Probabilistic model checking for the quantification of DoS security threats

Stylianos Basagiannis *Panagiotis Katsaros* *Andrew Pombortsis*

Nikolaos Alexiou

Department of Informatics

Aristotle University of Thessaloniki

54124 Thessaloniki, Greece

tel.: +30-2310-998532, fax: +30-2310-998419

{basags, katsaros, apombo, nalexiou}@csd.auth.gr

Abstract

Secure authentication features of communication and electronic commerce protocols involve computationally expensive and memory intensive cryptographic operations that have the potential to be turned into denial-of-service (DoS) exploits. Recent proposals attempt to improve *DoS resistance* by implementing a trade-off between the resources required for the potential victim(s) with the resources used by a prospective attacker. Such improvements have been proposed for the Internet Key Exchange (IKE), the Just Fast Keying (JFK) key agreement protocol and the Secure Sockets Layer (SSL/TLS) protocol. In present article, we introduce probabilistic model checking as an efficient tool-assisted approach for *systematically quantifying DoS security threats*. We model a security protocol with a fixed network topology using probabilistic specifications for the protocol participants. We attach into the protocol model, a probabilistic attacker model which performs DoS related actions with assigned cost values. The costs for the protocol participants and the attacker reflect the level of some resource expenditure (memory, processing capacity or communication bandwidth) for the associated actions. From the developed model we obtain a Discrete Time Markov Chain (DTMC) via property preserving discrete-time semantics. The DTMC model is verified using the PRISM model checker that produces probabilistic estimates for the analyzed DoS threat. In this way, it is possible to evaluate the level of resource expenditure for the attacker, beyond which the likelihood of widespread attack is reduced and subsequently to compare alternative design considerations for optimal resistance to the analyzed DoS threat. Our approach is validated through the analysis of the Host Identity Protocol (HIP). The HIP base-exchange is seen as a cryptographic key-exchange protocol with special features related to DoS protection. We analyze a serious DoS threat, for which we provide probabilistic estimates, as well as results for the associated attacker and participants' costs.

KEYWORDS: Denial of Service, Secure Authentication, Probabilistic Model Checking

1. Introduction

Formal techniques for verifying the absence of secrecy and authentication failures in cryptographic protocols have been effective in discovering design flaws that allow a malicious intruder to subvert a protocol's guarantees. All these approaches adopt the basic assumptions of a general intruder model introduced by Dolev and Yao in [9]. These assumptions are: (i) The encryption method used is unbreakable; (ii) The intruder can prevent any message from reaching its destination and (iii) The intruder can create messages of his own. However, an attacker with the mentioned abilities can also subvert the *availability* of the protocol participants and in the related bibliography there are only a few recent works that address the quantitative analysis of availability threats and the alternative countermeasures.

Our proposal introduces a probabilistic attacker model with assigned cost values that reflect the level of some resource expenditure (memory, processing capacity or bandwidth) for his actions. The model combines *attack tactics* selected from the formalized open-ended intruder model introduced by us in [5]. The analyzed DoS threat is expressed as a *probabilistic reachability property* that is automatically verified (according to [17] and [18]) with respect to an appropriate Discrete Time Markov Chain (DTMC) representing the protocol participants and attacker models. The overall analysis takes place in a probabilistic model checking toolset called PRISM [22]. This improves the usability of the analysis by the protocol designers and automates the generation of sensitivity results. It is thus possible to compare implementations with alternative parameter choices, for optimal resistance to the analyzed threat.

Our approach is described in terms of the performed analysis for the Host Identity Protocol (HIP) base-exchange. HIP aims to provide a "secure" multiaddressing mechanism that separates the two conflicting roles of host identifiers and network locations that IP addresses play in current Internet. The assigned costs quantifying the protocol's processing workload are based on related HIP performance studies [13]. We realized that an attacker model embedding three basic attack tactics that successfully represent the combined effects of *N zombie participants* breaks the utilized DoS resistance mechanism. Appropriate queries expressed in Probabilistic Computation Tree Logic (PCTL) provide illuminating probabilistic estimates together with the attacker and victim's costs for the analyzed DoS threat. Our analysis concludes with a sensitivity study of the obtained estimates with respect to the used model parameters.

In section 2 we review the few works found in related bibliography, in order to point out the differences with the proposed analysis approach. Section 3 provides a brief introduction to probabilistic model checking and defines the *DoS resistance property*, in terms of an appropriate probabilistic attacker model. Section 4 presents the PRISM model for the HIP base-exchange and comments on the results of the performed PCTL queries. We conclude with a summary of the overall analysis approach and a comment on its usability and its potential impact.

2. Related work

Design of protocols that manage the available resources effectively in the presence of DoS attacks is a complex task. Interesting techniques that reduce memory storage and processing costs for the responder in a protocol session have been introduced in [3] and [6]. The *client puzzles* that were first proposed in [4] introduce time consuming and verifiable proofs of work that artificially increase the computational workload of initiators in protocol

sessions. This strategy aims to assure that initiators have expended a predetermined level of computational effort prior to a responder committing resources, but as we will see in the case of the HIP base-exchange it has the potential to introduce new DoS exploits. Other techniques like the *cookies* [15] are used to weakly assure the origin of requests, before applying stronger cryptographic operations that implement the protocol's service. However, a gradually strengthening authentication can be successful only when its design is based on a careful analysis of the server resource usage.

The importance of enabling availability analysis for a given cryptographic protocol was first shown by Meadows in [20]. In that work the author examines DoS in the context of the resource intensive task of authentication and develops a framework for weighting the cost to the defender against the cost to the attacker. The paper concludes with descriptions of potential integration scenarios of the proposed framework into the automated security analysis tools that were available at that time. In [26] the authors exploit Meadows' framework to analyze the Just Fast Keying (JFK) protocol [2], in order to demonstrate its DoS prevention capabilities. Recently, the approach of [20] formed the basis for the analysis framework of [27] that according to the authors provides a more accurate representation of computational cost. However, quantitative evaluation takes place by simulation of the developed Timed Colored Petri Net model, without having exploited the formal analysis capabilities of the used toolset.

An interesting stochastic modeling approach for quantifying the availability of software systems under DoS threats is the one introduced in [19]. In that work, the authors formulate the analyzed system in terms of an appropriate semi-Markov process (SMP). Starting with the SMP model it is then possible to derive the embedded DTMC that involves only the considered state transition probabilities. After having computed the steady-state DTMC probabilities, the assumed sojourn time distributions for the model's states are used to

compute the SMP's steady-state probabilities. This makes possible to calculate the system's availability and subsequently perform parametric sensitivity analysis in order to examine the sensitivity of the computed availability. The overall approach requires stochastic modeling and analysis skills, since it is not carried out within an automated analysis tool like PRISM. Furthermore, the performed system-level analysis does not take into account the resource expenditure for the considered states and thus it is not possible to evaluate the message processing costs for DoS threats upon a security protocol model.

The most closely related work found in the bibliography is the one published in [1]. In that work, the authors specify in *probabilistic rewriting logic* a DoS resistant 3-way handshaking in the Transmission Control Protocol (TCP). Similarly to our approach, the developed specification includes a number of honest participants and a simple attacker, flooding the protocol model with spurious requests at a specific rate (parameter of an exponential distribution). In the VESTA toolset [24], the developed algebraic specification generates a timed probabilistic model, which is then analyzed by Monte Carlo simulation using a sequence of interrelated statistical hypothesis tests. In this way the analyst checks on the generated sample if the quantitative property of interest is satisfied and this reflects the so-called *statistical model checking* approach. Compared to the probabilistic model checking analysis that is proposed in our work, this approach *does not produce the same accurate results* [25]. Moreover, the aforementioned analysis does not take into account message processing costs as we do and for this reason it is not possible to weight the cost to the honest participants against the cost to the attacker. For the resource intensive authentication features of modern security protocols this may be a significant analysis restriction.

3. Security protocol analysis with probabilistic model checking

Security protocols exhibit varying and unpredictable patterns in their usage by the honest or potentially malicious protocol participants. This fact makes probabilistic model checking a promising approach towards the quantification of security threats. The whole analysis is based on labeling transitions between model states with information about the likelihood that they will occur ([17], [18]).

3.1 Probabilistic model checking basic concepts

In the PRISM language, a probabilistic model is defined as a set of m modules (reactive modules) $M_1, \ldots, M_m$. Each module M_i is a pair (Var_i, C_i) with Var_i a set of integer-valued local variables with finite range and C_i a set of commands. We denote by Var the set of all local variables in the model, i.e. $Var = \bigcup_{i=1}^{m} Var_i$. Each variable $v \in Var$ has an initial value $\bar{v}$.

Each command $c \in C_i$ takes the form $(g, (\lambda_1, u_1), \ldots, (\lambda_{nc}, u_{nc}))$, comprising a guard g and a set of pairs (λ_j, u_j) where $\lambda_j \in R > 0$ and u_j is an update for each $1 \leq j \leq n_c$. A guard g is a predicate over the set of all local variables Var and each update u_j corresponds to one possible transition of module M_i. If Var_i contains n_i local variables $v_1, \ldots, v_{ni}$, then an update takes the form $(v_1' = \mathrm{expr}_1) \wedge \ldots \wedge (v_{n_i}' = \mathrm{expr}_{n_i})$ where each $expr_j$ is an expression in terms of the variables in Var. When in an update the values of some variables in Var_i remain unchanged, the model description may omit this information. In a DTMC specification, the values λ_j determine the probability of the corresponding transition and for this reason $\lambda_j \in (0, 1]$ for $1 \leq j \leq n_c$ and $\sum_{j=1}^{n_c} \lambda_j = 1$. Also, in DTMCs terminating states are modeled by a single transition going back to the same state with probability 1.

The DTMC model corresponding to a PRISM language description is constructed as the parallel composition of its modules by computing the reachable state space of the model

and discarding any unreachable states ([17], [18]). In every global state, there is a set of commands (belonging to any of the modules) which are enabled. The choice between which command is performed is probabilistic, with *each enabled command selected with equal probability*.

Probabilistic model checking is based on a *probability measure definition* over the set $Path_s$ of all infinite paths starting in state s, for all states $s \in S$ of the DTMC state space. We omit the details of this construction (refer to [16] and [17]) that makes it possible to determine the probability that paths in a DTMC are taken. However, this construction gives us the ability to quantify the probability that the DTMC behaves in a specified fashion, by identifying the set of paths which satisfy the given property specification and assuming that this probability is measurable using the aforementioned probability measure.

Property specifications for DTMCs are expressed in PCTL [12], which is derived from the branching tree logic CTL (Computation Tree Temporal Logic) by exchanging the path quantifiers A (for all) and E (exist) with a probabilistic operator P. This operator defines an upper or a lower bound on the probability of the model evolution. As an example the formula $P_{\geq p}[\varphi]$ is true at a given time, if the probability that the future evolution of the system satisfies φ is at least p. In a similar manner, the formula $P_{\leq p}[\varphi]$ is true if the probability that the system fulfills φ is less than or equal to p. Apart from quantitative assertions, in PRISM we can also express properties which evaluate to a numerical value. These properties are specified in the form:

$$P_{=?}[\varphi]$$

PCTL property specifications are checked by applying appropriate model checking algorithms on the model, by induction over their syntax. The underlying computation in PRISM involves a combination of:

- Graph-theoretical algorithms, for reachability analysis and qualitative probabilistic model checking;
- Numerical computation (iterative solvers), for quantitative probabilistic model checking that in the case of a DTMC implies the solution of linear equation systems.

In our analysis approach we exploit the notion of *experiments*, which is a way for automating multiple instances of model checking. This allows us to easily obtain the outcome of one or more properties as functions of model and property parameters. The results may be obtained in graphs within the PRISM model checking environment or alternatively in an external spreadsheet application for further study.

Finally, the DoS resistance property, as it is defined in next section, implies the need to exploit the functions provided in PRISM for computing *state rewards*. These rewards represent consumption of an exhaustible resource (*cost*) that depending on the modeled DoS threat it may be communication capacity (bandwidth), memory or processing capacity. State rewards (that are also called *cumulative rewards*) are defined by means of a reward function $\underline{\rho}$: $S \rightarrow R \geq 0$ such that a reward of $\underline{\rho}(s)$ is incurred, if the DTMC is in state s for 1 time-step.

In PRISM, the logic PCTL is extended to allow specification of reward properties. Thus, for some $r \in R \geq 0$ and $k \in N$ a state s satisfies $R_{\leq r}[C^{\leq k}]$ if, from state s, the expected reward *cumulated* after k time-steps is less than or equal to r. As a last example, the formula $R_{\geq r}[\mathrm{F}\Phi]$ is true, if from state s the expected reward cumulated before a state satisfying formula Φ is reached will be at least r.

3.2 The probabilistic attacker model

It is generally adopted [21] that an effective attacker model for the analysis of DoS threats can be weaker than the typical Dolev - Yao attacker, which is used in verifying typical authentication goals. This is justified by the fact that the attacks found to subvert protocols' authentication goals do not necessarily arise in the case of DoS, although they are not ruled out entirely. The problem of determining the right set of abilities for the attacker model is connected to the understanding of the nature of the *DoS resistance property*. According to [8] "a DoS attack is characterized by an explicit attempt by the attacker(s) to prevent legitimate users of a service from using that service". Thus, the result of a DoS attack can be regarded as the absence of some sort of *progress properties* expected from a DoS resistant protocol. In [10], the author characterizes these properties as *"self-controlled" liveness properties* and realizes that they form a particular subset of liveness properties, which the attacker cannot control.

The strong assumptions of the Dolev - Yao attacker have been effective in the analysis of security guarantees that are formulated as *safety properties* (secrecy and authentication), but they are not entirely suitable for the analysis of security guarantees like DoS resistance that involves *liveness* [7]. A typical Dolev - Yao attacker has full control over the communication channels between the protocol participants and it is treated as a nondeterministic process that may attempt *any possible attack*. A protocol is considered secure if no possible interleaving of actions results in a security breach. However, the presence of non-determinism means that certain liveness properties cannot be established unless *fairness* is assumed. From this perspective, as far as fairness can be viewed as an abstraction of a probabilistic behaviour, it seems more natural to invest in a probabilistic model checking approach and to not adopt fairness assumptions, which are not valid for all attacker abilities considered in typical Dolev - Yao style analyses.

Some other facts that make the use of probabilistic model checking a preferable choice are:

- The requirement to model the ability of an attacker to send randomly chosen messages or to model some sophisticated (yet probabilistic polynomial-time) computation to derive an attack from eavesdropped messages.
- The need to model the probabilistic selection of implementation parameters, whose values affect the protocol's security.
- The need to capture the DoS faithfully, i.e. in terms of relative probabilities of certain observations by the attacker that depend solely on potentially probabilistic behaviours of the protocol participants (like for example the probability for a participant to resubmit a service request that was previously dropped from the server's admission queue).

A DoS attacker uses a fixed number *N* of compromised machines that are commonly called *zombie machines*, because they are identical to the machines used by honest protocol participants. The zombie participants create bogus protocol messages capable of tricking some honest protocol participant into fully expediting resources, before determining that the opened protocol sessions are bogus. Alternatively, instead of considering *N* zombie machines we may consider a powerful attacker *with identity spoofing abilities* that allow him to generate the combined effects of *N* zombie participants.

Our proposal for the DoS attacker model uses the open-ended attack tactics base we proposed in [5], from where the analyst selects the right set of abilities for his DoS problem. The selected attack actions are combined into a single PRISM module and the analyst assigns to the performed operations *cost values that depend on the operation's resource expenditure, as well as on some resource constraint for the attacker*. A similar cost assignment is performed for the honest protocol participants. In all cases, the assigned

values refer to *the same exhaustible resource*, which can be either, *memory*, *processing power* or *communication bandwidth*.

Table 1 Resource costs for a DoS attacker with N zombie participants and resource costs for the honest protocol participants

Variable	Description
N	The number of zombie machines.
$\#Ses$	The number of protocol sessions allowed for simultaneous processing by a protocol server.
u_s/v	The processing cost for a single protocol session in an honest participant having the capacity to process v instructions per second (u_s is the number of instructions required for the whole session).
k_s/m	The processing cost for creating a fake protocol session in a zombie machine having the capacity to process m instructions per second (k_s is the number of instructions required for the fake session).
x	Communication bandwidth for a zombie machine in bits per second.
X	Communication bandwidth for a honest server in bits per second.

The *severity* of all resource exhaustion possibilities can be first assessed by computing all *resource asymmetries* between the attacker – with the considered resource constraints – and the honest protocol participants. For a protocol message that requires u instructions of computation by some protocol participant having the capacity to process v instructions per second, the processing cost is u/v seconds of computation. On the other hand, if the attacker processes m instructions per second and an assigned computation takes k instructions, then the considered cost is k/m seconds of computation. We assume *the same resource expenditure for the attacker and the honest participants for the same operations* and we calculate the corresponding costs as described. Table 1 summarizes the notation used in the following calculations for the resource costs and the resource asymmetries in a typical protocol case.

For some N-zombie attacker if we assume that each zombie is connected to the network via a link with bandwidth x bits per second and a server participant is connected via a link with bandwidth X bits per second, then the zombies alone can saturate the server's bandwidth when $Nx \geq X$. Regarding the server's processing power, this can be saturated when

$$N \cdot k_s/m \geq (\#Ses) \cdot u_s/v$$

where k_s is the total number of instructions in a single protocol session for the attacker, u_s the number of instructions in a protocol session and *#Ses* the number of sessions allowed for simultaneous processing by the server. The problem of determining whether *processing DoS threats* dominate *bandwidth DoS threats* is a matter of comparing whether the server/zombie asymmetry is greater for bandwidth than for processing, i.e. whether

$$\frac{(\#Ses) \cdot u_s / v}{k_s / m} < \frac{X}{x}$$

Our approach is then applied by taking into account the cost values for the most serious resource exhaustion DoS threat.

For a given DoS threat, the *DoS resistance property is quantified by the probability for an attacker representing the modeled threat to eventually prevent - with disproportionaly low cost - honest participants from using the protocol's services*. We are interested to discover protocol designs and implementation parameters that minimize this probability and for this reason we try to make sure that every time a honest participant takes part in some action that requires the use of significant amount of resources, the attacker cannot fraudulently cause him to reach that step without spending a significant amount of its own resources.

We already noted that DoS protection is usually based on an appropriate *cookie-based* [15] or *client puzzle* mechanism [4], where a protocol participant passes a "cookie" (e.g. an "unforgeable" keyed hash value of the information identifying the connection) or a puzzle to another participant, in order to establish mutual trust, possibly in the form of some shared-secret. The idea is that the responder should remain stateless (protection against memory exhaustion) and refuse to perform expensive cryptographic operations (protection against processing power exhaustion), until it has verified the honesty of the initiator. In this setting, there are three key strategies by which an attacker can implement a DoS threat:

- *Counterfeiting*: The attacker sends *invalid* cookies, puzzles or puzzle solutions.
- *Time Shifting*: The attacker is prepared for an attack by computing fake shared secrets (either by solving puzzles or manipulating cookies), in order to expend them in a massive DoS attack.
- *Message replays*: The attacker may send the same *valid* cookie or puzzle solution many times.

The aforementioned strategies assume that the attacker model performs three basic operations, i.e. *message interception*, *message projection* and *message concatenation*, mentioned in decreasing order of processing demands. The analyst implements a DoS threat by selecting the right set of *attack tactics* from the ones formalized by us in [5] and composes them into a single PRISM module with appropriate cost values for the performed operations.

3.3 Attack tactics for DoS attacker models

An *atomic message* may come from one of the sets:

- *Keys,* with members that represent the keys used to encrypt messages, such that every key $k \in Keys$ has an inverse $k^{-1} \in Keys$. For symmetric cryptography the decryption key is the same as the encryption key, i.e. $k = k^{-1}$.
- *Agents,* with members that represent the names of the honest protocol participants.
- *Nonces*, which is an infinite set of randomly generated numbers. Members of *Nonces* are used as timestamps, i.e. any message containing one of them can be assumed having been generated after the nonce itself was generated.
- *Data*, with members that represent the plaintext strings exchanged between the protocol's participants.

- *Bogus_data,* with members that represent data created by the attacker for counterfeiting intercepted messages.

We denote by *At* the *attacker* (*At* ∉ *Agents*). Also, we define the binary relation,

$$is\_key\_of = \{ \ (k, id): k \in Keys, id \in Agents \cup \{At\},$$
$$\text{“}key\ k\ is\ used\ by\ the\ participant\ id\text{”}\}$$

such that $|is\_key\_of\ (k)| = 1$ in the case of public key cryptography or $|is\_key\_of\ (k)| = 2$ in the case of symmetric cryptography.

The set *Msgs* of *exchanged messages* is defined inductively over the *disjoint union*

$$AMsgs = Keys \cup Agents \cup \{At\} \cup Nonces \cup Data \cup Bogus\_data$$

that represents the set of atomic messages ($Set_i \cap Set_j = \varnothing$ for any two Set_i, Set_j of the unified sets). More precisely:

- If $\alpha \in AMsgs$ then $\alpha \in Msgs$.
- If $msg_x \in Msgs$ and $msg_y \in Msgs$ then $msg_x \cdot msg_y \in Msgs$, where $\cdot$ represents message concatenation.
- If $msg \in Msgs$ and $k \in Keys$ then $\{msg\}_k \in Msgs$.

Each $\mathtt{ag} \in Agents$ may attempt to execute the protocol for a bounded number of times say $\#Ses_{\mathtt{ag}}$ and each such attempt is a separate *protocol session noSes*, such that $1 \le noSes \le \#Ses_{\mathtt{ag}}$. In a protocol session, ag plays either the role of the *initiator* or the *responder*. We denote by $sent_n^{\mathtt{ag}_{noSes}}$ the finite-length concatenation sequence of messages sent by $\mathtt{ag} \in Agents$ in the course of session *noSes*:

$$sent_n^{\mathtt{ag}_{noSes}} = (sent_{n-1}^{\mathtt{ag}_{noSes}} \cdot msg_n)$$

with the first term equal to the *null sequence* that is, $sent_0^{\mathtt{ag}_{noSes}} = ()$. The sequence $sent_n^{\mathtt{ag}_{noSes}}$ represents participant's ag *history* for session *noSes*, after having sent msg_n. We denote by $rcvd_n^{\mathtt{ag}_{noSes}}$ the finite-length concatenation sequence of messages received by ag in the

course of session *noSes*. In a given time instant the acquired *participant's knowledge* for the ongoing protocol execution is given as

$$\mathtt{ag}_{\text{knowledge}} = \bigcup_{\mathtt{ag}_j} \{rcvd^{ag_j}_{\max(i)}\} \cup \mathtt{ag}_{in\_knowledge},$$

for all $1 \le j \le \#Ses_{\mathtt{ag}}$, where $\mathtt{ag}_{in\_knowledge}$ represents the *initial knowledge base* of $\mathtt{ag}$ (keys, agent identities and so on) and $i > 0$ represent the terms of the received message concatenation sequences. A protocol session for a honest participant $\mathtt{ag} \in Agents$ is defined formally as a 5-tuple $\langle \mathtt{ag}, j, \mathtt{ag}_{knowledge}, \mathrm{ag}^{j}_{history}, PR \rangle$, where $1 \le j \le \#Ses_{\mathtt{ag}}$ and PR is a *process description* given as a sequence of *actions* that represent the protocol definition for the considered role. We use the actions **send**[1] and **receive**[2] for sending and receiving messages to/from other protocol's participants. The assumptions mentioned in section 1 for the general Dolev - Yao attacker model imply that in a given time instant the acquired *attacker's knowledge* for the ongoing protocol executions is given as

$$At_{knowledge} = \bigcup_{\mathrm{ag}_j} \{sent^{ag_j}_{\max(i)}\} \cup At_{in\_knowledge},$$

for all $1 \le j \le \#Ses_{\mathtt{ag}}$, $\mathtt{ag} \in Agents \cup \{At\}$, where $At_{in\_knowledge}$ represents the initial attacker's knowledge base and $i \ge 1$ represent the terms of the eavesdropped message concatenation sequences.

The *protocol model* is given as the parallel composition of the models for all protocol sessions, including the attacker model, whose behavior depends on the defined *attack tactics*. DoS attack related actions are selected in a probabilistic manner and each possible path of the model corresponds to a non-empty alternating sequence of *states* and actions:

$$\tau = s_0\, \alpha_1\, s_1\, \alpha_2 \ldots s_n, \text{ for some } n \in N$$

such that $s_{i-1} \xrightarrow{a_i} s_i$ for $0 < i \le n$ and for the transition relation $\rightarrow$ defined as

$$\rightarrow \subseteq S \times PS \times A \times Msgs \times S$$

[1] Action **send**(*ag*, *v*, *msg*) expresses the event whereby *ag* sends *msg* to *v*
[2] Action **receive**(*v*, *u*, *msg*) expresses the event whereby *v* receives *msg* from *u*

where S is the set of states, PS is the set of protocol sessions and A is the set of action names.

In a DoS attacker model that combines message counterfeiting and message replays the DoS attack may take place anytime after the occurrence of some action **send**(ag, v, msg) with ag, $v \in Agents$ and some $msg \in Msgs$ that At alters with negligible processing cost, if msg is not encrypted. Alternatively, the DoS attack may also take place with substantial counterfeiting cost anytime after the occurrence of some action **send**(ag, v, $\{msg\}_k$) such that $At \notin is\_key\_of(k) \wedge k^{-1} \in At_{knowledge}$. The first mentioned action results in a state, where

$$exists(msg,\ sent^{ag_j}_{\max(i)})^3 = \texttt{true}$$

for some $1 \le j \le \#Ses_{ag}$, with $i \ge 1$ representing the terms of the eavesdropped message concatenation sequence and respectively, the second mentioned action results in a state, where

$$exists(\{msg\}_k,\ sent^{ag_j}_{\max(i)}) = \texttt{true}$$

The attacker model (At) manipulates message msg by applying a protocol-specific sequence of message projection and message concatenation operations, which in effect replace some part of msg with $bg\_data \in Bogus\_data$, such that for the altered message msg' we have

$$exists(bg\_data, msg') = \texttt{true}$$

This message counterfeiting is denoted by

$$alter(msg, bg\_data) \Rightarrow msg'$$

or respectively

$$alter(\{msg\}_k, bg\_data) \Rightarrow \{msg'\}_k$$

and is completed by a straight message replay corresponding to the action **send** (At, v, msg') or the action **send** (At, v, $\{msg'\}_k$) in the second case.

[3] Boolean predicate $exists(msg, str)$ is true if the message $msg \in Msgs$ appears in string str

N zombie participants trying to trap the victim $v \in Agents$ in some form of resource exhaustion are represented by N distinct pairs of message counterfeiting and message replay actions. Every single pair of these two actions uses a different $bg\_data \in Bogus\_data$, which may be chosen randomly or in the case of a time shifting attack, it has been pre-computed by the attacker.

Next we recapitulate the sequence of actions in a DoS resource exhaustion attack against victim $v \in Agents$, which is based on counterfeiting some message msg:

in $s \in S$: $exists(msg, At_{knowledge}) = \texttt{true}$

do for N times

$\exists\ bg\_data \in Bogus\_data$: $alter(msg, bg\_data) \Rightarrow msg'$

$Bogus\_data = Bogus\_data - \{\ bg\_data\ \}$

$msg = msg'$

send (At, v, msg')

end_do

When a DoS attack is based on counterfeiting some message $\{msg\}_k$ for $k \in Keys$ such that $At \notin is\_key\_of\ (k) \wedge k^{-1} \in At_{knowledge}$ the given DoS attack definition is adjusted appropriately. In order to break a protocol's secrecy "guarantee" a DoS attacker can combine the described sequence of actions with an appropriate attack tactic from the ones defined in [5].

The DoS attack succeeds if there is a reachable $s' \in S$ where

for each counterfeited $msg' \in \bigcup_{At_j} \{sent^{At_j}_{\max(i)}\}$ with $1 \le j \le \#Ses_{At}$

$\exists\ m,\ 1 \le m \le \#Ses_v$ such that $exists(msg', rcvd^{v_m}_{\max(i)}) = \texttt{true}$

and s' represents a resource exhaustion state for participant $v \in Agents$, i.e. a state in which some implementation parameter or resource constraint hinders v from executing **receive** actions for all protocol messages that are send by $ag \in Agents - \{v\}$. If the probability to

reach *s´* is significant and the cumulative cost to the Attacker is disproportionally low when compared with the cumulative cost to *v*, then the DoS resistance property – as we define it in section 3.2 – is violated.

Figure 1 provides a high level view of the analyzed DoS threat upon the HIP base-exchange. Contrary to related analyses of HIP [27], which focus on the DoS resistance of the responder in a protocol session, our work examines a *serious counterfeiting DoS threat against the initiator*. The attacker (*At*) intercepts the message traffic between the initiator (*I*) and the responder (*R*) and alters the puzzle contained in msg_R by simple message concatenations, in order to create *N* zombie messages that are subsequently sent to the initiator. The attack tactics used in the attacker model are the message integrity violation combined with multiple straight message replays.

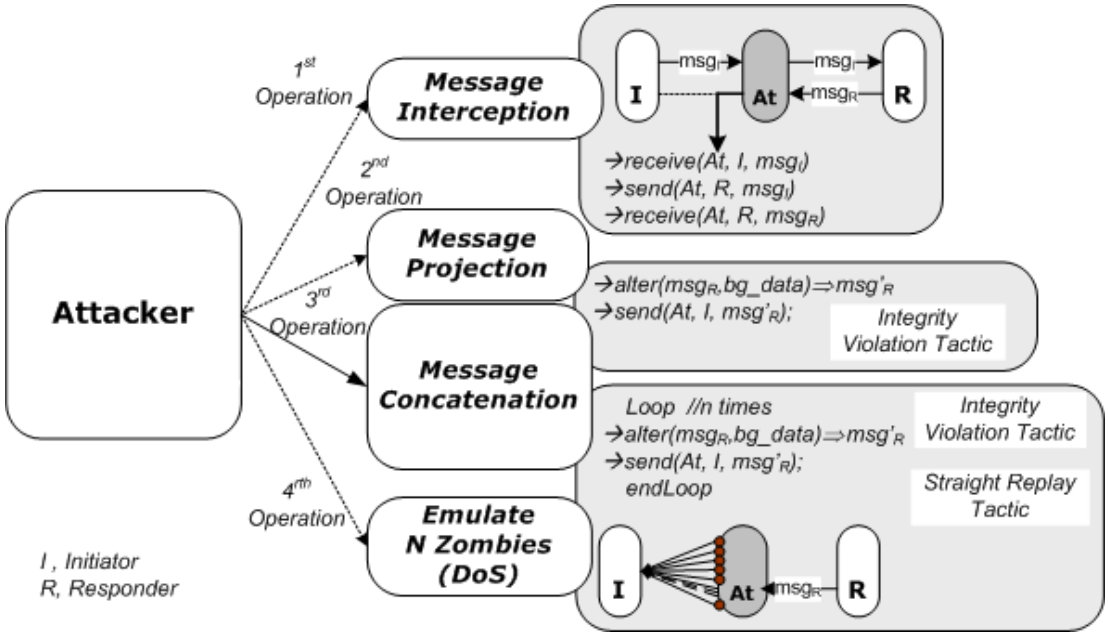

Figure 1. A DoS threat with message counterfeiting for N zombie participants

4. Analysis of a DoS threat against the HIP base-exchange protocol

4.1 Introduction to the HIP base-exchange

The main goal of HIP [14] is the separation of host identifiers from locations in the IPv4 and IPv6 Internet. In a typical TCP/IP stack, IP addresses serve both roles and this fact introduces certain limitations regarding mobility and multi-homing. HIP is a new protocol layer that is introduced between the network and transport layers, in order to map the host identifiers to network addresses and vice versa.

However, HIP also plays the role of a *security protocol* that defines host identifiers for naming the communication endpoints and performs authentication and IPsec *security associations* between them. The HIP base-exchange is built around a classic authenticated Diffie - Hellman key exchange, in an attempt to establish session keys between the communication endpoints. If the HIP base-exchange succeeds, all subsequent packets are protected by an Encapsulating Security Payload (ESP) header. Depending on the options selected at the time of security association establishment the used ESP header may provide confidentiality, data origin authentication, connectionless integrity, an anti-replay service and limited traffic flow confidentiality.

In HIP, the *host identity* (HI) of the protocol participants plays the role of a public key: this identifier can be used to verify signatures without access to certificates or a public-key infrastructure. It is usually represented by the *host identity tag* (HIT), which is a 128-bit hash of the HI. As shown in Figure 2, the HIP base-exchange includes four messages that are supposed to provide a certain degree of DoS protection. The Initiator first sends the message I1 with the HIT_I and the HIT_R tags, to the Responder. We note that all messages contain the Initiator and Responder identity tags (HIT_I, HIT_R) in the header.

HIT_I	*I* identity tag
HIT_R	*R* identity tag
g^R	pre-computed part of R1
sig_R	signature of *R*
sig_I	signature of *I*
C	puzzle nonce
k	puzzle difficulty
J	puzzle solution
LSB_k	returns the k least significant bits
K_e, K_s	generated Diffie-Hellman keys
E_x	message x encrypted with K_e
HK_s	cryptographic hash with key K_s
HMAC	HMAC based message authentication code computed using K_s

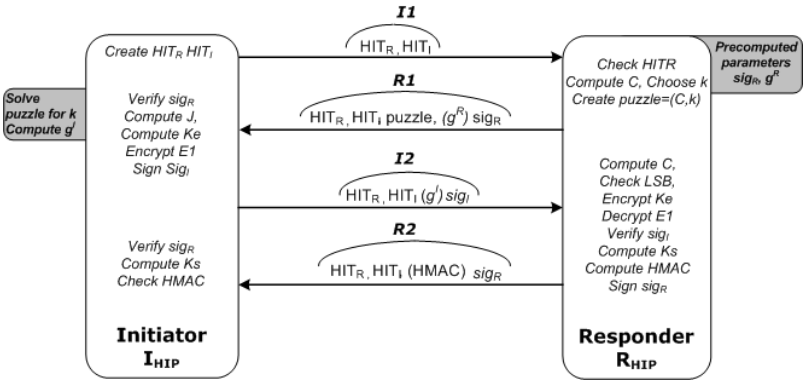

Figure 2. The HIP Base Exchange

Message R1 is partially pre-computed by the Responder, even before the receipt of I1. The pre-computed part (g^R) includes (i) the HIT_R, (ii) the Responder's Diffie-Hellman key, (iii) the Responder HI, (iv) the proposed cryptographic algorithms for the next steps of the base-exchange, (v) the proposed ESP transforms and (vi) an echo request field. The last mentioned field is used to store some data and relieves the Responder of maintaining protocol state data at this step (protection against memory exhaustion). The Responder signs the pre-computed part of R1 with sig_R. All other parts of R1, i.e. the cryptographic puzzle and the HIT_I fields are populated after receiving an I1 and *they are not protected by the signature* sig_R. A host may receive more than one R1 messages, either due to having sent multiple I1s or due to a replay of an old R1.

The used puzzle has three components: the puzzle nonce *C*, the difficulty level *k* and the corresponding solution *J*. The puzzle solution is verified as follows: we compute the SHA-1 hash of the concatenation of *C*, HIT_I, HIT_R and *J* and then we check that the *k* low-order bits of the hash are all zeros.

$$LSB_k(\text{SHA-1}(C \mid HIT_I \mid HIT_R \mid J), k) == 0$$

While the Initiator performs a brute-force search for J that takes $O(2^k)$ trials, the Responder verifies the solution by computing a single hash (protection against processing power exhaustion).

On receiving R1 the Initiator checks that it has sent a corresponding I1 and verifies the signature using the Responder HI. Then, it solves the puzzle and creates the message I2 that includes HIT_I, HIT_R and a signed part g^I. The signed part contains (i) the puzzle and its solution, (ii) the Initiator's Diffie-Hellman key K_e, (iii) the HIP and ESP transforms proposed by the Initiator, (iv) the Initiator HI (public key) encrypted using K_e, which is shown as E_1, (v) the HIP and ESP transforms proposed by the Initiator, (vi) a security parameter index for the Responder-to-Initiator security associations and (vii) the echo response generated for the received echo request.

On receiving I2, the Responder verifies the puzzle solution, decrypts E_1 that contains the Initiator HI, verifies the signature on I2 and computes the session key K_s. For the Initiator, the HIP base-exchange is concluded by the receipt of R2, which allows verifying the HMAC (hash-based *message authentication code* that is computed using the session key K_s) and the signature. The end of protocol session is signaled by a CLOSE/ CLOSE_ACK handshake that transmits a signed verifiable HMAC to both sides.

If a host receives an unexpected message, it must drop the message. Also, an HIP implementation is free to drop an association at any time, based on its own policy. If a host decides to drop an association, it deletes the corresponding HIP state, including the keying material. Our probabilistic model checking approach allows validating potential DoS threats, in order to design implementation-specific DoS resistance policies.

4.2 The PRISM model for the HIP base-exchange

The PRISM model for the HIP base-exchange implements the state machine shown in Figure 3. This machine captures the Initiator and Responder processes in a single system view, representing either an Initiator or a Responder. The protocol participants' states are defined in Table 2.

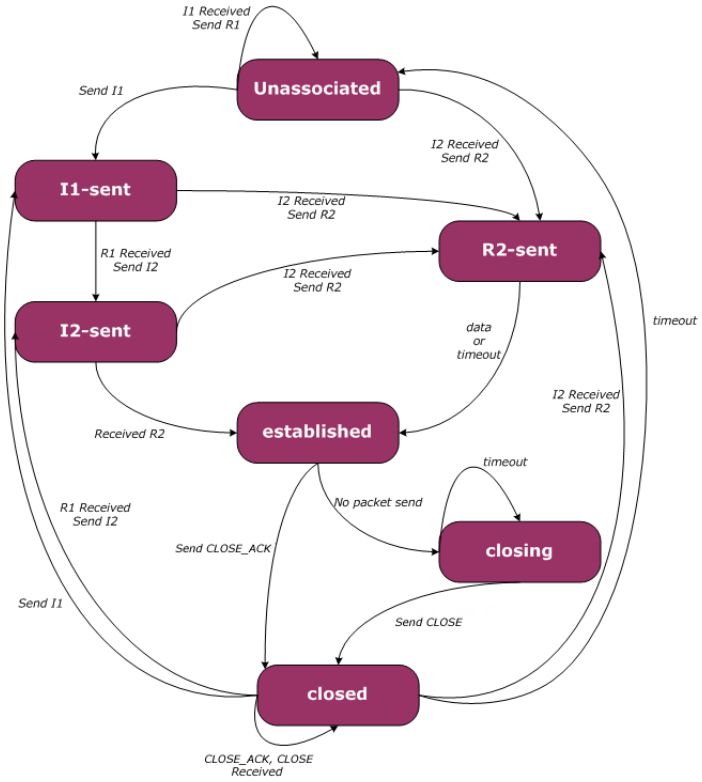

Figure 3. The HIP state transition diagram

Table 2 The HIP base exchange states

State	Description
Unassociated	State machine start
I1-sent	Initiating base exchange
I2-sent	Waiting to complete base exchange
R2-sent	Waiting to complete base exchange
established	HIP association established
closing	HIP association closing, no data can be sent
closed	HIP association closed, no data can be sent

We observe that there are states which imply a waiting time for the receipt of some specific message sent by the right participant. State `I1-sent` for example is reached either

from `Unassociated` or `closed` upon dispatch of I1 and in this state the participant waits for message R1 or I2, whose receipt triggers the actions shown in the corresponding transitions. States `establish`, `close` and `closing` represent successful establishment of HIP security associations.

Our PRISM model includes four modules: (i) the Medium (*m*), representing the communication channel used by the protocol participants to exchange messages, (ii) the Initiator (*I*), (iii) the Responder (*R*) and (iv) the Attacker (*At*). In their parallel composition, the aforementioned modules interact by updates to their local variables. These updates correspond to the modeled state transitions.

```
dtmc
const int INIT_=10;                    //time for initiating/sending first message
const int PR_I1_GEN_R1=15;             //processing time of I1 and generation of R1
const double PR_R1_GEN_I2_k5=0.000934; //processing time of R1 and generation of I2 with R1(puzzle(k)=5)
const double PR_R1_GEN_I2_k10=0.0225;  //processing time of R1 and generation of I2 with R1(puzzle(k)=10)
const double PR_R1_GEN_I2_k15=0.808;   //processing time of R1 and generation of I2 with R1(puzzle(k)=15)
const double PR_R1_GEN_I2_k20=15.3;    //processing time of R1 and generation of I2 with R1(puzzle(k)=20)
const double PR_R1_GEN_I2_k25=630;     //processing time of R1 and generation of I2 with R1(puzzle(k)=25)
const int PR_I2_GEN_R2=170;            //processing time of I2 and generation of R2
const int PR_R2=9;                     //processing time of R2
const int TRNSMT_=10;                  //transmition cost
const int TRNSMT_INTR=9;               //attacker's transmition cost

//HIP formulas being used into the model

formula no_action = c1=2 & c2=2;                    //no action over the channel
formula sending_init = c1=1 & c2=2;                 //initiator sending
formula sending_resp = c1=2 & c2=1;                 //responder sending

formula message1 = HITi=1 & HITr_sent=1;            //creation of message I1
formula message1_int = HITi_stolen=1 & HITr_stolen=1;   //message I1 for the attacker
formula message2 = HITi_received=1 & HITr=1 & puzzle=true & HITr=1 & gr=1 & HITi_received=1 & signr1=1;
global connection : bool init false;                //connection flag indicating an open connection between I and R

//Constants
const int M = 50;                      //number of distinct counterfeited messages
const int B= 80;                       //maximum number of messages in Initiator's admission queue
const int proc_limit = 40;             //number of messages that are served simultaneously by the Initiator
const int MAX = 40;                    //number of available threads for serving messages in Initiator's admission queue
global a : [0..20] init 0;             //number of R1 message replays for each counterfeited message
```

Figure 4. Global variables of the HIP model

Figure 4 introduces the global declarations of the model. We note that the discrete time steps of our DTMC represent state transitions that reflect the occurrence of one or more events in the system model and they are not used as a means to represent time delays in

protocol execution. The constants in the first part of the shown declarations introduce processing time costs for the reward associations defined for the model's states. The used values are based on the considered message processing demands (data taken from related HIP performance studies [13]) and on the relative differences between the interacting participants regarding their processing capacity (number of instructions processed per second). Variables *c1* and *c2* are local variables of module *m* and the formulae of Figure 4 define values that jointly represent a message flow either from the *Initiator* to the *Responder* or vice-versa. Two variables are related to HIP implementation parameters: (i) *proc_limit* represents the number of messages that are served simultaneously by *I* and (ii) *B* denotes the maximum number of messages in Initiator's admission queue. Finally, global variable a represents the number of R1 message replays generated by module *At* for each counterfeited message created by the attack tactics shown in Figure 1 and *M* expresses the total number of distinct counterfeited messages.

```
//Medium of the communication network handling all exchanged messages
module medium
c1 : [1..2] init 2;                    //c1=1: no action towards responder
c2 : [1..2] init 2;                    //c1=2: action towards responder
//MESSAGE I1
[m1]c1=2 & c2=2 -> c1'=1;
[finish_m1](c1=1 & c2=2 & s3=1) -> (c1'=2) & (c2'=2);
////////////////
//MESSAGE Ix (either I1 or I2), Attacker->Responder
[intr_1]no_action -> c1'=1;
[finish_intr_1]sending_init & s3=4-> c1'=2;
////////////////
//MESSAGE R1
[m2]no_action & message2 -> c2'=1;
[finish_m2]sending_resp & s1=4 -> (c1'=2) & (c2'=2);
////////////////
//SPOOFED MESSAGE Rx (either R1 or R2) , Attacker->Initiator
[spoof]no_action -> c2'=1;
[finish_spoof]sending_resp -> c2'=2;
///////////////
//MESSAGE I2
[m3]no_action -> c1'=1;
[finish_m3]sending_init -> c1'=2;
///////////////
//MESSAGE R2
[sendingr2]no_action -> c2'=1;
[finishr2]sending_resp -> c2'=2;
endmodule
```

Figure 5. The PRISM module for the communication medium (*m*)

Figure 5 shows the guarded commands

[ ] <guard> → <command>;

of module *m*. The command(s) corresponding to some guard specify the possible transition(s) for *m*, if the guard is true. The declarations of Figure 4 defined the possible message flows in *m*, which are the following:

a. *No messages* (*c1*=2 & *c2*=2) – there is no undelivered message
b. *Message origin 1* (*c1*=1 & *c2*=2) – *I* (or *At*) sends a message to module *R*
c. *Message origin 2* (*c1*=2 & *c2*=1) – *R* (or *At*) sends a message to module *I*

The values of *c1* and *c2* and the values of variable `s3` (with `s3`=4 meaning that *At* received I1) and `s1` (with `s1`=4 meaning that *I* is ready to receive some message) determine the different states where some command is enabled for module *m*. The message transmissions are implemented in two stages thus allowing interference of module *At* taking place in between them. In this way *At* has absolute control over the network, since the exchanged protocol messages can enhance the Attacker's knowledge base.

However, for the analyzed DoS threat it is enough the Attacker's knowledge to include only the first two messages of the HIP base-exchange. The *Attacker* exploits the fact that the puzzle sent to the *Initiator* in message R1 is not included in the pre-computed signed part of the message. The puzzle is generated on demand based on a random nonce and a parameter *k* that adjusts the puzzle difficulty and in effect influences the Initiator's cost to compute the solution. Certainly, the protocol requirement for generation of fresh puzzles protects the *Responder* from time shifting and message replay DoS threats, but at the same time makes the protocol vulnerable to the counterfeiting DoS threat shown in Figure 1.

If in a HIP implementation there is no providence that will prevent the *Initiator* from accepting replayed R1s, then the considered *Attacker* (*At*) can prevent the *Initiator* from ever solving an authentic puzzle. This attack succeeds when the frequency of the

counterfeited messages is higher than the roundtrip between the Initiator and the Responder. In this case the first R1 that arrives at the Initiator after having sent I1 is always a replay of a counterfeited message and the protocol's *DoS resistance* depends on the selected values of *proc_limit*, *B* and *k*.

```
module attacker
s3 : [1..9] init 1;
//1=wait
//2=message I1 received
//3=get hit's
HITi_stolen : [0..1] init 0;
HITr_stolen : [0..1] init 0;
puzzle_received : [0..1] init 0;
puzzle_change : [0..M] init 0;
flag : bool init false;

//GETS I1 AND SENDS IT TO RESPONDER
[finish_m1]s3=1 & message1 & sending_init -> s3'=2;
[]s3=2 -> HITi_stolen'=1 & HITr_stolen'=1 & s3'=3;
[]s3=3 & message1 -> s3'=4;
[intr_1]no_action & s3=4 & flag=false -> s3'=4 & flag'=true;
[finish_intr_1]s3=4 -> s3'=5;
////////////////////////////////

//RECEIVES R1 AND CREATES AND SENDS THE SPOOFED MESSAGES
[finish_m2]s3=5 & message2 & sending_resp ->s3'=6;
[]s3=6 -> 0.5:(a'=10) & (s3'=7) + 0.5:(a'=20) & (s3'=7);
[]s3=7 -> puzzle_received'=1 & s3'=8;
[spoof]s3=8 & puzzle_change<M & no_action -> puzzle_change'=puzzle_change+1 & s3'=9;
[finish_spoof]s3=9-> s3'=8;
////////////////////////////////
endmodule
```

Figure 6. The PRISM module for the Attacker (*At*)

Figure 6 shows the guarded commands of module *At*. The Attacker's behavior is encoded into nine (9) distinct states represented by local variable `s3`. In state 2, module *At* intercepts I1 and subsequently forwards the intercepted message to the intended recipient. In state 5, module *At* intercepts R1 and subsequently sends to module *I* a trickle of counterfeited message replays that aim to trick the *Initiator* into spending its processing power for solving wrong puzzles.

The modeled *Attacker* is supposed to create in total *M* counterfeited R1s with wrong puzzles. Each counterfeited R1 is replayed `a` times and these message replays are modeled with a single state transition. The number of message replays (`a`) is selected by module *At*

with equal probability between the two values shown in Figure 6. Finally, Figure 7 introduces the processing costs that are assigned to selected states of module *At*. These costs correspond to the generation of the messages implementing the modeled DoS threat and the message transmission costs, which depend on the value of a.

```
rewards "attacker"
......
[finish_m1] true :(INIT_+TRNSMT_);
[intr_1] true : (TRNSMT_INTR);
[finish_intr_1] true : a*(TRNSMT_INTR);
[finish_m2] true : a*(TRNSMT_+PR_I1_GEN_R1);
[finish_spoof] true : a*(TRNSMT_INTR);
[spoof] true : (TRNSMT_INTR);
......
endrewards
```

Figure 7. Processing costs assigned to selected Attacker states

```
module initiator
s1 : [1..7] init 1;
//states for initiator
//1: initial state
//2: prepare I1
//3: sending I1
//4: wait for R1 and send I2
//5: complete the protocol processing
HITi : [0..1] init 0;                    //initiator's host identity tag
HITr_sent : [0..1] init 0;               //responder's host identity tag
counter : [0..B] init 0;                 //number of messages in server's admission queue
verify_sign : bool init false;           //signature verification for received message R2
ex : [0..MAX] init 0;                    //number of utilized threads serving messages in Initiator's admission queue
verify_HMAC : bool init false;           //HMAC verification

// CREATE MESSAGE I1
[]s1=1 -> s1'=2;
[]HITi=0 & HITr_sent=0 & s1=2 -> (HITi'=1) & (HITr_sent'=1);
[]s1=2 & message1 -> s1'=3;
[m1]no_action & s1=3 ->s1'=3;
[finish_m1]s1=3 -> s1'=4;
//////////////////
//PROCCESS R1 & CREATE AND SEND MESSAGE I2
[finish_spoof]s1=4 & counter<=B-a -> counter'=counter+a;
[]counter>=a & ex<=MAX-a & a!=0 -> ex'=ex+a & counter'=counter-a;
[m3]s1=4 & ex>=a & ex>0 -> ex'=ex-a & s1'=5 ;
////////////////////
//PROCCESS R2
[]s1=5 -> verify_HMAC'=true & connection'=true;
[finishr2]s1=5 & verify_HMAC=false ->s1'=7;
////////////////////
//FAIL CONDITIONS
[]ex=MAX & counter>=proc_limit ->fail'=2;
[]s1=7 -> fail'=1;
////////////////////
endmodule
```

Figure 8. The PRISM module for the *Initiator* (*I*)

Figure 8 shows the guarded commands of module *I*. The Initiator's behavior is encoded into seven (7) distinct states represented by local variable `s1` and one state corresponding to DoS failure, which is activated when the local variable `fail` is assigned the value 2. The different module states and the associated conditions enable appropriate send and receive actions for the messages contained in the HIP base-exchange. Module *I* receives batches of `a` replays of counterfeited messages R1 and allocates threads taken from a pool of *MAX* threads, for processing the messages queued in the server's admission queue. The reward association shown in Figure 9 assigns the processing cost to some state of module *I*, where the Initiator computes the solutions for the puzzles of `a` counterfeited R1 messages. These costs depend on the selected value of puzzle difficulty (*k*) that in fact determines the number of trials required in a brute-force search, in order to find the solution in a given puzzle (data based on related HIP performance studies [13]). To keep our description concise we omit the code of module *R*, which the reader can obtain from [23].

```
rewards "k_value"
......
[m3] k=1 : a*(TRNSMT_+PR_R1_GEN_I2_k5);
[m3] k=10 : a*(TRNSMT_+PR_R1_GEN_I2_k10);
[m3] k=15 : a*(TRNSMT_+PR_R1_GEN_I2_k15);
[m3] k=20 : a*(TRNSMT_+PR_R1_GEN_I2_k20);
[m3] k=25 : a*(TRNSMT_+PR_R1_GEN_I2_k25);
......
endrewards
```

Figure 9. Processing costs to the Initiator for different values of *k*

4.3 Probabilistic model checking results

For the generated DTMC, the PCTL query

```
Q1: P=? [true U fail=2]
```

evaluates the probability of taking a path that *eventually* reaches some state, where variable `fail` is equal to 2. In any such state, module *I* processes simultaneously

proc_limit messages with wrong puzzles and in the steady state the server's admission queue includes *B* messages in total. As a consequence of these facts, any valid R1 message will be dropped. According to what we mentioned in Section 3.2 for quantifying the DoS resistance of the modeled protocol, if the *Attacker* brings the system into this situation with high probability and the cumulated processing cost is disproportionaly low compared to the cost to the *Initiator*, we have proved an outstanding case of DoS threat.

```
PRISM
=====

Version: 3.1.1
Date: Tue Jan 15 17:25:09 EET 2008
Hostname: user

Parsing model...

Building model...

Computing reachable states...

Reachability: 520 iterations in 110.02 seconds (average 0.211569, setup 0.00)

Time for model construction: 255.547 seconds.

States:      8733343 (1 initial)
Transitions: 31988778

Transition matrix: 155297 nodes (16 terminal), 31988778 minterms, vars: 78r/78c
Transition rewards (0): 173 nodes (2 terminal), 5 minterms
Transition rewards (1): 9387 nodes (6 terminal), 2203809 minterms
Transition rewards (2): 9378 nodes (5 terminal), 2203809 minterms
Transition rewards (3): 9439 nodes (3 terminal), 2892600 minterms

Model checking: P=? [ true U fail=2 ]

Prob1: 1052 iterations in 25.67 seconds (average 0.024403, setup 0.00)

Prob0: 103 iterations in 3.61 seconds (average 0.035039, setup 0.00)

yes = 3288740, no = 3324002, maybe = 2120601

Computing remaining probabilities...

Building hybrid MTBDD matrix... [levels=78, nodes=104514] [2449.5 KB]
Adding explicit sparse matrices... [levels=24, num=1141, compact] [785.4 KB]
Creating vector for diagonals... [dist=6, compact] [17057.4 KB]
Creating vector for RHS... [dist=2, compact] [17057.3 KB]
Allocating iteration vectors... [2 x 68229.2 KB]
TOTAL: [173808.1 KB]

Starting iterations...

Jacobi: 500 iterations in 148.11 seconds (average 0.280094, setup 8.06)

Time for model checking: 178.532 seconds.

Result: 0.8948948113273193
```

Figure 10. Probability to eventually reach a state where the Initiator is not available

The results shown in Figure 10 reveal an unacceptably high probability (`P=0.895`) to eventually reach a state that reflects the discussed situation.

A more illuminating view of the variation of the probability to reach a state where module *I* is unavailable is the one of Figure 11. For the purposes of the shown PCTL experiment we defined the constants `msgs_in_service` and `msgs_in_queue`, in order to examine the behaviour of the model in a bounded range of values for the variables `proc_limit` and `B`. The shown results are obtained with the query:

```
Q2: P=? [true U msgs_in_service=proc_limit & msgs_in_queue=B]
```

where `proc_limit` is varied between 0 and 200 with step 50 and `B` is varied between 0 and 400 with step 50.

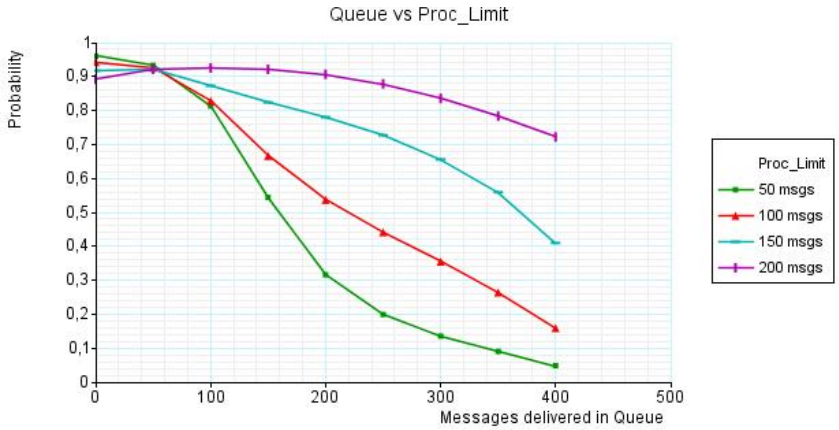

Figure 11. Probability to eventually reach a state where module *I* is unavailable for different values of `proc_limit` and `B`

We note that the system's DoS resistance is improved for allowed queue lengths of 250 messages or more, under the condition that module *I* will not process simultaneously more than 50 messages. These findings provide valuable information for fine-tuning HIP protocol implementations.

However, a complete view for the protocol's DoS resistance is obtained only when having reviewed the results of the following reward queries regarding *I*'s and *At*'s processing costs. Query `Q3` provides for different values of puzzle difficulty *k* the expected cumulated processing cost to module *I*, when having reached some state in which it is not

available any more. We note that the puzzle difficulty is originally selected by module *R* in a probabilistic manner.

```
Q3: R[Initiator_cost]=? [true U msgs_in_queue=B & k=puz_dif]
```

Figure 12 shows the obtained results for `B` varied between 0 and 100 with step 20 and assigned puzzle difficulty *k* selected from the values 1, 10, 15, 20, 25 (`puz_dif`). For allowed queue lengths between 0 and 60 the cumulated processing cost for module *I* can be dramatically increased, especially when the used puzzle difficulty is a number greater than 20.

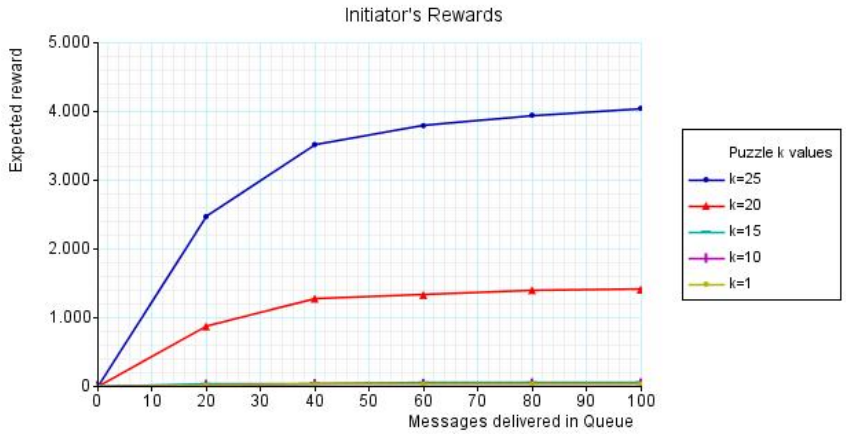

Figure 12. Expected cumulated processing cost for Initiator when becoming unavailable

Reward queries `Q4` and `Q5` complete the picture for the system's DoS resistance. The graphs shown in Figure 13 provide a comparative view of the expected cumulated processing cost to module *At* against the cost incurred to module *I* when becoming unavailable:

```
Q4: R[Initiator_cost]=? [true U msgs_in_queue=B & k=puz_dif]
Q5: R[Attacker_cost]=? [true U msgs_in_queue=B & k=puz_dif]
```

As before, `B` varied between 0 and 100 with step 20 and we generated results for puzzle difficulty 1, 10, 15, 20 and 25 (`puz_dif`).

The shown results provide valuable insight regarding the severity of the demonstrated DoS threat for different combinations of puzzle difficulty, the number of messages simultaneously processed by the *Initiator* and its allowed queue length. Our *Attacker* module counterfeits partially signed messages dispatched by module *R* and replays the counterfeited messages, in order to trick the module representing the *Initiator* into processing wrong puzzles. An efficient approach towards the deamplification of the demonstrated attack is the use of puzzle difficulty values between 10 and 18 and an appropriate adjustment in the allowed queue length for the *Initiator* and the number of messages simultaneously processed. We note that the puzzle difficulty values that make it possible to deamplify the demonstrated attack are also proposed as appropriate implementation parameters in technical reports for the HIP base-exchange ([14], [13]).

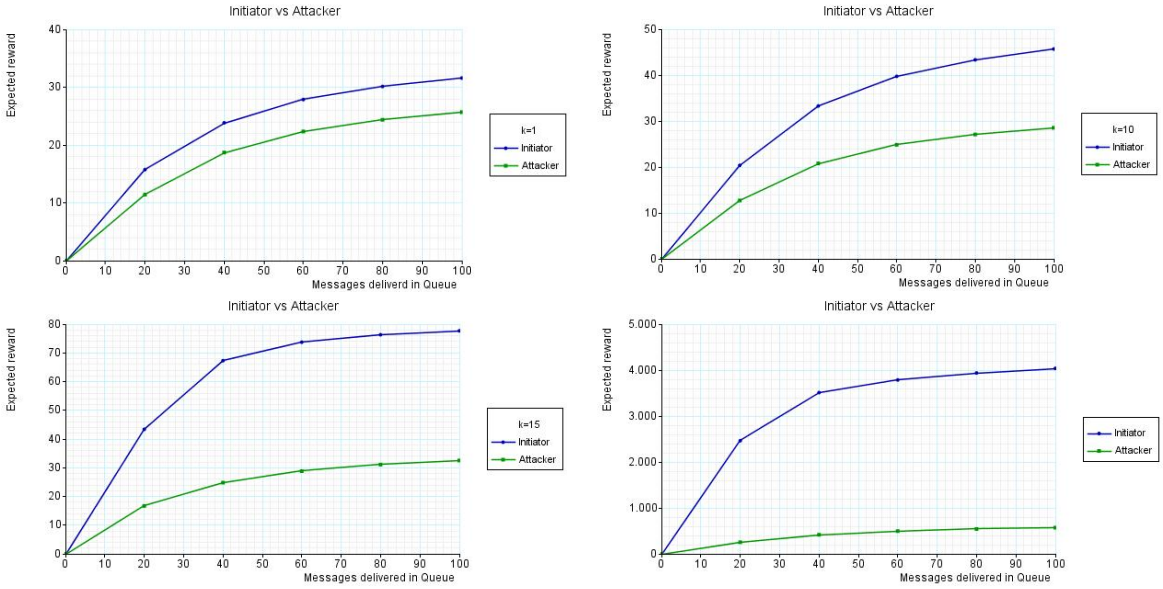

Figure 13. Expected cumulated processing costs to the Initiator and the Attacker for $k = 1, 10, 15, 25$

An alternative countermeasure with considerable implementation cost is the use of an R1 generation counter per host identity ([14]). This monotonically increasing counter will

indicate the current generation of puzzles and a host will accept puzzles from the same generation, but it may be also possible to accept puzzles from earlier generations. Again, our quantitative verification approach is a valuable means in the design of such countermeasures.

In PRISM it is possible to combine multiple DoS attacks within the same model. In this case, the underlying probabilistic model will be a Markov Decision Process (MDP), instead of DTMC. However, an inherent risk in trying to quantify DoS resistance with an MDP model is the potentiality for state space explosion or inexpedient query response times that will make the needed experimentation practically infeasible. Also, comparison of the costs to the different protocol participants will not be easy, since when the PCTL reward queries that are applied to an MDP compute the minimum or maximum costs for the analyzed condition. These costs will correspond to different DoS attacks and for this reason there is no obvious way to compare the participants' costs under the same DoS threat.

Table 3 PCTL queries performance

DTMC model generation		
CPU time (sec):	**Number of states:**	**Number of transitions:**
255.547	8733343	31988778
PCTL query Q1		
CPU time (sec):	**Memory allocation (KB):**	**Jaccobi iterations:**
178.532	173808.1	500 (in 148.11 sec)
PCTL query Q2 (proc_limit=200 msgs, B=400)		
CPU time (sec):	**Memory allocation (KB):**	**Jaccobi iterations:**
204.953	178593.2	488 (in 187.86 sec)
***PCTL query* Q3** (k=10)		
CPU time (sec):	**msgs_in_queue:**	**Jaccobi iterations:**
7.547	20	113 (in 6.52 sec)
16.109	40	213 (in 15.17 sec)
27.625	60	313 (in 25.78 sec)
36.359	80	413 (in 35.20 sec)
45.547	100	513 (in 44.64 sec)
***PCTL query* Q5** (k=20)		
CPU time (sec):	**msgs_in_queue:**	**Jaccobi iterations:**
7.469	20	173 (in 6.47 sec)
16.218	40	273 (in 15.19 sec)
27.516	60	373 (in 25.72 sec)
36.016	80	473 (in 34.78 sec)
45.875	100	573 (in 45.78 sec)

In Table 3, we provide representative performance statistics for the queries shown in this paragraph. All experiments took place on a Pentium IV machine (3.6 GHz) with 2 GB memory and as we see the required CPU time for the used iterative solver to converge is determined by the size of the generated DTMC state space. This finding makes it worth to comment on the efficiency and the scalability of our analysis approach with respect to the system model size. We observe that the consumed CPU time for the performed PCTL queries increases with sublinear rate with respect the number of messages allowed in the server's admission queue. The main factors that affect the growth of the DTMC state space are:

- Protocol implementation parameters such as the size of the servers' admission queues and the number of messages that are served simultaneously.
- The attack tactics implemented by the modeled *Attacker* and parameters such as the number of counterfeited messages and the number of message replays in the DoS attack.

The parameters used in the analysis of the HIP base-exchange represent realistic protocol implementation cases that were also used in related performance measurement studies. On the other hand, the DoS attack tactics of section 3.3 require interception of one particular protocol message and for this reason it is reasonable to assume that the provided statistics are representative cases of the vast majority of analysis problems targeted by our approach. However, if the implemented DoS attacker combines additional attack tactics from the ones described in [5] (e.g. parallel sessions, type flaws etc.) then we propose the analyst to consider using a smaller number of counterfeited messages and a higher number of message replays that will be assigned on a deterministic basis.

5. Conclusion

This work introduces a quantitative verification approach as a means for deamplification of DoS security threats or for studying the effectiveness of countermeasures, which can be applied in potential protocol implementations. Our approach is based on a general probabilistic attacker model encompassing the most common DoS attack strategies. We treated the protocol's DoS resistance property as a quantifiable measure that depends (i) on the probability to reach a state where some protocol participant becomes unavailable and (ii) the requirement for the attacker to cause this event with disproportionaly low cost compared to the cost incurred to the victim. The general framework makes it possible to model any form of resource expenditure and in effect the analyst develops a cost model representing the most prominent resource exhaustion possibility (communication bandwidth, memory or processing power).

The overall analysis takes place in a mature probabilistic model checking tool called PRISM. We developed a probabilistic model for the HIP base-exchange protocol that uses a client puzzle mechanism for protection against DoS. The model is coupled to a reward structure representing the processing costs for the protocol participants and the considered instance of the general attacker model. The obtained results provide valuable insight regarding the severity of the demonstrated DoS threat in different protocol implementation cases.

To the best of our knowledge, our work is the first attempt to quantify DoS security threats with probabilistic model checking. We believe that the proposed approach can be utilized in the design of new security protocols and protocol implementations. The analyst works with a toolset that allows focusing on the formulation of the protocol as a state-transition system associated with an appropriate cost model. In general this task is not particularly difficult for most security experts. Moreover, the PCTL queries for quantifying

the protocol's DoS resistance can be potentially generated by an appropriate specification pattern system that will guide the specification of probabilistic and reward properties based on a structured English grammar [11].

As a future research prospect, we plan to extend the range of DoS attack strategies included in the general attacker model. A first priority aim is to make it possible to develop model instances, where a number of attackers collude in some way, in order to perform a *distributed denial of service attack* (e.g. the well known "cookie jar" attack).

References

[1] Agha, G., Greenwald, M., Gunter, C. A., Khanna, S., Meseguer, J., Sen, K., Thati, P. Formal modelling and analysis of DoS using probabilistic rewrite theories, Proceedings of the IEEE Workshop on Foundations of Computer Security (FCS '05), Chicago, 2005

[2] Aiello, W., Bellovin, S. M., Blaze, M., Ioannidis, J., Reingold, O., Canetti, R., Keromytis, A. D. DoS-resistant, secure key exchange for internet protocols, Proceedings of the 9th ACM Conference on Computer and Communications Security, ACM Press, 48-58, 2002

[3] Aura, T., Nikander, P. Stateless connections, Proceedings of the 1st International Conference on Information and Communication Security (ICICS), LNCS 1334, Springer Verlag, 87-97, 1997

[4] Aura, T., Nikander, P., Leiwo, J. DOS-resistant authentication with client puzzles, Proceedings of the Security Protocols Workshop, Cambridge, UK, LNCS 2133, Springer Verlag, 170-181, 2001

[5] Basagiannis, S., Katsaros, P. and Pombortsis, A. Intrusion Attack Tactics for the model checking of e-commerce security guarantees, Proceedings of the 26th International Conference on Computer Safety, Reliability and Security (SAFECOMP), Nuremberg, Germany, LNCS 4680, Springer Verlag, 238-251, 2007

[6] Castelliccia, C., Mykletun, E., Tsudik, G. Improving secure server performance by rebalancing SSL/TLS handshakes, Cryptology ePrint Archive 2005/037, IACR, http://eprint.iacr.org/2005/037 (last access: 18th of August 2008)

[7] Cederquist, J. and Dashti, M. T. An intruder model for verifying liveness in security protocols, Proceedings of the 4th ACM Workshop on Formal Methods in Security, 23-32, 2006

[8] CERT Coordination Center. Denial of service attacks. http://www.cert.org/tech_tips/denial_of_service.html, (last access: 28th of December 2007)

[9] Dolev, D. and Yao, A. On the security of public-key protocols, IEEE Transactions on Information Theory, 2 (29), 198-208, 1983

[10] Gärtner, F. Revisiting liveness properties in the context of secure systems, Proceedings of the 1st International Conference on Formal Aspects on Security, Springer LNCS 2629, 203-211, 2003

[11] Grunske, L. Specification patterns for probabilistic quality patterns, Proceedings of the 30th International Conference on Software Engineering (ICSE), ACM Press, 31-40, 2008

[12] Hansson, H., Jonsson, B. A logic for reasoning about time and reliability, Formal Aspects of Computing, 6 (5), 512-535, 1994

[13] InfraHIP Project Web Site, http://infrahip.hiit.fi/, (last access: 21st of December 2007)

[14] Internet Engineering Task Force (IETF) - Network Working Group. Host Identity Protocol. Internet Draft, February 2007

[15] Karn, P. and Simpson, A. Photuris: Session-key management protocol, RFC 2522, IETF Network Working Group, 1999

[16] Kemeny, J., Snell, J., Knapp, A. Denumerable Markov Chains, 2nd edition, Springer, 1976

[17] Kwiatkowska, M., Norman, G., Parker, D. Stochastic model checking, In: Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM´07), Ed: M. Bernardo and J. Hillston, Springer LNCS 4486, 220-270, 2007

[18] Kwiatkowska, M. Quantitative verification: Models, Techniques and Tools, Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), ACM Press, 449-458, 2007

[19] Madan, B. B., Goseva-Popstojanova, K., Vaidyanathan, K., Trivedi, K. S. Modeling and quantification of security attributes of software systems, Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 02), IEEE Computer Society Press, 2002

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

[21] Meadows, C. Formal methods for cryptographic protocol analysis: emerging issues and trends, IEEE Journal on Selected Areas in Communications, 21 (1), 44-54, 2003

[22] PRISM Model Checker Web Site, http://www.prismmodelchecker.org/, (last access: 3rd of January 2008)

[23] Probabilistic Model Checking DoS in HIP – PRISM model online: http://mathind.csd.auth.gr/dos_work.html, (last access: 2nd of September 2008)

[24] Sen, K., Viswanathan, M., Agha, G. VESTA: A statistical model-checker and analyzer for probabilistic systems, Proceedings of the 2nd Int. Conference on the Quantitative Evaluation of Systems (QEST'05), 251-252, 2005

[25] Sen, K., Viswanathan, M., Agha, G. On statistical model checking of stochastic systems, Proceedings of the 17th Int. Conference on Computer Aided Verification (CAV'05), Springer LNCS 3576, 266-288, 2005

[26] Smith, J., Gonzalez-Nieto, J. M., Boyd, C. Modelling denial of service attacks on JFK with Meadows's cost-based framework, Proceedings of the Fourth Australasian Information Security Workshop (AISW-NetSec), CRPIT Series, 125-134, 2006

[27] Tritilanunt, S., Boyd, C., Foo, E., Gonzalez Neto, J. M. Using Coloured Petri Nets to simulate DoS-resistant protocols, Proceedings of the Seventh Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, University of Aarhus, Denmark, 2006