoS-resistant. Gorrieri has created several tools [4] to analyze protocols for non-interference. We study the relationship between fail-stop, non-interference, and DoS-resistance and explore the utility of evaluating protocol DoS-resistance in terms of these protocol properties.

The remainder of the paper proceeds as follows. Section 2 outlines the cost-based framework that we use to analyze protocols. Section 3 gives our results from applying the framework to protocol fragments. Section 4 discusses the relationship between DoS-susceptibility and the two protocol properties discussed above, *i.e.*, fail-stop and non-interference. Section 5 concludes the paper.

2 Cost-Based Framework

To analyze protocol fragments, we use Meadows's cost-based framework [9]. In this section, we describe the framework and its important definitions.

The motivation behind the framework is that DoS-susceptible protocols have the following property: At the point in a protocol execution at which an attacker can send a message to cause a DoS attack, the cost of doing so in relation to his resources is small, while the cost to the defender to accept and process the message is relatively more expensive. If this relationship of costs does not occur during a protocol execution, we can say that the protocol is DoS-resistant.

To formalize this notion, the framework has basic definitions to describe the assignment of costs to actions. We require the following property for costs.

Definition 1 *A* cost set *C is a monoid with operator $+$ and partial order $\leq$ such that $x \leq x+y$ and $y \leq x+y$ for all $x, y \in C$.*

Examples of cost sets can be found at the end of this section. In particular, elements can be a set of qualitatively different levels of computational difficulty or vectors whose components represent resource use during the protocol execution.

The following definitions outline how protocols are formally specified.

Definition 2 *An* annotated Alice-and-Bob specification P *is a sequence of statements of the form* $L: \ A \rightarrow B : T_1, \ldots, T_k \| M \| O_1, \ldots, O_n$.

- $\mathcal{E}$ *is the* agent capability, i.e., *the set of actions available to the agents.*
- M *is the message supposedly sent from* A *to* B *at step* L *in the protocol.*
- $\{T_i\} \subset \mathcal{E}$ *are the* operations performed at A, i.e., *the ordered sequence of actions involved in preparing* M.
- $\{O_j\} \subset \mathcal{E}$ *are the* operations performed at B, i.e., *the ordered sequence of actions involved in processing* M.

We write actions as atomic symbols, *e.g.*, *encrypt*. The difference between actions and operations is that operations constitute ordered sequences of actions, and their placement in a given protocol specification is important. Protocol specifications often require actions to be performed multiple times in different contexts; thus, in order to distinguish operations of the same action, we will write an operation as the atomic symbol for the corresponding action with a numbered subscript indicating its position (*e.g.*, $accept_k$) or with parameters indicating its specific use (*e.g.*, $hash(X)$).

The operations in a protocol specification are further classified into three types of *events*.

Definition 3 X *is an* event *in line* $L: \ A \rightarrow B : T_1, \ldots, T_k \| M \| O_1, \ldots, O_n$ *if it is*

- *one of the operations at* A, T_i,
- *one of the operations at* B, O_j,
- *"*A *sends* M *to* B*," or*
- *"*B *receives* M *from* A*."*

There are three types of events.

1. Normal events *are those that always succeed and are followed by the next event in the protocol specification. These include the send and receive events and computation events such as exponentiation.*
2. Verification events *are operations that can result in success or failure,* e.g., *checking that two quantities are equal. A successful event is followed by the next event in the protocol specification. A failure is followed by some error-handling procedure so that normal protocol execution stops. Verification events only occur at the receiver* B.

3. *The* accept event *is always the last operation on a line, O_n, which indicates that B has completed processing M and that execution of step L is complete. We assume that accept events always result in success, as they should only be reached after a successful sequence of verification events that process M.*

We are interested in assigning costs to the preparation actions T_i and the processing actions O_j so that the total cost of reaching a step in the protocol where M is accepted (at O_n) can be calculated for the sender and receiver. These costs are defined as follows.

Definition 4 *Let P be an annotated Alice-and-Bob specification, let $\mathcal{E}$ be the set of operations in P, and let $\mathcal{C}$ be a cost set for those operations.*

- *δ is an* event-cost function *if it maps $\mathcal{E}$ to $\mathcal{C}$ and is* 0 *on the accept events. δ defines the agent cost structure used in protocol analysis.*
- *The* message-processing cost function for *δ is the function δ' defined on verification events $\{V_i\} \subset \{O_j\}$ such that for $A \rightarrow B : \ldots \| M \| O_1, \ldots, O_n$, if $V_i = O_j$ then $\delta'(V_i) = \delta(O_1) + \cdots + \delta(O_j)$.*
- *The* protocol-engagement cost function for *δ is the function Δ defined on accept events O_n such that $\Delta(O_n)$ is the sum of all the costs of operations occurring at the receiver up to O_n, plus the costs of any preparation operations if the receiver immediately sends a message in response*[1] *to the accept event.*

The above definition carefully includes the resources exhausted by agents to process a message and perform any naturally subsequent steps. This is important, because the costs incurred in preparing the next message might well be those that consume enough resources to cause denial of service.

Analogously, we define the cost structure for the attacker.

Definition 5 *Let $\mathcal{I}$ be the* intruder capability*, which is the set of actions available to an attacker, including those that affect messages received by legitimate protocol participants. Let $\mathcal{J}$ be a cost set for these actions, and let ϕ map $\mathcal{I}$ to $\mathcal{J}$. Let Φ, the* intruder cost function*, be defined on a sequence of intruder actions as $\Phi(\{i_1, \ldots, i_n\}) = \phi(i_1) + \cdots + \phi(i_n)$ for $i_k \in \mathcal{I}$.*

The attacker cost set and capability need not be the same as those of the legitimate participants (*i.e.*, the defender) in Definition 4.

We are interested in comparing the cost to the attacker and the cost to the defender to reach specific points in the protocol. Essentially, a protocol would be DoS-resistant if the cost for an attacker to interfere with messages in the protocol exceeds some reasonable amount. The following definitions state this more precisely.

[1] This definition is enough for an intuitive understanding of which actions are included in calculating the cost of reaching a point in a protocol. For a more precise definition of "immediately responding with a message" in terms of *causally-after* and *desirably-after* relations, see Meadows's original paper about this framework [9].

Definition 6 *Let* Θ*, the* attack-cost function*, map from an annotated Alice-and-Bob specification* P *to a cost set* C*.* P *is* fail-stop with respect to Θ *if, for every event* $E \in P$*, if an intruder interferes with a message that should arrive at or before* E*, then no events that should occur after* E *will occur, unless the cost to the intruder is at least* $\Theta(E)$*.*

For every protocol P, we can determine the minimal attack-cost function Θ such that P is fail-stop with respect to Θ. By minimal, we mean that evaluating the function at each accept event gives the cost the attacker must incur in order to cause the legitimate participants to reach that accept event without detecting an attack. (To rigorously analyze P, we want to determine the "cheapest" possible way for an attacker to succeed, and the attack cost function represents these costs.) P will be fail-stop with respect to Θ as long as, for all events $E \in P$, $\Theta(E) < c$, where $c \in \mathcal{J}$ is the cost of interfering with the message corresponding to E (*i.e.*, $c = \Phi(S)$, where $S \subset \mathcal{I}$ is a sequence of intruder actions). In other words, Θ represents the vulnerability of P. We next want to define a criterion to evaluate whether this Θ is reasonable, *i.e.*, whether it establishes a difficult cost threshold for attackers to interfere with messages. The following definition further clarifies the role of the attack cost function in terms of a relationship between attacker and defender resources.

Definition 7 *Let* $\mathcal{C}$ *and* $\mathcal{J}$ *be the defender and attacker cost sets. Then* $\mathcal{T} \subseteq \mathcal{C} \times \mathcal{J}$*, the* tolerance relation*, is defined as follows:* $\mathcal{T}$ *contains all pairs of costs* (c, j) *such that the defender will expend cost* c *in resources only if the attacker reveals his identity or expends cost* j *or greater in resources.* (c', j') *is* within the tolerance relation *if there is a* $(c, j) \in \mathcal{T}$ *such that* $c' \leq c$ *and* $j' \geq j$*.*

The tolerance relation should be defined to model reasonable circumstances. The cost sets can account for memory and computation resource consumption, and the tolerance relation can be designed to take into account the accumulation of costs over a period of time or several protocol instances. This is important when considering distributed denial-of-service (DDoS) attacks.

Example Cost Structures We now present some examples of cost sets that are useful for analyzing different aspects of protocol executions.

Example 1 The following cost structure describes an informal way to track the amount of computation required through a protocol and is found in [9].

Let $\mathcal{C}$ be the monoid containing the following costs and order: $0 <$ cheap $<$ medium $<$ expensive. Addition on $\mathcal{C}$ is defined as taking the maximum given this order (*e.g.*, cheap + expensive = expensive). Let $\mathcal{E}$ contain the actions found in the protocol specification. The event-cost function δ is defined with the following guidelines: cryptographic operations involving exponentiation are expensive, including some public-key encryption, some key generation, and some signature operations; other cryptographic operations are medium, including hashing and most symmetric-key encryption and decryption; all other operations are cheap. $\delta(accept) = 0$.

Let $\mathcal{J}$ be the monoid containing the costs and order above, but with the additional two costs very expensive and maximal. The intruder capability $\mathcal{I}$ contains actions for standard computation, like the defender capability $\mathcal{E}$, but we also take into account that the attacker might do the following.

- Spoof a message. The cost depends on what properties the bogus message must have. We assume that $\phi(\mathit{spoof.return.address}) =$ cheap, but $\phi(\mathit{spoof.exp})$, where $\mathit{spoof.exp}$ is the action to spoof the result of an exponentiation, will vary depending on how that value is verified (*e.g.*, see the discussion in Section 3.2). On the whole, the cost of sending a message will take into account the cost of preparation events preceeding it (by definition of Φ in the framework), *e.g.*, deconstructing other messages and minor computation.

- Read messages in real-time. We assume the attacker has access to the communication channels between legitimate protocol agents, but discerning the necessary information from any messages along these channels is non-trivial. Thus, $\phi(\mathit{eavesdrop}) =$ medium.

- Disabling a legitimate principal X. Meadows sets this cost, $\phi(\mathit{disable}(X))$, to medium, because X can be used more than once; the cost of the DoS attack used to disable X can be amortized over a number of attacks.

- Substituting messages in real time or dropping messages between participants. This is basically a man-in-the-middle attack, and its cost is very expensive.

- Cryptanalysis. Breaking cryptosystems, computing discrete logarithms, factoring numbers, NP-hard computations, and violating other standard cryptographic assumptions are all of maximal cost.

A reasonable tolerance relation would include the pairs $(0, 0)$, (cheap, cheap), (medium, medium), and (expensive, expensive). A more restrictive relation might include (cheap, medium) and (medium, expensive) instead. A more tolerant relation might include (medium, cheap). Under all circumstances, (expensive, cheap) is probably an unacceptable pair to include in the tolerance relation.

Example 2 The following structure is simpler and more useful for tracking different resources. In this paper, we will only consider computation (CPU time) and storing state (memory). Let $\mathcal{C}$ and $\mathcal{J}$ be monoids consisting of pairs of non-negative rational numbers and the naturally induced order, where $(p, q)+(0, r) = (p, q + r) \geq (p, q)$, $(p, q) + (r, 0) = (p + r, q) \geq (p, q)$, and $(p, q) + (r, s) = (p + r, q + s) \geq (p, q)$, but we impose no order between $(p + r, q)$ and $(p, q + s)$. The tolerance relation, however, might suggest something about how to compare such tuples. The first component will represent computation, and the second will represent memory usage.

The tolerance relation can reasonably contain all pairs (q, q). In this paper, we will refer to one reasonable pair in the relation to make our point, rather than carefully define the complete relation to describe all possible scenarios.

3 Analysis of Protocol Fragments

In this section, we apply the Meadows framework to several protocol fragments and discuss susceptibility or resistance to denial of service.

3.1 Storing State Without Authentication

The canonical example of denial of service is the TCP SYN attack [2], in which many instances of a three-way handshake are initiated but not completed; the server consumes some amount of memory to store state for each incomplete handshake and eventually overflows its buffer, becoming incapable of handling any more protocol instances. One way to recover might be to drop open handshakes after some fixed time interval, but this is not a complete solution. Doing so could interrupt legitimate communication, and the level of incoming bogus messages often is so high in an attack that the server is rendered useless anyway.

This attack is not specific to TCP. Any protocol that requires the responder to store state for an initiator without authentication is subject to attack, because the messages causing resource consumption can be easily generated by an attacker. IKE's aggressive mode [8, 3] requires the responder to store security-association (SA) information upon receipt of the first message in order to establish a secure channel for later communication.

The protocol fragment has the form

$$I \rightarrow R: \ \textit{createmessage} \parallel M \parallel \ \ldots, \textit{storestate}, \ldots, \textit{accept} \tag{1}$$

where some inexpensive operations have been omitted for clarity. Let our cost sets be sets of pairs representing computation and memory resources as in Example 2. Because there is no authentication, preparation of the message is cheap and requires no memory, so $\delta(\textit{createmessage}) = (0,0)$. On the other hand, storing state in response to the message requires some memory; so let us assume that $\delta(\textit{storestate}) = (0,1)$. (This is a very generous assignment, because we assume that no computation is required.) We then have the situation in which $\Delta(\textit{accept}) = (0,1) + d$, where d is the preparation cost of a message, if any, that R sends in response to M. On the other hand, $\phi(\textit{createmessage}) = (0,0)$, and so $\Theta(\textit{accept}) = (0,0)$, which is an unacceptable attack-cost function. With essentially no work, the attacker can force the responder to consume more and more memory. This situation would not fit within any reasonable tolerance relation.

Because only one message is required to force state storage (there is no interaction in the above fragment), an attacker can easily hide his identity by spoofing the source of message M. This makes the vulnerability even more serious.

Note that one alternative to storing state — passing it back and forth using later messages — may not solve the problem. Without some other work, it would be impossible for the responder to tell whether or not later messages are out of sequence, because, without storing some state, it would be difficult to determine whether or not messages were part of a sequence that started at the beginning of the protocol. An attacker might be able to spoof the contents of later messages, forcing the responder to do some processing anyway. Therefore, some form of escalating authentication is necessary before any expensive processing on the part of the responder occurs.

In summary, we have the following.

Result 1 *Protocols that require storing state without authentication,* i.e., *containing fragments like (1), are susceptible to DoS attacks.* $\Theta(accept) = (c, 0)$ *where* c *is the cost of creating the bogus message in (1), while* $\Delta(accept) = (v, 1) + d$*, where* v *is the cost of verifying the bogus message, and* d *is the cost of preparation operations in the line following (1). Unless* $c, v,$ *and* d *are such that* $(\Delta(accept), \Theta(accept)) \in \mathcal{T}$*, the state storage makes the protocol vulnerable.*

3.2 Immediate Strong Authentication

Protocols that require immediate strong authentication (for example, a Diffie-Hellman exchange) may prevent bogus messages and secrecy leaks, but they are subject to DoS attacks because of the expensive operations frequently required in strong authentication.

Consider the protocol fragment

$$
\begin{aligned}
L_1 : \quad & I \to R : \quad \mathit{createexp}(g^a) \parallel g^a, [\text{group-info}] \parallel \mathit{verify}, \mathit{accept_1} \\
L_2 : \quad & R \to I : \quad \mathit{createexp}(g^b), \ldots \parallel g^b, \ldots \parallel \ldots, \mathit{accept_2}
\end{aligned}
$$

We can use Meadows's basic cost set described in Example 1 to analyze this fragment. In L_1, the initiator is supposed to send an exponential to the responder, and, after accepting it, the responder is supposed to send an exponential in response in L_2.

The execution could proceed in two ways, depending on the thoroughness of the operation *verify*. If $\delta(\mathit{verify}) =$ cheap, then we can assume that the responder cannot check the validity of the exponential g^a thoroughly enough to ensure that the event would fail on a bogus g^a. In this case, the protocol-engagement cost of L_1 should include the cost of creating the exponential in L_2; thus $\Delta(\mathit{accept_1}) =$ expensive. The inadequacy of *verify* also justifies assigning the cost $\phi(\mathit{spoof.encrypt}) =$ cheap for the attacker. Thus, $\Theta(\mathit{accept_1}) =$ cheap. $(\Delta(\mathit{accept_1}), \Theta(\mathit{accept_1}))$ does not fit in any reasonable tolerance relation; so we can say this fragment is DoS-susceptible.

If *verify* can check the validity of the exponential g^a, then it is fair to assume that its cost is more than cheap. There are two subcases. The attacker might attempt to generate an exponential to pass the *verify* operation. This would increase the cost of $\mathit{spoof.exp}$. Because $\Delta(\mathit{accept_1}) =$ expensive (because the responder would exponentiate), doing so might be a fruitful attack. Depending on the value of $\phi(\mathit{spoof.exp})$ (corresponding to the strength of *verify*), $(\Delta(\mathit{accept_1}), \Theta(\mathit{accept_1}))$ might still fall outside the tolerance relation. On the other hand, if $\delta(\mathit{verify})$ is very large, the attacker might send a garbage exponential, setting $\phi(\mathit{spoof.exp}) = \text{cheap}$. With $\Theta(\mathit{accept_1}) = \text{cheap}$, $(\Delta(\mathit{accept_1}), \Theta(\mathit{accept_1}))$ would not fit in any reasonable tolerance relation, because $\Delta(\mathit{accept_1})$ includes the cost of *verify*.

Strong authentication methods, in general, would require the same level of computation as the specific case of exponentiation; thus, this attack can be generalized to other methods. In particular, forcing public-key decryption is a simple way to exhaust the server's computation resources. Assume that, in the following fragment, the initiator knows the responder's public key. This is a fair assumption, because eavesdropping on a public-key request or contacting a third-party are easy ways to obtain the public key without revealing one's identity to the responder.

$$\begin{aligned} I \to R: \quad & \mathit{assemble}(M), \mathit{encrypt}(M, k_{\text{pub}}) \parallel [M]_{k_{\text{pub}}} \\ & \parallel \mathit{decrypt}([M]_{k_{\text{pub}}}, k_{\text{priv}}), \mathit{verify}, \ldots, \mathit{accept} \end{aligned}$$

For the purposes of this example, assume that $\phi(\mathit{assemble}) = \text{cheap}$, because M can contain complete garbage. In addition, $\phi(\mathit{spoof.encrypt}) = \text{cheap}$, because only some trivial work is required to disguise $[M]_{k_{\text{pub}}}$ as a possibly legitimate message. On receipt of the message, however, the responder must perform a decryption to verify the contents, and $\delta(\mathit{decrypt}) = \text{expensive}$. Therefore we have the situation in which $\Delta(\mathit{accept}) \geq \text{expensive}$, $\Theta(\mathit{accept}) = \text{cheap}$, and the identity of the initiator need not be revealed, which is unsatisfactory.

In summary, we have the following.

Result 2 *Protocols that require the responder to exponentiate or perform expensive cryptography operations when receiving a message that cannot be authenticated cheaply are susceptible to DoS attacks. In particular, unless there exists some authentication operation* verify *with $\delta(\mathit{verify})$ comparable to $\phi(\mathit{spoof})$ (where* spoof *is the intruder action to assemble the message), the protocol-engagement cost of the message will not fit the tolerance relation $\mathcal{T}$.*

The analysis here demonstrates that protocol designers should be wary of steps that accept messages that require expensive decryption with too-weak (easily spoofed) or no other authentication in the same message. JFK is a good example of a protocol in which medium-level authentication is used to prevent attacks such as the above. A more detailed analysis of JFK appears in Section 3.4.

3.3 Fast-Escalating Authentication

Weak authentication can sometimes be used to prevent resource exhaustion by requiring simple verification before any state is stored or expensive computations are performed. Unfortunately, authentication that is too weak can be spoofed by eavesdropping. Additionally, if an attacker has the resources to launch a DDoS attack and is willing to reveal the identity of the machines involved, the protocol can cheaply proceed to the point where the responder is forced to exhaust resources.

Consider the following protocol fragment.

$$\begin{array}{lll}
L_1: & I \to R: & \mathit{computenonce}(N_I), \mathit{storenonce}(N_I) \parallel N_I \parallel \mathit{accept_1} \\
L_2: & R \to I: & \mathit{computenonce}(N_R), \mathit{hash}(N_I, N_R) = H \parallel H, N_R \parallel \mathit{accept_2} \\
L_3: & I \to R: & \mathit{createexp}(g^a) \parallel N_I, N_R, H, g^a \parallel \mathit{hash}(N_I, N_R) = H', \\
 & & \mathit{verify}(H = H'), \mathit{accept_3} \\
L_4: & R \to I: & \ldots, \mathit{createexp}(g^b), \ldots \parallel \ldots \parallel \ldots, \mathit{accept_4}
\end{array}$$

The dangerous line is L_3; $\Delta(\mathit{accept_3})$ = expensive, because exponentiation is required during preparation in L_4. No state is stored before L_3; thus, the responder can check whether an L_3 message might be out-of-sequence only by verifying the hash of the initiator and responder nonces, N_I and N_R. The contents of L_1 and L_2, however, are enough to replay L_3 at a later time and cause the responder to exponentiate.

Eavesdropping on L_1 and L_2 and matching source and destination addresses will yield a viable (N_I, N_R, H) combination that can be used in a later L_3 message. Alternatively, the identity of one agent can be exposed to obtain a viable (N_I, N_R, H) combination, and many bogus L_3 messages can be created using forged source addresses or via a DDoS attack.

As presented, the attack is not necessarily very severe. $\phi(\mathit{eavesdrop})$ = medium, but the contents of L_1 and L_2 must be matched for a valid hash. On the other hand, the cost can be amortized over many attacks, because only one match is necessary to replay L_3 many times, and the replays are not time-sensitive. However, even without amortizing the cost of *eavesdrop*, we obtain the relation $\Delta(\mathit{accept_3})$ = expensive while $\Theta(\mathit{accept_3})$ = medium, which is unsatisfactory. It seems even more so when the medium cost is amortized over many attacks.

The responder's implementation of this protocol might defend against such an attack, but the defenses could be disabling. Because a replay of L_3 actually succeeds in passing the verification test, the responder cannot know that the L_3 instance is out of sequence without storing some state. Any state storage used to compensate for this could then be exhausted in other ways to cause denial of service. This includes attempts to log source addresses that correspond to (N_I, N_R, H) combinations.

If the responder limits the number of protocol instances initiated with a given hash, he becomes immune to constant replays of L_3; however, this could prevent a legitimate agent from communicating with the responder if the attacker's message arrives first. In this case, service could be denied to one legitimate participant without completely disabling the server.

Small changes to the protocol can exacerbate the problem. If in L_2 the responder echoes the initiator's nonce N_I in the clear, an eavesdropper would only have to catch L_2 messages to replay L_3 at a later time. Also, if the hash in L_2 is replaced with a weaker response, such as a cleartext cookie or a cleartext identity, an eavesdropper could more easily spoof L_3 messages.

Result 3 *Protocols using weak authentication that can be falsely replayed after eavesdropping are DoS-susceptible. In addition, the following defenses to the eavesdropping attack can cause a different type of DoS attack:*

- *storing state (*e.g.*, source addresses) corresponding to weak authenticators, yielding a buffer-overrun DoS-attack; and*
- *rejecting potential replays, thereby individually denying service to legitimate participants by confusing them with bogus connections.*

These protocols are also highly susceptible to DDoS attacks, because one valid weak authenticator can be replayed, with spoofed source addresses, by any number of compromised computers involved in the attack.

Some modes of IKE suffer from this problem, because, once identities are exchanged via cookies, the responder can be forced to exponentiate. An eavesdropper can take advantage of the weak protection of identities and replay the later messages to exhaust the responder's resources; alternatively, one agent that exposes his identity can obtain the required information for other computers involved in a DDoS attack.

The JFK protocol seems to be susceptible to this attack, because it uses hashing as medium-level authentication before exponentiation is performed (see Section 3.4). However, JFK enforces strict time limits on the validity of the hash authenticators, which reduces the severity of the attack. In fact, securing a timestamp in a medium-level authenticator (like a hash) and enforcing limits on message lifetimes can generally reduce susceptibility to replay attacks.

3.4 Just Fast Keying (JFK)

We now analyze an alternative to IKE, the Just Fast Keying (JFK) [1] protocol. An annotated Alice-and-Bob specification for the protocol is:

$$\begin{array}{lll}
L_1: & I \to R: & \mathit{computenonce}(N_I), \mathit{createexp}(g^a) \parallel N_I, g^a \\
 & & \parallel \mathit{verifygroup}(g^a), \mathit{accept}_1 \\
L_2: & R \to I: & \mathit{computenonce}(N_R), \mathit{sign}(g^b) = S, \mathit{hash}(g^a, g^b, N_I, N_R) = H \\
 & & \parallel N_I, N_R, [\text{group-info}], \mathrm{ID}_R, g^b, S, H \parallel \mathit{accept}_2 \\
L_3: & I \to R: & \mathit{computekey}(N_I, N_R, g^{ab}) = K, \\
 & & \mathit{sign}(N_I, N_R, g^a, g^b, \mathrm{ID}_R, \mathrm{SA}) = T, \\
 & & \mathit{encrypt}(K, \{\mathrm{ID}_I, T, \mathrm{SA}\}) = C \\
 & & \parallel N_I, N_R, g^a, g^b, H, C \parallel \mathit{hash}(g^a, g^b, N_I, N_R) = H', \\
 & & \mathit{verify}(H = H'), \mathit{computekey}(N_I, N_R, g^{ab}) = K, \mathit{decrypt}(K, C), \\
 & & \mathit{verify}C, \mathit{accept}_3
\end{array}$$

$$
\begin{array}{lll}
L_4: & R \to I: & computenonce(N'_R), sign(N_I, N'_R, g^a, g^b, \mathrm{ID}_I, \mathrm{SA}, \mathrm{SA}') = W, \\
& & encrypt(K, \{W, \mathrm{SA}'\}) = D \parallel D \\
& & \parallel decrypt(K, D), verifyD, accept_4
\end{array}
$$

The protocol begins with a form of weak authentication — the intiator supplies an identifying nonce and attempts to establish components later used to compute an encryption key. The responder separately chooses a g^b Diffie-Hellman value that is valid for a given time period; this is why the response of $g^b, sign(g^b)$ is acceptable in L_2 — no extra work is required to respond with this information. The verification of L_1 simply checks that the Diffie-Hellman value g^a would work with g^b; if not, the responder replies with a rejection message indicating which groups are acceptable for g^a. L_2 involves medium-level authentication by the responder; a hash of the weak authenticators (the nonces) and the Diffie-Hellman values is computed. Only the responder can produce this hash value H; it can serve as an authenticator, because future messages require reference to the hash in L_2. Therefore, we have that $\Delta(accept_2)$ = medium, while $\Theta(accept_1) = \Theta(accept_2)$ = cheap, because $\phi(spoofexp)$ = cheap: The attacker could easily spoof the exponential in L_1 with garbage disguised as a valid group value. However, this situation is not too severe: The initial-message attack forces no state storage on the responder, and it is state storage that normally makes the attack so severe. At worst, the responder computes a cryptographic hash at medium cost. Forcing medium cost with no state storage fits within our less-restrictive tolerance relation of Example 1.[2]

Accepting L_3 is dangerous for reasons discussed earlier. Once the medium-level authenticator is verified, decrypting the contents of the L_3 message requires exponentiation by the responder. Unfortunately, within the time period where the nonce and g^b are valid, an eavesdropper can obtain enough information to replay L_3 by obtaining the hash H from L_2. JFK has built-in protection for this attack, however. Upon responding to L_3, the L_3 message and the L_4 response are cached. Therefore, a replay of L_3 with a previously used H just causes the responder to retransmit the cached copy of L_4 without expensive operations. Thus, even though $\Delta(accept_3)$ = expensive and $\Theta(accept_3)$ = medium (eavesdropping), the responder is able to avoid exhaustion by essentially ignoring replays of L_3, never having to repeat expensive operations as a result of an attack. In addition, the cache can be cleared when the nonce and g^b expire, preventing a state-exhaustion attack. All of these mechanisms ensure that the amortized cost $\Delta(accept_3)$ is low.

If the responder begins to receive troublesome L_3 messages, the corresponding hash H can be added to a "blacklist" so that future messages with this authenticator are ignored. This is acceptable because the authenticators are

[2]It seems unreasonable to expect that we can employ some type of escalating authentication without a cost relationship like (medium, cheap) appearing in the protocol analysis. Therefore, we permit this situation in the absence of state storage. More careful attention is given to steps involving state storage or expensive actions. Recall that DoS attacks generally occur by forcing repetition of resource-consuming steps, and the medium-cost actions without state storage are often not enough to overcome a server.

valid only within a specific time interval, and the blacklist can be cleared when the values expire. A legitimate participant using this authenticator can then receive the appropriate reject messages and start the protocol again or simply wait until the values expire to negotiate a new Diffie-Hellman exchange with the responder. If the secrecy interval is set appropriately, service will be interrupted for only a short time.

Therefore, although some DoS attacks examined earlier can affect the JFK protocol, their effects cannot be amortized over a long period of time, because JFK contains several recovery mechanisms and slowly escalating authentication. In practice, we can say the following.

Result 4 *The Just Fast Keying (JFK) protocol is DoS-resistant, because its enforcement of weak-authenticator expiration times and treatment of possible replay messages (allowing amortization of the cost of an expensive verify operation) justify an attack-cost function Θ that fits within the tolerance relation $\mathcal{T}$ discussed in Example 1.*

4 Fail-Stop and Non-Interference

Fail-stop[3] [7] and *non-interference* [5] are properties used to evaluate the secrecy of a protocol. These properties are also related to denial of service, because they describe how a protocol behaves in response to interference by an intruder.

Definition 8 *A protocol is* fail-stop *if, whenever a message is interfered with (bogus, tampered with, or out-of-sequence), any accept events scheduled to follow the receipt of that message do not occur.*

The consequence of the fail-stop property is that active attacks cannot accomplish anything more than passive attacks. Because any injected or tampered-with messages can be discovered and render the protocol instance a failure, an attacker can only eavesdrop to gain information. Once a protocol is shown to be fail-stop, secrecy can be proved or disproved simply by considering passive attacks.

The relationship to denial of service is clear — a message that has been interfered with is a possible starting point for a DoS attack. Detection of such messages is an important weapon against DoS-susceptibility. Unfortunately, the resources used must be minimal enough to prevent becoming overwhelmed by the detection steps alone.

The non-interference property is defined in terms of Gorrieri's *Secure Process Algebra* (SPA) [4, 5, 6]. The SPA is a language used to express protocol actions using the notion of *security levels.* Legitimate protocol participants are placed in one security level, and potential attackers are placed in another level.

[3] *Fail-stop with respect to* Θ is the definition for a DoS-resistant protocol using Meadows's framework (Definition 6). *Fail-stop* without reference to a cost function is the original Gong-Syverson protocol property discussed in this section (Definition 8).

If a protocol satisfies non-interference, then no actions from the attacker security level can influence the progression of the protocol on the participant level. The following definition captures the major qualities of non-interference and is equivalent to protocol properties collectively referred to as non-interference in Gorrieri's papers.

Definition 9 *A protocol P satisfies* strong non-deterministic non-interference (SNNI) *iff*

$$P/H \approx_T P\backslash H$$

where H is the set of actions corresponding to enemy activity, P/H is the process in which the intruder acts indiscriminately (his actions are hidden from participant view), $P\backslash H$ is the process in which the intruder is restricted (its actions are forbidden), and $\approx_T$ means trace equivalence.

In other words, P is SNNI if and only if the sets of *traces*, or protocol executions, are the same for the participants alone and for the participants with an enemy.

Essentially, this definition means that an intruder can have no effect on a non-interference protocol. Intuitively, this is a stronger definition than fail-stop, and the following discussion makes this clear.

Theorem 1 *An SNNI protocol is fail-stop.*

Proof: Simply, a protocol that satisfies non-interference cannot be interfered with by an intruder, and so the definition of fail-stop is satisfied.

The following is a more detailed proof. Suppose not; then, if the SNNI protocol were not fail-stop, there would exist a protocol execution in which a message interfered with would not be caught, and a subsequent accept event would succeed. However, this violates the trace equivalence in the definition of SNNI. SNNI requires that enemy actions have no effect on protocol executions; if the protocol is not fail-stop, we have a protocol trace in which an action by the enemy causes an accept event to succeed, which is not allowed. This is a contradiction, and therefore, an SNNI protocol must also be fail-stop. □

To examine the relationship between fail-stop and non-interference in more detail, we first define fail-stop and non-interference in a common way. Gong and Syverson [7] and Gorrieri [5] both discuss these properties in terms of protocol execution graphs or trace graphs.

Definition 10 *Given a protocol P, let the* trace graph T of P *be defined as follows. T is a directed graph. Let the nodes of T be the different states in the execution of P, where the states represent completion of operations and knowledge of information in the protocol. Let the edges of T be the* state transitions *of P, corresponding to events or operations by a legitimate protocol agent or an intruder. T should contain both a start state and a final state that indicates successful completion of the protocol by the legitimate agents.*

Using this definition, an annotated Alice-and-Bob specification of a protocol describes a sequence of actions that are the arcs along a path p from the start state to the finish state involving transitions that are operations in $\mathcal{E}$. The operations in $\mathcal{I}$, the intruder capability, represent arcs of the graph where transitions can occur by interference from the enemy.

We now attempt to redefine fail-stop and non-interference in terms of trace graphs so that we may better obtain a relationship between the two properties.

Definition 11 *A* fail-stop *protocol is one that has a trace graph in which every arc representing an enemy action either: (1) leads to a halt state, when it interferes with a message, or (2) is a loop edge, when the action has no effect and can be ignored.*

We argue that Definition 11 is equivalent to Definition 8. Indeed, both say that the protocol can detect the influence of enemy actions that interfere with messages and that, upon such detection, the protocol does not proceed to a normal accepting state.

Definition 12 *An* SNNI *protocol is one in which any arc in its trace graph representing enemy actions is a loop edge; that is, no enemy actions can cause a transition to an accept or successful verification state in the protocol execution.*

We also argue that Defintion 12 and Definition 9 are equivalent. In fact, this equivalence is clearer than the one above: Because an SNNI protocol shares an equivalence between traces where the enemy is hidden and where the enemy is restricted, there cannot be a transition between states as a result of the enemy. Thus, in the trace graph of such a protocol, any arcs representing enemy actions must be loop edges (otherwise, they would show a transition from one state to another in the trace graph).

The proof of Theorem 1 is now obvious: An SNNI protocol (satisfying Definition 12), in its trace graph, only has loop edges for all enemy actions. Thus by Definition 11, the protocol is also fail-stop.

Gorrieri has created tools to analyze protocols for non-interference [4]. This tool can check for other protocol properties as well; in particular, there is a possibility that the tool could check for DoS-resistance using the Meadows framework.

SPA can model properties described in terms of trace graphs, and so an extension of SPA might model properties of a trace graph whose definition allows adding costs to the model. In a protocol trace graph, simply add a weight to each arc that represents the cost (from some cost set) of completing the transition with the corresponding action. The engagement cost of a state S can then be defined as the sum of the costs of defender actions along the path from the start state to S, plus the cost of any immediately following preparation actions. The attack cost of a state is the sum of the costs of the attacker used to reach S. Then, a protocol is fail-stop with respect to a tolerance relation if, for all states S, the engagement cost and attack cost lie within the tolerance relation. Recall that there are separate states for failure due to interference, and so this method also analyzes how much cost is incurred in reaching a state in which the legitmate agents are aware of attack.

Further work is necessary to determine whether Gorrieri's analysis tools can be modified to take costs into account and check states against the tolerance relation. If so, it could be used to automate protocol analysis using the Meadows framework.

It is also important to note that although non-interference is such a stringent property, it alone reveals nothing about a protocol's DoS susceptibility. Examples of SNNI protocols often use immediate strong authentication to achieve the required level of security [4, 5, 6], and we have already seen that this method is DoS-susceptible. However, just as the Meadows framework extends the original definition of fail-stop (which also did not imply anything about DoS-resistance), an extension of non-interference tools, as discussed above, may prove useful.

5 Conclusion

We studied the problem of designing protocols resistant to DoS attacks. The design goal of secrecy often conflicts with DoS-resistance because authentication methods can open avenues to memory and CPU exhaustion attacks. We examined three fragments that often appear in protocols that perform some type of authentication and analyzed them using Meadows's cost-based framework; the results indeed show that DoS attacks are possible on protocols containing these common fragments. In particular, we showed that storing state without authentication and using strong authentication inappropriately cause DoS-susceptibility. We also analyzed JFK, a protocol designed with DoS-resistance in mind, and showed how its error-handling mechanisms and cautious use of state storage and medium-level authentication remove the threat of DoS attacks in practice.

The analysis also demonstrates the usefulness of the Meadows cost-based framework. The framework is built around the notion that a protocol is a sequence of operations with cause-effect relationships: An action by one agent (a legitimate protocol participant or the attacker) usually causes a sequence of actions by another agent that incurs some cost. Protocol analysis, then, involves calculating the cost expended by the legitimate participants and the attacker to reach any point in a protocol specification. We can compare the costs incurred and determine whether an attacker can force agents to exhaust resources easily, *i.e.*, whether an attacker can perform a DoS attack by expending little cost on his own. Because the framework requires an assignment of cost to every operation in the protocol specification, we expose all trouble spots where agents are forced to use resources, and can then consider what an attacker could do to reach those points in the protocol.

Finally, we studied the relationship between *fail-stop* and *non-interference*, protocol properties that seem relevant to DoS-susceptibility and secrecy. We showed that SNNI is a stringent property that implies that protocol messages cannot be interfered with, thus implying that an SNNI protocol is also fail-stop.

However, both of these secrecy properties do not automatically assert anything about a protocol's DoS-susceptibility. We discussed a direction for future work that involves extending existing tools that check for non-interference properties to include costs, much like the way that Meadows's framework extends the original definition of fail-stop. If successful, these tools could automate checking DoS-resistance of protocols using the Meadows framework.

References

[1] W. Aiello, S. M. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A. D. Keromytis, and O. Reingold. Efficient, DoS-resistant, secure key exchange for internet protocols. Unpublished draft including specification of the Just Fast Keying (JFK) protocol.

[2] Carnegie Mellon Software Engineering Institute. TCP SYN flooding and IP spoofing attacks. CERT Advisory CA-1996-21, CERT Coordination Center, 1996.

[3] N. Doraswamy and D. Harkins. *IPSec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks.* Prentice Hall, Upper Saddle River, NJ, 1999.

[4] A. Durante, R. Focardi, and R. Gorrieri. CVS: A compiler for the analysis of cryptographic protocols. In *PCSFW: Proceedings of The 12th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1999.

[5] R. Focardi, R. Gorrieri, and F. Martinelli. Secrecy in security protocols as non interference. In S. Schneider and P. Ryan, editors, *DERA/RHUL Workshop on Secure Architectures and Information Flow*, volume 32. Elsevier, 1999.

[6] R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. In *Automata, Languages and Programming*, pages 354–372, 2000.

[7] L. Gong and P. Syverson. Fail-stop protocols: An approach to designing secure protocols. In *Proceedings of the 5th International Working Conference on Dependable Computing for Critical Applications (DCCA-5)*, pages 44–55, 1995.

[8] D. Harkins and D. Carrel. The internet key exchange (IKE). Request for Comments (RFC) 2409, Internet Engineering Task Force, 1998.

[9] C. Meadows. A cost-based framework for analysis of denial of service in networks. *Journal of Computer Security*, 9(1/2):143–164, 2001.