A Method to implement a denial of service protection base

Jussipekka Leiwo and Yuliang Zheng

Peninsula School of Computing and Information Technology
Monash University
McMahons Road, Frankston, Vic 3199, Australia
Phone +61-(0)3-9904 4287, Fax +61-(0)3-9904 4124
E-mail: {skylark,yuliang}@fcit.monash.edu.au

Abstract. Denial of service attack is an attempt from any authorized or unauthorized entity to allocate resources excessively to prevent normal operation of the system. A method will be presented to specify and enforce a resource allocation policy to prevent denial of service attacks. Resource allocation policy can be formally derived from a waiting time policy where maximum acceptable response times for different processes are specified and enforced by a formally specified algorithm.

1 Introduction

An efficient method will be proposed to specify and enforce a Resource Allocation Policy (RAP) to protect against denial of service attacks. Denial of service attack occurs when authorized entities are intentionally prevented from accessing information or service. Threats that may lead to denial of service can be divided into those through resource allocation and those through resource destruction [13]. Our focus is on the prevention of threats through resource allocation.

Several formal access control list (ACL) based models exist to protect confidentiality and integrity, but as shown by [9], ACL based protection of denial of service is an undecidable problem. Glasgow et al. [7] suggest that temporal operators included in the integrity specifications could be applied to the specification of availability properties but do not study the issue further. An effective solution to denial of service must be based on the control of resource allocation [8]. Denial of service protection base (DPB) is a layer of trusted computing base (TCB) that controls allocation of resources [13]. This paper proposes an efficient algorithm to implement a DPB that monitors allocated and available resources and, based on RAP, determines whether the resource allocation request should be denied or granted.

Prevention of denial of service is a fundamental objective of protection of availability, that is one of three common information security objectives [1] and a major factor in dependable computing [11]. Comprehensive protection requires measures at several technical and non-technical layers [12] but when formal protection models are considered, the scope needs to be narrowed. Some sources, such as [5], suggest that due to the aforementioned problems, availability should

not be considered as a general objective of information security. This is, anyhow, a dangerous limitation. As analyzed by Needham [14], cases can be identified where availability is the major concern of information systems design, and other security objectives are less important. As violations of availability easily lead to considerable long interrupts of services, they may cause serious harm to business. As shown by [16], even short interrupts in information processing services may cause serious operational delays for organizations. Recently, wide spread denial of service attacks in TCP/IP protocol suite, such as TCP SYN flooding [3] and ping attack [2] have provoked discussion on the protection against denial of service.

Protection methods of availability listed in [10] mostly focus on administrative routines. Administrative routines are considered effective measures for recovering and correcting the system after an attack, or for reducing the probability of becoming a target of an attack [15], but to prevent violations, different approach must be taken. This paper presents a method to specify and enforce a formal RAP. The paper will start by the specification of architecture for a DPB in section 2. After this, an exact definition for a denial of service attack will be given in section 3. Resource allocation policy and its enforcement will be analyzed in detail in section 4. Performance of the suggested DPB algorithm will be evaluated in section 5. An example of the application of the model to prevent TCP SYN flooding will be given in section 6. Finally, conclusions will be drawn and directions for future work highlighted in section 7.

2 DPB architecture

DPB is a layer added to the TCB, that guarantees, that no resource can be allocated in a manner that would cause a denial of service. For the purposes of this paper, the architecture illustrated in figure 1 is assumed. When a process requests for a resource, the request is first handled by the DPB. DPB consults the RAP, that is a statement of authorized resource allocations based on higher level Waiting Time Policy (WTP) to decide whether the request should be accepted according to the resource allocation rules, and Resource Allocation Table (RAT) for the current allocation of resources among different processes. DPB can either grant or deny the request, or declare the request undecidable.

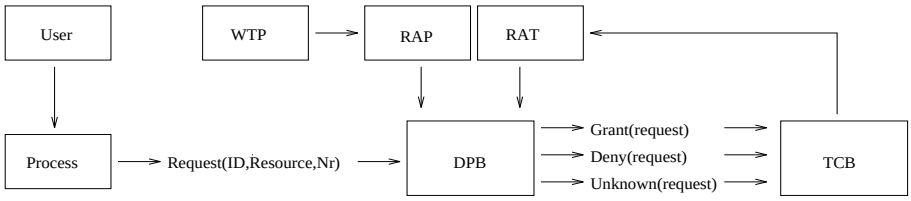

Fig. 1. DPB architecture

A system consists of a finite number of resources, and processes requesting for access to resources. Resources can be classified into types, and the total number of resources of a type can vary. A resource can be allocated for a particular process, or it can be free. If a resource is free, it can be allocated on condition that RAP rules are not violated. The information concerning the states of resources is stored in the RAT. DPB can not write information on RAT, only read the state of a process. All updates must be carried out by other parts of TCB, where the actual allocation of resources is done according to other security policies.

Resources can be requested only by processes. Each process has an owner, that can be either a user or other process. In the case of the owner being other process, it is assumed that the original user that the process is operating on behalf, can be tracked down. User can be either an actual user of the system, or a logical user, that can be a machine, a channel such as input device or cryptographic channel, conjunction of users, a group, a user in a role, or a user on behalf of another user [4]. How much a user can have allocated resources, and what is the total amount of resources that can be allocated, will be specified in RAP.

3 Denial of service attack

A denial of service occurs when an entity either is prevented from performing a task it is authorized to, or prevents other entities of performing their tasks. The nature of these threats can vary from logical attacks via sabotage and other physical attacks and accidents to natural catastrophes like floods and falling trees. As this paper is concerned on the protection against denial of service attack, that is an intentional logical attempt to violate the system's availability, it is assumed that a denial of service attack is any intentional attempt to excessively allocate or destroy resources so that normal operation of the system is disabled and an authorized request is either delayed or prevented. Since unauthorized destruction of a resource is merely a problem of confidentiality or integrity, the focus within this paper will be on the prevention of excessive allocation of resources. A simple example of a denial of service attack is a process, acting on behalf of a user, that duplicates itself until it occupies most of the memory and gets most of the CPU time.

Formally, we assume that for each resource ρ_i within the system, exists a maximum limit of simultaneous allocations that can be executed on behalf of a single user, δ_i. We consider each δ_i as a threshold that shouldn't be exceeded in normal operation. Assume that for each user j and process ρ_i, exists a function $Total$ where $Total(i,j) = n$ where n is the number of resources ρ_i allocated to user j. A denial of service attack is an attempt of a process acting on behalf of a user to allocate resources so, that the total number of resource ρ_i allocated to the user j exceeds the threshold δ_i. This is formulated in definition 1.

Definition 1. Denial of service attack is an attempt of a user j to allocate instances of any resource ρ_i so, that if allocated, $Total(i,j) > \delta_i$.

Function $Total$ will be exactly specified in equation 3, in section 4.3. Threshold δ can be formulated according the Resource Allocation Policy, specified in section 4.2.

4 Structure of DPB

In this section, the core of this paper, resource allocation policy, and enforcement of this policy will be presented. The analysis will be started by giving an high level view of the algorithm that enforces DPB in section 4.1. Section 4.2 is a detailed analysis of the policy. Then, section 4.3 specifies the contents of the resource allocation table. Section 4.4 provides with an algorithm to enforce the resource allocation policy. Issues related to the specification and transformation of WPT into RAP will be discussed in section 4.5. Section 4.6 discusses handling of undecidable requests.

4.1 Overview

The general high level algorithm for the enforcement of DPB is as follows. Let MAX^i_{total} be the number of resource ρ_i that can be allocated to all processes, MAX^i_{proc} the number of resource ρ_i that can be allocated to one process, CUR_i be the current allocation of resource ρ_i, and $TOTAL^i$ be the total amount of resource ρ_i available. Exact specifications for these variables will be given later within this section.

INPUT: id, i, nr
Begin
  Fetch MAX^i_{total} from RAP
  Fetch MAX^i_{proc} from RAP
  Fetch CUR_i from RAT
  Fetch $TOTAL^i$ from RAP
  If $(CUR_i + nr > MAX^i_{total})$ Then Result=Deny
  Else If $(CUR_i + nr > MAX^i_{proc})$ Then Result=Deny
  Else If $(CUR_i + nr > TOTAL^i)$ Then Result=Undecidable
  Else Result=Grant
End
OUTPUT: Result

The following notation shall be used throughout the rest of the paper:

a,b,c are used as temporary indexes
i is used to index different resources ρ_i
j is used to identify each user
k represents number of users
l represents the number of resources
δ_a represents a threshold that any allocation of resource ρ_a should not exceed
ρ_a represents resource of index a

4.2 Resource allocation policy

The DPB consults the resource allocation policy and resource allocation table to determine whether a request of a process, acting on behalf of a user, should be granted or denied. Four major cases can be identified, where the request should be denied:

1. The user has already allocated too many resources
2. The process has already allocated too many resources
3. Too many of the particular resource have already been allocated
4. The request is acceptable, but no instances of requested resource are available

For the cases 1,2, and 3, it is necessary to specify the amount of available resources, maximum amount resources a user can allocate, and a maximum amount of resources an individual process can allocate. This leads to the definition 2.

Definition 2. Resource Allocation Policy (RAP) is a tuple (M, U, P, A) where M is a vector $\langle m_1, m_2, \ldots, m_l \rangle$ representing the numbers of l types of different resources within the system, U is a $l \times k$ matrix, specifying the number of resources that each user is authorized to allocate, P is a vector $\langle p_1, p_2, \ldots, p_l \rangle$ that represents the number of each resource a single process may allocate, and A is a vector $\langle \alpha_1, \alpha_2, \ldots, \alpha_l \rangle$ representing the maximum number of each resource that can be allocated in total.

Let R be the set of resources within a system and l be the number of types of resources available. Each resource can be uniquely identified, so $R = \{\rho_a,\ a = 1, 2, \ldots, l\}$. Let M be a vector $\langle m_1, m_2, \ldots, m_l \rangle$ where each m_a, $a \leq l$ determines the maximum number of resource ρ_a available. Let k be a number of users within the system. U will be specified as a $l \times k$ matrix, where $u_{a,b}$, $a \leq l$, $b \leq k$ is the maximum number of resources ρ_a that user b is authorized to allocated. P is a vector $\langle p_1, p_2, \ldots, p_l \rangle$ where each p_a, $a \leq l$ determines the maximum number of resource type ρ_a a single process can allocate. A is a vector $\langle \alpha_1, \alpha_2, \ldots, \alpha_l \rangle$ where each α_b, $b \leq l$ determines the maximum amount of resource ρ_b that can be allocated simultaneously.

To adapt the policy into the needs of multilevel security (MLS), the only modification required is to redefine P to be a vector $\langle p'_1, p'_2, \ldots, p'_l \rangle$. Assume, that there are c priority classes $\lambda_1, \lambda_2, \ldots, \lambda_c$, specified in the terms of allowed response time. Basically, this means that in the case of limited resources, available resources are allocated according to these priorities. Each p'_a, $a \leq l$ is then a tuple $\langle p^1_a, p^2_a, \ldots, p^c_a \rangle$ determining the maximum amount of resource ρ_a authorized to be accessed at priority level λ_a. To simplify the policy, it can be assumed that each user belongs to a particular group, and resource allocation rights U are determined according to the group membership. Establishment of a RAP is simply determination of M, U, P and A.

4.3 Resource allocation table

Resource allocation table is used to store the information about allocated processes, as specified in definition 3. RAT is used to store the number of each type of resource ρ_a allocated to different processes and can be used to calculate the total amount of allocated resources. Also, as the owner of each process can be tracked down, information about allocations need not to be stored separately.

Definition 3. Resource Allocation Table is a $l \times k$ matrix $RA(a,b) = r$, $a \leq l$ and $b \leq k$, where r is the number of resource ρ_a allocated to process b.

For efficient operation of the DPB, questions of how much resources of a given type area allocated to a given process, how much resources of a given type are allocated to a given user, and what is the total amount of allocated resources of a given type must answered. Determination of the first of this information is a direct read of RAT. Let $Owner$ be a function $Owner : ID \rightarrow UID$, where ID is the set of process identities within the system, and UID refers to the user identities within the system. Let function $Owner$ be specified so as in equation 1, where where uid is the owner of process with identity id.

$$Owner(id) = uid \tag{1}$$

Let function $User$ be a function $User : UID \times R \rightarrow N$, where $User(uid, a) = p$ so that p is the number of resources ρ_a allocated to user uid. On a given user identity uid and resource ρ_b, value of $User(uid, b)$ can be calculated as in equation 2.

$$User(uid, b) = \sum_{Owner(c)=uid} RA(b, c) \tag{2}$$

Let $Total$ be a function $Total : R \rightarrow N$ where $Total(a) = p$, so that p is the total number of resource ρ_a allocated to different processes. On a given resource ρ_a, where k is the number of processes, function $Total(a)$ can be easily calculated as in equation 3.

$$Total(a) = \sum_{b=1}^{k} RA(a, b) \tag{3}$$

4.4 Enforcement of the policy

Informally, DPB is to enforce function $\Theta : ID, R, N \rightarrow \{deny, grant, unknown\}$, where

$$\Theta(id, a, nr) = \begin{cases} deny & if\ request(id, a, nr)\ should\ be\ denied \\ grant & if\ request(id, a, nr)\ should\ be\ granted \\ unknown & if\ request(id, a, nr)\ is\ undecidable \end{cases}$$

The incoming request $request(id, a, nr)$ means that a process id, operating on behalf of $owner(id)$ is requesting to allocate nr units of resource ρ_a. To check

whether this should be granted, denied, or declared undecidable, three additional boolean functions must be specified. Function C_1, specified in equation 4 checks whether a resource allocation request would exceed the maximum number of resource ρ_a that can be allocated in total. Function C_2, as in equation 5 determines whether the requested allocation would exceed the maximum amount of resource ρ_a that can be allocated to a single process. Finally, comparison C_3 as in equation 6 determines whether there are enough of resource ρ_a available to satisfy the request.

$$C_1(a, nr, A) = \begin{cases} True \ \ if \ \alpha_a \leq Total(a) + nr \\ False \ if \ \alpha_a > Total(a) + nr \end{cases} \tag{4}$$

$$C_2(a, nr, P) = \begin{cases} True \ \ if \ p_a \leq Total(a) + nr \\ False \ if \ p_a > Total(a) + nr \end{cases} \tag{5}$$

$$C_3(a, nr, M) = \begin{cases} True \ \ if \ Total(a) + nr \leq m_a \\ False \ if \ Total(a) + nr > m_a \end{cases} \tag{6}$$

Now, $\Theta(id, a, nr)$ can be now specified as in equation 7, where Resource Allocation Policy $RAP = \langle M, U, P, A \rangle$ is specified as in definition 2, and C_1 is the result of $C_1(a, nr, A)$ specified as in equation 4, C_2 is the result of $C_2(a, nr, P)$ in equation 5, and C_3 is $C_3(a, nr, M)$ as specified in equation 6. If $\Theta(id, a, nr) = Deny$ or if $\Theta(id, a, nr) = Grant$, exit. Otherwise, $\Theta(id, a, nr) = Unknown$.

$$\Theta(id, a, nr) = \begin{cases} Deny & if \ (\neg C_1) \vee (\neg C_2) \\ Unknown & if \ (C_1) \wedge (C_2) \wedge (\neg C_3) \\ Grant & if \ (C_1) \wedge (C_2) \wedge (C_3) \end{cases} \tag{7}$$

Based on equation 7, an exact definition can be given to a Denial of service Protection Base, as in definition 4.

Definition 4. Denial of service protection base is a tuple $\langle RAP, RAT, \Theta, R_a, R_k \rangle$ where RAP is the Resource Allocation Policy, RAT is the Resource Allocation Table, Function Θ is a function $\Theta : R_a \rightarrow R_b$, that for each input vector R_a determines the output $R_k \in \{Deny, Grant, Unknown\}$.

4.5 Waiting time policy

Enforcement of waiting time policy that specifies the maximum throughput time for each process is the responsibility of both DPB and OS scheduling algorithm. The two major concerns of designing the system are provision of assurance that RAP does allow each process to allocate satisfactory amount of CPU time to successfully terminate within given time constraints and provision of assurance that OS scheduling algorithm is capable of satisfying WTP. As the latter issue is widely studied in OS research (see for example [6]) it will not be studied in detail within this paper.

Assume that each request contains a specification of the Maximum Waiting Time (MWT) that is the relative time in which the process must terminate.

MWT is composed of three factors: The time in waiting queues, the actual processing time and time of different context switches when the process state is changed. From the DPB point of view, the most important factor is the execution time of the process. The fundamental requirement is the correct transformation of time requirements into RAP. WTP can be seen as a higher level policy than RAP, so the two major factors, notation of WTP and transformation from WTP into RAP must be addressed.

WTP is seen as a specification of the maximum number of CPU quanta each user and an individual process acting on behalf of the user is allowed to allocate CPU. WTP is now seen as a set of tuples $\langle UID, T_m, T_s \rangle$ where user UID is authorized to allocate at maximum T_m quanta of CPU resource, so that any single process won't allocate more than T_s quanta. A formal specification is given in definition 5. $m_{CPU} \in M$ and $p_{CPU} \in P$ are components of RAP.

Definition 5. Waiting time policy $WTP = \{\langle UID, T_m, T_s \rangle\}$ where UID is an existing user identity, $T_m < m_{CPU}$ and $T_s < p_{CPU}$.

The system must attempt to satisfy as many requests as possible. As the cost of protection is controlled denial of some processes requests, a method is needed to study the feasibility of each request. A simple feasible transformation from WTP to RAP is to find the tuple $\langle UID, T_m, T_s \rangle$ from WTP where T_m and T_s are greatest and to specify RAP so that $U(CPU, ID) = T_m$ and $p_{CPU} = T_s$. This transformation guarantees that RAP allows each process request to be satisfied. Formal specification for a feasible transformation will be given in definition 6, where function Max is as specified in equation 8.

$$\begin{aligned} &Max(UID, M, S) = \langle id, m, s \rangle \in WTP \\ &\quad where\ \forall \langle ID, T_m, T_s \rangle \in WTP : (ID \leq id) \wedge (T_m \leq m) \wedge (T_s \leq s) \end{aligned} \tag{8}$$

Definition 6. WTP to RAP transformation is feasible when $U(CPU, ID) = T_m$ and $p_{CPU} = T_s$ where $\langle ID, T_m, T_s \rangle = Max(UID, M, S)$.

4.6 Undecidable requests

If a request is acceptable but can not be granted due to lack of resources, the DPB outputs *Undecidable*. To further analyze undecidable requests, an undecidability request policy (URP) and a mechanism to enforce that policy, Undecidability request base (URB), must be established. Undecidability can be handled either by ignoring undecidable requests or by forcing the system to release resources and grant the request. Both of these alternatives lead to a controlled denial of service. Which action must be taken is an administrative issue and depends on the types of tasks typically carried out within the system.

For the purposes of URB, we assume that a process is a tuple $\langle ID, O, \tau \rangle$, where ID is the identity of the process, O is a set of pairs $\{a, nr\}$ that indicates that the process occupies nr instances of resource type ρ_a, and τ refers to the

optional field of other properties that can be used by URB, such as priority. Each property of a process is indexed by process identity. On a given identity $id \in ID$, O_{id} refers to the resources occupied by the process id, and other properties are identified by τ_{id}. For an optional property p of a process id, the notation τ_{id}^{p} will be used.

Assume the set of all processes $PR = \{pr_a | a \leq k\}$, where k is the number of processes in the RAT. The URB will be seen as a tuple $\langle PR, URP, \Sigma, R_a, R_l \rangle$, where PR is the processes specified as above, URP is the undecidability request policy, Σ is a function $\Sigma : R_a \rightarrow R_l$ specifying the output R_a. Input vector R_a is of form (id, a, nr) where a process id is requesting nr instances of resource ρ_a. Output vector is of form $\{deny, f_{id',a',nr'}\}$ when the request will either be denied, or the decision is made to force process id' to release nr' instances of resource $\rho_{a'}$.

URP is seen a collection of tuples $\langle Precond, Op, Opcond \rangle$, that can be intuitively interpreted as *If condition Precond is satisfied, then action specified in Op should be taken so that condition Opcond will be satisfied.* It is, anyhow, not within the scope of this paper th analyze the issue further.

5 Performance analysis

Let k be the number of processes, and l the number of resources. This section provides with the analysis of the complexity of method proposed in this paper. This complexity is the overhead to be added on top of OS and TCB. The complexity of supporting functions $User$ and $Total$, as specified in equations 2 and 3, will be first analyzed in theorems 7 and 8.

Theorem 7. *The complexity of function $User$ is in $O(k + \log_2 l)$.*

Proof. Assume that the complexity of function $Owner$ is in $O(1)$ (we assume a hash table implementation) since it only consults the owner field of the process identity table. Therefore, the complexity $User$ is determined by the traverse of the RAT. On a given resource ρ_a, the first step is the search of the row of RAT that reflects that process with the minimum time for search $\log_2 l$. The other component is to traverse through each process and compare the $Owner$ field into the given uid. The complexity of this is k. Therefore, the total complexity is in $O(k + \log_2 l)$.

Theorem 8. *The complexity of function $Total$ is in $O(\log_2 k + l)$.*

Proof. Function $Total$ is simply calculated by sum product of all allocations of a given resource l. Searching for the requested process takes $\log_2 k$ units of time.

Based on these two theorems, the complexity of function Θ can be specified as in theorem 9. Based on Θ and previous results, the complexity of DPB can be specified as in theorem 10.

Theorem 9. *The complexity of function Θ, as specified in equation 7 is in $O(3\log_2 k + 3l)$.*

Proof. Function Θ is calculated using functions C_1, C_2, and C_3 as specified in equations 4, 5, and 6. Therefore, the complexity $O(\Theta) = O(C_1) + O(C_2) + O(C_3)$. The function $Total$ is the major component of each C_1, C_2, and C_3, their complexity is determined by it. Therefore, $O(C_1) = O(C_2) = O(C_3) = O(Total) = \log_2 k + l$. Then, $O(\Theta) = 3\log_2 k + 3l$.

Theorem 10. *The complexity of DPB is in $O(k + 4\log_2 k + 4l + \log_2 l)$.*

Proof. On the analysis of a request $req(id, a, nr)$, the first step is the identification of the $Owner(id)$, that has the complexity $O(1)$. Then, the system calculates functions $User$ and $Total$, that has complexities $O(k + \log_2 l)$ and $O(\log_2 k + l)$, respectively, as specified above. After these, the calculation of actually Θ can be performed, the complexity being $O(\Theta) = 3\log_2 k + 3l$. The total complexity of DPB is $O(Owner)+O(User)+O(\Theta) = O(k+\log_2 l)+O(\log_2 k+l)+O(3\log_2 k+3l) = O(k + 4\log_2 k + 4l + \log_2 l)$.

There is a clear time memory trade off. If storage space is used to store values of functions $User$ and $Total$, the computations of Θ can be reduced. By precalculating these functions for different inputs, the time complexity of C_1, C_2, and C_3 can be reduced to $\log_2 j$, and therefore the complexity of Θ reduces to $3\log_2 j$.

6 An example: SYN flooding

The client committing SYN flooding attack [3] initializes several connections to the victim server by sending TCP SYN packets. Once the SYN-ACK packet is received, the client leaves the connection 'half open' by not sending the expected ACK packet. This causes the list of initialized connections at the server to overflow and no further connections can be established until invalid connections are removed.

Assume that the data structure storing initialized but not fully established connections is resource ρ_a. The RAP will be specified so that $m_a = b$ where b is the size of ρ_a, that is maximum number of initialized connections. Since the origin of a TCP SYN packet can not be identified, there must be a special category of a user u_n, labeled 'Unknown' that means the owner of the SYN process is acting on behalf is not known. The maximum resource allocation matrix must be established $U(l \times k) = c$ where c is the maximum number of initialized connections. Obviously, it must be that $c \leq b$. As each process must be capable of allocating only one instance of ρ_a, it must be that $p_a = 1$, where $p_a \in P$. Also, the total number of resource ρ_a that can be allocation must be $\alpha_a = j$, $\alpha_a \in A$.

Each incoming SYN packet is passed via DPB where the comparison to RAT is made, that is $\Theta(id, a, 1)$ is calculated. If $\Theta(id, a, 1) = Grant$ then the

request will be passed to other parts of TCB and the allocation is made unless other reasons exist for denying. In the case where $\Theta(id, a, 1) = Undecidable$, the control must be passed to URB.

The resource allocation table is maintained in two cases. First, when a SYN packet arrives and the request can be granted. This is the case in normal circumstances when an allocation is added to the RAT. Second case is after a successful allocation of resources for SYN request, when the corresponding ACK packet is received. The obvious problem with this approach is that all incoming requests must be stored in a queue while DPB is processing the request. Therefore, the solution may lead to the denial of service where the wait queue overflows. Since the DPB performs only a limited task, resources can be devoted to it to prevent the flooding of DPB itself.

The problem can be reduced by enhancing the fundamental solution above. If a rough categorization will be made between requests originating from internal and external sources, and user u_n, unknown, will be replaced by two users (or roles): u_n, external, and u'_n, internal. RAP can now be modified to enforce certain amount of resources to be allocated only for internal requests and certain amount for external requests. If this method is combined with input filtering, where packets with source address that points to the internal network are discarded, the likelihood of complete denial of service can be reduced.

7 Conclusions and future work

A method to specify and enforce a resource allocation policy (RAP) to prevent denial of service attacks have been presented in this paper. The method assumes that a formal policy can be established based on a maximum waiting time policy (WTP) that is a tool to specify maximum acceptable response times for different operations. Formal mapping from WTP to RAP enables checking of correctness and comprehensiveness. Though, the focus of this paper has been on the establishment of the model and not in the specification of tools to support analysis of the model. There are also upper level issues that have not been comprehensively analyzed within this paper, such as specification of the system behavior in undecidable requests.

The cost of protection against denial of service consists of two major factors: computational cost of RAP enforcement algorithms and controlled denial of requests in order to maintain system's availability. The cost of algorithms has can be formally analyzed, and it doesn't cause a significant overhead to the operation system and other parts of trusted computing base. The cost of controlled denial of services, on the other hand, is more abstract and difficult to measure. Further research is required to specify a notation for undecidable request base to take into account urgency of different requests, and to establish procedures and routines that enforces this notation in an optimal manner.

Typical denial of service attacks that have been recently discussed exploit vulnerabilities in operating systems and network protocols, and therefore not only operating system design methods can provide adequate protection. Com-

prehensive protection requires actions taken on all areas of computer security, and methods to integrate these measures to provide assurance of security and to reduce the cost of protection by identifying multiply enforcement of protection. The model presented in this paper provides a starting point for analysis and further research.

References

1. International standard ISO 7498-2. information processing systems - Open systems interconnection - Basic reference model - Part 2: Security architecture, 1988.
2. Denial of service attack via ping. CERT Advisory CA-96.26, December 1996. Available at ftp:info.cert.org/pub/cert_advisories/CA-96.26.ping.
3. TCP SYN flooding and IP spoofing attacks. CERT Advisory CA-96.21, September 1996. Available at ftp:info.cert.org/pub/cert_advisories/CA-96.21.tcp_syn_flooding.
4. M. Adabi, M. Burrows, B. Lampson, and G. Plotkin. A calcululs for access control in distributed systems. In J. Fagenbaum, editor, *Advances in Cryptology-Crypto'91*, LNCS 576. Springer-Verlag, 1991.
5. D. Bailey. A philosophy of security management. In M. D. Abrams, S. Jajodia, and H. J. Podell, editors, *Information Security - An Integrated Collection of Essays*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995.
6. H. Chetto and M. Chetto. Some results of earliest deadline first algorithm. *IEEE Transactions on Software Engineering*, 15(10):1261–1269, 1989.
7. J. Glasgow, G. MacEwen, and P. Panangaden. A logic for reasoning about security. *ACM Transactions on Computer Systems*, 10(3):226–264, August 1992.
8. V. Gligor. A note on the denial-of-service problem. In *1983 IEEE Symposium on Research in Security and Privacy*, 1983.
9. M. Harrison, W. Ruzzo, and J. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.
10. K. J. Keus and M. Ullman. Availability: Theory and fundamentals for practical evaluation and use. In *Proceedings of the 10th Annual Computer Scurity Applications Conference*, 1994.
11. J. Laprie. *Dependability: Basic Concepts and Terminology in English French, German, Italian and Japanese*. Springer-Verlag, 1992.
12. J. Leiwo and Y. Zheng. Layered protection of availability. In *Proceedings of the 1997 Pacific Asian Conference on Information Systems*, Brisbane, Australia, April 1997.
13. J. K. Millen. A resource allocation model for denial of service. In *1992 IEEE Symposium on Research in Security and Privacy*, Oakland, California, May 1992.
14. R. Needham. Denial of service. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, 1994.
15. D. B. Parker. A new framework for information security to avoid information anarchy. In *Proceedings of the IFIP TC11 11th international conference of Information Security*, Cape Town, South Africa, May 1995.
16. A. Reed. Computer disaster: The impact on business in the 1990s. In *Proceedings of the IFIP TC11 8th International Conference on Information Security*, Singapore, May 1992.

This article was processed using the LaTeX macro package with LLNCS style