Resilient Reference Monitor for Distributed Access Control via Moving Target Defense

Dieudonne Mulamba and Indrajit Ray(✉)

Department of Computer Science,
Colorado State University, Fort Collins, CO 80523, USA
{indrajit,mulamba}@cs.colostate.edu

Abstract. Effective access control is dependent not only on the existence of strong policies but also on ensuring that the access control enforcement subsystem is adequately protected. Protecting this subsystem has not been adequately addressed in the literature. In general, it is assumed to be implemented as a reference monitor in a trusted computing base (TCB) that is tamper-proof. However, in distributed access control, ensuring TCB security kernel to be tamper proof is not always feasible. It needs to be implemented in software and on platforms that can potentially have vulnerabilities. We posit that allowing a very limited opportunity to the attacker to enumerate exploitable vulnerabilities in the access control subsystem can considerably facilitate its protection. Towards this end we propose a moving target defense framework for access control in a distributed environment. In this framework, access control is provided by cooperation of several distributed modules that materialize randomly, announce their services, enforce access control and then disappear to be replaced by another module randomly. As a result, the attacker does not know which process can be targeted to compromise the access control system.

1 Introduction

Many emerging distributed applications rely on on-demad network-enabled access to a shared pool of computing resources. Examples of such applications are IoT applications, sensor networks or enterprise level distributed workflow systems. This distributed computing model brings with it some unique challenges to access control that require re-visiting the traditional TCB approach [3]. In traditional systems, access control is implemented by the cooperation of four functional modules that are part of the trusted computing base:

1. *Policy Administration Point* (PAP): The PAP is a repository for the authorization policies that are expressed in terms of the actions that subjects (human users, devices, processes, organizations etc.) can take on various objects in the system. The authorization policies are essentially an instantiation of the access control model tailored towards the organization. It is the main component for the authorization portion of access control.

G. Livraga and S. Zhu (Eds.): DBSec 2017, LNCS 10359, pp. 20–40, 2017.
DOI: 10.1007/978-3-319-61176-1_2

2. *Policy Information Point* (PIP): The PIP is the component that gathers together all the attribute information that are needed to evaluate an authorization policy.
3. *Policy Decision Point* (PDP): The PDP gets relevant information from the PIP and consults the PAP to arrive at a decision whether to grant or deny an access request.
4. *Policy Enforcement Point* (PEP): The PEP receives access requests from subjects in the external world, hands them to the PDP for evaluation, and after receiving the grant or deny response from the PDP, ensures the appropriate action is taken.

One of the requirements of a TCB is that it implements the concept of reference monitor [3]; that is, the TCB mediates all access to objects by subjects, is tamper-proof and cannot be bypassed, and is small enough to be thoroughly tested and analyzed. In the newer distributing computing environments however, making the TCB small and tamper-proof is very difficult. This is because access control in such environment needs to be achieved via the cooperation of both local as well as remote access control engines. To remain within the confines of the TCB paradigm, not only all of these separate components need to be made tamperproof, but also all communications and coordinations among the components. As a result, the trusted computing base needs to be enlarged in scope and functionality, which violates the principles of reference monitor. Moreover, the sheer size of these distributed systems, the degree of heterogeneity among the different devices (potentially virtual machines), and the dynamicity of the whole system, compound the problem many fold. It appears, therefore, that it is next to impossible to rely on a single TCB to provide access control in these environments. The access control subsystem should try to satisfy as many properties of a TCB as possible but should also incorporate certain self-defending strategies to make it secure.

In this work, we treat access control in such a distributed environment as a service that needs to be proactively protected. From a functional perspective, this service is achieved by the four functional units – PAP, PIP, PDP and PEP. We assume that like any other service the access control service can be attacked by an attacker and hence needs to be protected. An attacker intent on damaging the access control service will launch reconnaissance efforts seeking exploitable vulnerabilities for this subsystem. We propose being proactive and allow only limited opportunity to the attacker to enumerate exploitable vulnerabilities, thus reducing the attack surface of the access control subsystem. Towards this end, we propose employing a Moving Target Defense (MTD) paradigm for the access control subsystem. The four functional modules are effectuated by randomly materializing processes that announce their services, enforce access control and then disappear to be replaced by another module randomly. As a result, the attacker does not know which processes can be targeted to compromise the system. Moreover, the window of opportunity for targeting processes is varied to further reduce opportunities of attack. We describe an implementation of this system to handle RBAC policies using COTS components.

The rest of the paper is organized as follows: Sect. 2 reviews previous works on access controls, leader election protocols, moving target as well as on service discovery protocols. In Sect. 3 we first present the reference architecture for access control in distributed environments. We then give an overview of our moving target defense approach for protecting the access control subsystem. Section 4 presents the moving target defense architecture. We present our implementation in Sect. 5 as well as an analysis of the security of the proposed approach. Finally, we conclude this paper in Sect. 6 and give some directions for future works.

2 Background and Related Works

2.1 Protection of Access Control Subsystems

One of the most important aspect of security is ensuring that users access only resources to which they are authorized. Research on designing and deploying access control in computers and networks can be traced back several decades [43]. Early standards of access control included discretionary and mandatory access control [13,17,34,42]. However, Role based Access Control (RBAC) represented a major leap forward in term of flexibility. RBAC is built on the principle that users do not have discretionary access to enterprise objects. In a RBAC model, roles are created and users belong administratively to these roles, while permissions are administratively assigned to the roles. This arrangement provides more flexibility and simplicity to the management of authorization [13]. RBAC has been traditionally implemented for centralized systems. In recent years, several works have been done to provide the capabilities of this access control model to distributed systems and the cloud. For instance, in [21] authors present an access control tailored for distributed control systems. [41] explains how one can provide access control to anonymous users while verifying their authorization in a decentralized manner.

In several works, including recently in [14], researchers have worried that a malicious program may tamper with the operation of an access control system. The notion of a trusted computing base implementing the reference monitor concept was proposed by Anderson [3], in order to address this problem. Security kernels such as Scomp [15] and GEMSOS [45], included a reference validation mechanism to satisfy the reference monitor concept of the TCB. Other operating systems such as Trusted Solaris [36], the Linux Security Modules (LSM) framework [52], TrustedBSD [49], Mac OS X, and the Xen hypervisor provide various degree of support for reference validation so as to enable some shade of reference monitor. However, the major problem with these systems is that the tamperproof property that needs to be ensured for provably implementing a reference monitor, is hard to achieve. Tamperproofing requires the TCB to have a very small footprint. It can be shown that a general algorithm to prove that an arbitrary program behaves correctly reduces to solving the Halting problem. While current algorithms can prove correctness properties of specific programs, the variety of reference validation code and the complexity of correctness properties preclude verification for all but the smallest, most specialized systems.

Unfortunately, for most of these systems, the TCB is too large to determine whether tampering is prevented. Moreover, for practicality and functionality, many systems allow user-level processes to modify the kernel. However, none of these user-level processes are immune to tampering, thus becoming one of the weakest links. In this work, we are interested in the protection of the access control subsystem where ensuring the tamperproof property of a reference monitor is challenging.

2.2 Moving Target Defense

Several works have been done on Moving Target Defense (MTD). In order to improve the understanding of MTD, [54] presents key concepts and their basic properties. Other works on MTD are mainly focused on network-based MTD [2,4,10]. In addition, in [12], the effectiveness of MTD using low-level techniques to defend a computing system has been studied. Those low-level techniques include Address Space Randomization, Instruction Set Randomization, and Data Randomization. In [19] two measures are designed that allow a defender to quantify its gain in security while deploying a MTD system. In order to chose a particular MTD technique, one needs to know its effectiveness. For that purpose, [53] proposes a comparison of different MTD techniques based on their effectiveness. However, none of these techniques are applicable for the problem we are addressing.

2.3 Leader Election

In a distributed system, leader election is a fundamental problem that requires that a unique leader be elected from among a set of given nodes. The goal of a leader election algorithm is to elect a good processor as a leader in a setting where there are n processors among which a certain number $m < n$ are bad, while ensuring that no bad processor get elected as a leader [27].

A distributed computing system often requires that active nodes continue performing their task after a failure has occurred. This reorganization or reconfiguration necessitate that a coordinator be elected in the first place [16]. This is the reason of the wide interest the leader election problem [31] has received. Several works have been done on Leader Election [1,6,16,44,47].

2.4 Consensus Algorithms

One approach for building fault-tolerant applications is the Lamport's approach. The core of this approach involves two primitives: consensus and atomic broadcast [28]. Leader election protocols are generally used to solve the consensus problem.

Bully algorithm [5] and Ring algorithm [48] are among the most used algorithms for solving the consensus problem. A hugely popular algorithm is the

Paxos algorithm proposed by Lamport [30]. Another popular algorithm, considered even superior to Paxos due to its simplicity, is the Raft consensus algorithm [20]. Raft provides the capabilities for Leader election and log replication.

ZooKeeper [22], an open-source replicated service for coordinating web applications, and Chubby [7] are some practical systems exploiting these algorithms. Recently, GIRAFFE [46] has been proposed to provide a coordination service in scalable distributed system. Another practical system is the Apache Kafka [51] which allows the building of a replicated logging system. However, these algorithms and protocols do not take into account Byzantine failures. In addition, their approach for electing a new leader does not prevent a malicious host from being elected as a leader.

2.5 Byzantine Fault Tolerance

A computer system can be affected by a type of failure that can cause it to behave in an arbitrary way. After being affected, the computer system can be led either to process requests incorrectly, to corrupt their local state, and/or to produce incorrect or inconsistent outputs. This type of failure is called Byzantine failures. The problem for coping with this type of failure is known as the Byzantine Generals Problem [29]. The goal of Byzantine fault tolerance is to allow computer system to be immune against Byzantine failures.

Several works have been proposed to reach a consensus in the face of Byzantine failures. Building on Paxos, the authors in [9] have proposed an improvement that allows Paxos to support Byzantine fault tolerance with a modest latency. Castro and Liskovs proposed the Practical Byzantine Fault-tolerance protocol [8] that reduces the number of messages exchanged to only four messages. [33] looked at improving the number of communications in the Byzantine Paxos protocols. In [11], authors took the task of improving Raft to support Byzantine fault-tolerance. They reach their goal by combining the ideas from the original Raft algorithm and from Practical Byzantine Fault-tolerance protocol [9].

2.6 Service Location Protocol

A zero configuration approach is a self-management networking approach that allows network devices to be automatically configured, discover services automatically, and to access service without the involvement of a network specialist [Intelligent Self-Management Home Multimedia Service System]. Three major self-management technologies have been proposed. The Internet Engineering Task Force (IETF) promoted SLP [18,38] as an intranet standard for automatic network resource discovery. Intel and Microsoft, on their part, proposed the Universal Plug and Play (UPnP) [39] as a standard for automatic communication between network devices using XML messages. Apple Inc. proposed a protocol called Bonjour [23] as its zero configuration networking standard. Recently, z2z [32] has been proposed for the discovery of network services beyond the local network.

In this work, our contribution includes the development of a system that leverages these existing concepts into a single framework so as to address the problem of protecting a reference monitor.

3 Architecture Overview

We assume that the access control model is Role-Based Access Control. We start with a high level operational architecture of access control (AC) in the distributed setup. The resources we are concerned about are the shared resources. The AC architecture is composed of four functional entities. Each entity has specific functions and participates in the communication as a client, as a server or both.

3.1 Access Control Architecture Components

The four entities are:

1. *A users client*: It is an endpoint entity whose main objective is to access protected resources. It is responsible to initiate or terminate a session with the Resource Access Server. It resides on the individual devices running the applications that require access to shared resources.
2. *Resource Access Service (RAS)*: It is an entity that manages the various distributed resources and controls the access to it. It acts as both client and server when receiving or replying to access requests from the client. The decision to grant or deny the access to protected resources is received from the Authorization Control Server. The Resource Access Server is responsible for reinforcing that decision.
3. *Authorization Control Service (ACS)*: It is an endpoint entity that governs the access to each protected resource. It hosts the Access Control engine. The access control engine is based on RBAC model (other models are also possible) and is designed to prevent unauthorized access to protected resources. The policies defining the access to protected resources are also managed by the Authorization Control Service. This service receives any client request and replies with the decision to grant or deny the access to the needed resources.
4. *Discovery Service*: Since the protection of the Authorization Control Service requires this later to be moved to a different location in a non predictable manner, the clients need to be able to discover the new location of the Authorization Control Service. The Discovery Service provides such capability.

Using these entities, the distributed access enforcement proceeds as follows. When a client needs to access a protected resource, it sends a request to the Authorization Control Service (ACS). The ACS verifies the policy governing the access to the needed resources, and replies with a decision to grant or deny access to the requested resource. If the decision was to grant access, the request is forwarded to the Resource Access Service along with a limited life-time token given to the client allowing it to access that particular resource. The occurrence of an election triggers the Authorization Control Service to be Switched to a different location. In this case, the ACS will register its new location with the Discovery Service. In addition, the client will need to consult the Discovery Service in order to find the new location of the ACS.

3.2 Threat Model

In our architecture, we consider access control (AC) as a service and we assume that the Access Control Service can be attacked and compromised. To mitigate the vulnerability of the open network in which an attacker passively listens to various communications, we make all access related communications go over secure channels. This makes the communication secure, but does not protect the endpoints, particularly the Resource Access Server and the Authorization Control Server where the AC engine components reside. We assume that an attacker can masquerade as one of these servers. Alternately, some numbers of these servers can themselves be compromised by malware and behave in a Byzantine manner. An attacker masquerading as a valid server or corrupting a server are treated similarly.

To protect these two entities, we propose the Moving Target Defense strategy. Our motivation for this approach follows from the observation that an attacker needs a reconnaissance window to explore the vulnerabilities in a system before attacking it. The moving target defense strategy reduces this window of opportunity. It requires both the Resource Access Server and the Authorization Control Server to be replicated. At each instance there is only one Resource Access Server and one Authorization Control Server that is responsible to handle access requests. These are called respectively, RAS leader and ACS leader servers. In addition, both leader servers are periodically replaced by a pair of leader servers randomly chosen by following a Byzantine consensus process executed by the existing candidate RAS and ACS servers. The replacement is achieved through a migration process that relies on a secure service discovery process.

Using moving target defense in this manner to protect the Access Control Engine raises several challenges. Those challenges are as follows.

1. How does one avoid migrating a leader server to a malicious server during the migration process?
2. Which access control component is going to be migrated? And,
3. After the migration process, how does one discover which server is currently providing the services?

In the following sections we are going to address these challenges (Figs. 1 and 4).

4 Distributed Access Control Architecture

We present an architecture that provides access control services. In order to protect the access control service against certain attacks, we have proposed to implement a Moving Target Defense strategy on the access control service architecture. In this section, we present the different components that constitute our Moving Target Defense architecture.

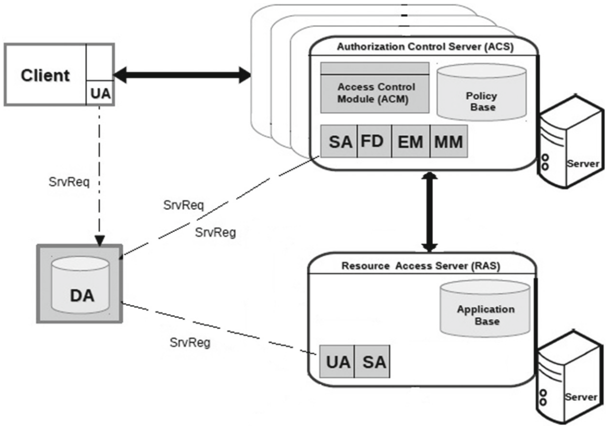

Client
UA
SrvReq
SrvReq
SrvReg
DA
SrvReg
Authorization Control Server (ACS)
Access Control Module (ACM)
Policy Base
SA
FD
EM
MM
Server
Resource Access Server (RAS)
Application Base
UA
SA
Server

Fig. 1. Moving Target Defense architecture

4.1 The Client

It is an endpoint entity whose main objective is to access protected resources. It is responsible to initiate or terminate a session with the Resource Access Service. It resides on the individual devices running the applications that require access to shared resources.

4.2 The Authorization Control Service

In this section we present the different components that allow the Authorization Control Service to provide access control service, to be elected as a leader, and to announce its services once elected as a leader.

Access Control Engine. The access control engine is based on RBAC model (other models are also possible) and is designed to prevent unauthorized access to protected resources. It comprises the Policy Enforcement Point (PEP), the Policy Decision Point (PDP), the Policy Administration Point (PAP), and the Policy Information Point (PIP).

Fault Detector Module (FD). The Fault Detector Module is designed to detect the byzantine faults occurring in the server providing the Access Control service. The failure and the compromise of the current leader server are reasons to trigger the Moving Target Defense. The Fault Detector is able to detect any kind of byzantine faults that are local to the leader server. The unavailability of the leader server is detected by the Fault Detector of any other server that probes the aliveness of the leader server. The occurrence of either failure, compromise, or unavailability is used to trigger the election of a new leader server that will be responsible to provide access control services.

Election Module (EM). The Election Module is responsible for processing the election of a new leader server. Several causes can trigger this election. Since the leader server is assuming this function for a limited time, called here a term, the expiration of this term is a cause that triggers the election of a new leader server. We add randomness to the duration of this term in order to prevent an attacker from correctly guessing the occurrence of the next leader election. At the expiration of his term, the current leader server proceeds to the election of a new leader. In other circumstances, the first server noticing the failure of the current leader server, is responsible to proceed to a new election.

Leader Election Protocol. In our system, after having opted to replicate the Authorization Control Service among several servers, a single server is responsible to provide this service at any given time. We call this server the leader. At any instance, this leader may be the subject of attacks or of failures. Thus to realize the moving target defense, the leader is required to be periodically changed. This change can occur at the expiration of the current leader's lifetime or when the current leader fails. Moreover, the next server is not pre-determined but is elected by existing servers, each of which is a candidate. All this is done in an environment where we assume that some servers may be malicious attackers. Thus, we need a leader election algorithm that can ensure that a malicious server cannot be elected as leader. Once a server is elected, it sets a random lifetime for itself.

For the sake of maintaining our distributed system in a good functioning state, it becomes crucial to prevent faulty nodes from becoming leader. We adapt the algorithm from [26] in order to realize a leader election. The election process proceeds in the form of a distributed protocol as follows.

The election algorithm proceeds in rounds and in each round there is a node that is coordinating the consensus, called the coordinator. For each round r there is a coordinator c known a priori by each participating node by computing $c \equiv (r \ mod \ n) + 1$ with n being the total number of nodes. At each node, there are several local variables that are maintained, among which there is the estimate value which is an input value selected by the node, its current election round, its current coordinator c_p, and a timestamp ts_p. The consensus algorithm runs by exchanging messages between nodes participating in the distributed system. These messages include the types ESTIMATE, SELECT, CONFIRM, READY/NREADY, and SUSPECT. The algorithm runs in a sequence of five tasks that are concurrently executed.

The consensus algorithm, (see Algorithm 1), works as follows. The algorithm starts with each node p picking its estimate of the input value, and sending an ESTIMATE message to all nodes. The coordinator, after receiving $n - k$ ESTIMATE messages that it was waiting for, selects a value es based on all the estimate values received. It then sends a SELECT message carrying the es value to all nodes. Each node p, upon receiving SELECT message from the coordinator, sends a CONFIRM message carrying the es value to all other nodes. The es value should be the same for a given round r. After receiving a CONFIRM message from $\lfloor (n + k)/2 \rfloor + 1$ distinct nodes, each node p updates its local variables.

It then sends a READY message or a NREADY message depending on whether it had received the same *es* value or not from $\lfloor(n+k)/2\rfloor+1$ CONFIRM messages. It should be noted that if the CONFIRM messages received by a node p did not contain the same *es* value, p will assume that the coordinator had deviated from the algorithm. The node p will therefore add the coordinator to its list of Suspect, and will send a SUSPECT message containing the *id* of the suspected coordinator to all. After a node p received the same *es* value as content of READY message from $\lfloor(n + k)/2\rfloor + 1$ distinct nodes, it will decide on that value. A node q is confirmed to be malicious by a node p and added to $Output(D)_p$ if and only if node q have been reported malicious by at least $k + 1$ nodes. $Output(D)_p$ is the final list of malicious nodes. Any round in which the coordinator has not been reported with malicious nodes will end with a consensus on the input value and the coordinator being confirmed as the new leader. Otherwise, a new round will start with a new coordinator.

Migration Module (MM). The Migration Module is responsible for executing the migration protocol. The Migration Module receives a notification from the Election Module that a new leader has been elected. This notification contains the identity of the new leader server. Upon receiving the notification from the Election Module, this module executes the migration protocol, which transfers the Access Control service to the new elected leader server.

Service Migration Protocol. In our architecture, replacing the current leader server by a new one necessitates the migration of the Authorization Control Service provided by the current leader server to the new one. For this purpose, we need to put in place a service migration protocol that can handle this task.

Process migration is the movement of a running process from one host to another. A process migration protocol can have several components like the transfer policy, the selection policy, and the location policy. Since we are interested with the migration of an access control service, we define these policies in terms of the requirements of the access control service.

1. *The Transfer Policy:* This policy determines when a host needs to send a process to another host. In our case, it determines when the current leader server needs to send the access control services to a newly elected leader server. In our architecture, this decision is triggered by the successful completion of the leader election protocol.
2. *The Location Policy:* This policy determines the destination host to which to transfer the process to be migrated. In our architecture, this information is provided by the leader election protocol that communicates the identity of the new elected leader server to all the hosts. This new Authorization Control Service is the destination host for the migration protocol.
3. *The Selection Policy:* Determines which resource to transfer. It is question here to determine which component of the access control service needs to be migrated. We have designed the Authorization Control Server to be fully replicated. We assume the existence of a replicated protocol. Therefore, the

Algorithm 1. Leader Election

```
/* Each node $p$ executes the following */
/* Initialization */
$e_p = V_P$ {Chosen value}
$r_p = 0$ {Initial round}
$ts_p = 0$ {Initial timestamps}
$Estimates_p = \emptyset$ {list of estimate msg}
$Confirms_p = \emptyset$ {list of comfirm msg}
$Suspected_p = \emptyset$ {list of suspected nodes}
$Output_p = \emptyset$ {blacklisted nodes}
for $r$ in $listnode$ do
  $Suspecting_p[r] = \emptyset$
end for
COBEGIN {Concurrent tasks}
{Task 1:}
while true do
  $r_p \leftarrow r_p + 1$
  {Select a Coordinator $c_p$}
  $c_p \leftarrow (r_p \bmod n) + 1$
  {Task 1, Phase 1 : each node creates estimate msg}
  $estimate_p = (ESTIMATE, p, r_p, e_p, ts_p)$
  Send $estimate_p$ to all
  {Task 1, Phase 2: Coordinator counts received estimate msg}
  if [$p = c_p$] then
    [Wait until received $(n - k)$ distinct $estimate_q$ messages from $q$ nodes]
    $Estimates_p \leftarrow estimate_q$
    $ts$ = largest $ts_q$ : $estimate_q \in Estimates_p$
    if [$ts = 0$ and (at least $(k + 1)$ distinct $estimate_q \in Estimates_p$ have common value $e$)] then
      $es \leftarrow e$
    else
      $es \leftarrow e_p$
    end if
    {Coordinator creates select msg}
    $select_p = ((SELECT, p, r_p, e_s)_p)$
    Send $select_p$ to all
  end if
  {Task 1, Phase 3 : receiving confirm msg}
  [Wait until received $[(n+k)/2]+1$ distinct $confirm_q$ messages or $c_p \in Output_p$]
  $confirm_p = ((SELECT, p, r_p, e_s)_p)$
  if [$[(n + k)/2] + 1$ distinct $confirm_q \in Confirms_p$ have common value $e$] then
    $ts_p \leftarrow r_p$
    $e_p \leftarrow e$
    $Confirm_p \leftarrow (CONFIRM, q, r_p, e)_q$
    $ready_p = (READY, p, r_p, e)_p)$
    Send $ready_p$ to all
  else
    $nready_p = (NREADY, p, r_p, e)_p$
    Send $nready_p$ to all
  end if
end while
{Task 2: }
while true do
  if [p received $select_p$ msg from $c = (r_p \bmod n) + 1$ and p has not sent $confirm_q$ msg] then
    $Selects_p \leftarrow ((SELECT, c, r, e, ts)_c)$
    $confirm_p = (CONFIRM, p, r, e)_p$
    send $confirm_p$ to all
  end if
end while
{Task 3:}
while true do
  [Wait until received $[(n + k)/2] + 1$ distinct $ready_q$ messages from $q$ nodes with common $r$,$e$]
  decide(e)
end while
{Task 4:}
while true do
  {Send list of suspected nodes}
  $Suspected_p \leftarrow D_1$
  $suspect_p = (SUSPECT, p, Suspected_p)_p$
  Send $suspect_p$ to all
end while
{Task 5:}
for $r$ in $S$ do
  When p receives $Suspect_q$ from $q$
  if $r$ in $Suspected_q$ then
    $Suspecting_p[r] \leftarrow Suspecting_p[r] \cup (q)$
  else
    $Suspecting_p[r] \leftarrow Suspecting_p[r] - (q)$
  end if
  if $|Suspecting_p[r]| \geqslant k + 1$ then
    $Output_p = Output_p \cup (r)$
  else
    $Output_p = Output_p - (r)$
  end if
end for
```

discussion about the replication protocol is beyond the scope of this paper. Being fully replicated, each Authorization Control Service has the same PDP, PAP, and PEP. There is no need to migrate those entities. However, only the current leader server has the information about the granted access requests in addition to having the most up to date policy database. Granted access requests information is stored in the session history of the current Authorization Control Service. Therefore, the session history and the policy database need to be migrated to the new Authorization Control Service to allow users with granted access a continuous use of the allowed resources.

4.3 The Discovery Service

Implementing a Moving Target Defense for an Access Control service requires switching frequently but randomly the server that is responsible for offering the Access Control service. Once the Authorization Control service has been switched to a newly elected leader server, users need a way to rediscover the new server offering the service. In this section, we adapt the Service Location Protocol or SLP in order to enable users to discover the new location of the services they need.

SLP Module. The SLP Module is responsible for running the service discovery protocol. This service is provided to clients that need to consult the SLP Module in order to discover the new Authorization Control Service location.

SLP Overview. Service Location Protocol (SLP) is a protocol designed by the Internet Engineering Task Force to eliminate the need of a manual configuration from users of communication networks in order to discover services, applications, and devices available in those networks. Since users, mainly mobile users, increasingly experience changing environments and the fact that the Internet has became more service oriented, service location is becoming more helpful in today's complex networks [18,50].

SLP Architecture. The SLP framework includes three main components called "Agents" to process SLP information. These agents are: User Agents (UA), Service Agents (SA), and Directory Agents (DA).

- *User Agents (UA)*: They are responsible for requesting services on behalf of the users or applications.
- *Service Agents (SA)*: They are entities that advertise the location and description of services on behalf of services. To advertise services, the SAs embed the service information into an URL. These information include the IP address, the port number, the service type and the path. Each service type is characterized by specific attributes along with their default values. All of these are specified by the Service Templates.
- *Directory Agents (DA)*: They are central repositories that aggregate SLP information. Since service information are embedded in a URL, these URL are stored by the DA which provides them to any UA that have issued a request which matched some attributes in the URL.

At the beginning of the protocol, any service provider needs to advertise its services. For that purpose, its SA registers the service with the DA. This step is known as the Service Registration. The DA acknowledges the registration by issuing a service ACK message to the SA. A user that needs to use

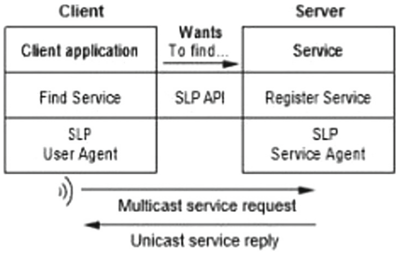


Fig. 2. Service discovery flow

the given service needs to have his UA issuing a query with the appropriate attributes to the DA. This is known as the Service Request step. The DA may reply back to the UA with the address and characteristics of the desired service (Service Replay). This is the general approach of the SLP protocol as illustrated on Fig. 2.

There is a major issue with the general approach of the SLP protocol as explained above. No one from both the UA and the SA knows the address of the DA. Before registering a service with the DA, the SA needs to discover the existence of the DA. The same thing applies to the UA.

Three different methods are used to discover the location of the DA: static, active, and passive. When using the static discovery method, both the UA and the SA learn the address of the DA from a DHCP server. In case of an active discovery, SLP agents contact the DA by sending service requests to the SLP multicast address on which the DA is configured to listen to for incoming communications. Upon receiving a service request, DA responds directly to the requesting agent via the agent's unicast address. The passive discovery method involves DAs periodically advertising their existence through the SLP multicast address. The other SLP agents discover the DAs location after listening the multicast advertisements. They can then contact the DAs directly through their unicast addresses for other operations [37].

Besides the basic SLP architecture involving SAs, DAs, and UA, it is possible to set a SLP architecture without DAs. In this case, UAs and SAs need to communicate directly to each other. In order to discover available services, UAs repeatedly send out their service requests to the SLP multicast address. On the other hand, SAs are listening for incoming requests on the SLP multicast address. Upon receiving a request corresponding to a service they are advertising, SAs reply through unicast address to UAs.

4.4 The Resource Access Service

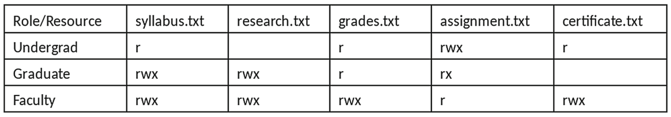
It is a server that manages the various protected resources. It acts as both client and server when receiving or replying to access requests from clients. The decision to grant the access to those resources is received from the Authorization Control Service.

5 Implementation

In this section, we introduce the proof-of-concept implementation of our proposed architecture. We have designed a test case to exemplify the functioning of the protocol.

5.1 Clients and Resource Access Service

We assume that we have a set of users represented by client applications. Those users are grouped into a set of roles (Undergraduate, Graduate, and Faculty). We also have a set of resources, in this case files stored on a file server. This file server constitutes our Resource Access Service. We have also defined a set of actions to be performed on those files by users. We have chosen basic Linux actions: Read, Write, and Execute. The different permissions given to users over those files are represented on Fig. 3.



Role/Resource	syllabus.txt	research.txt	grades.txt	assignment.txt	certificate.txt
Undergrad	r		r	rwx	r
Graduate	rwx	rwx	r	rx	
Faculty	rwx	rwx	rwx	r	rwx

Fig. 3. Roles-Permissions assignment

5.2 Authorization Control Server

Using socket programming, we have implemented five Authorization Control Services named $ACS1$, $ACS2$, $ACS3$, $ACS4$, and $ACS5$. Those have been implemented as Java client-servers. Each of those Authorization Control Services has an Access Control Module that is responsible for verifying user's authorization to resources that are stored on the Resource server. At each time, only one Authorization Control Service is responsible for providing the access control service.

Access Control. We start our process with the Authorization service being provided by, let us say, the Authorization Control Server ACS1. We consider that user Tom submits a request to read a file named certificate.txt. This request is intercepted by ACS1 which run the Access Control Module. The Access Control Module has been implemented using the Balana [24] open source implementation of XACML. Tom's query will be a tuple user-id, action, resource-name where in this particular case user-id is Tom, action is read, and resource-name is the file certificate.txt requested by Tom.

Policy Enforcement Point. Tom's query is intercepted by the Policy Enforcement Point (PEP). In fact, the PEP intercepts all queries sent to the Resource Access Service [40]. We have developed a wrapper that converts the original query into a XACML request. The request is then sent to the PDP for verification. Figure 5 illustrates the form of the XACML request.

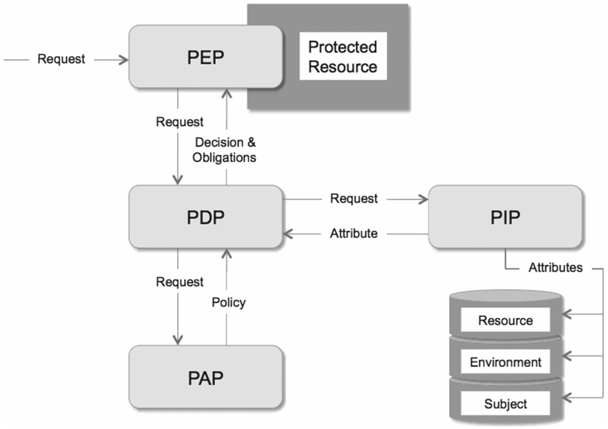
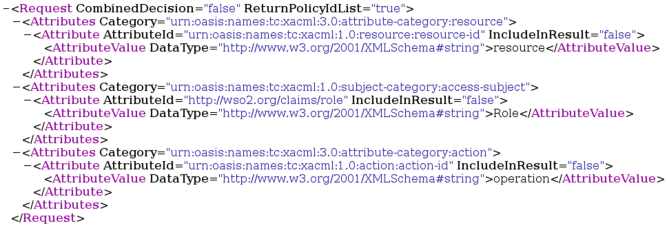


Fig. 4. XACML architecture



```
-<Request CombinedDecision="false" ReturnPolicyIdList="true">
 -<Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
  -<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" IncludeInResult="false">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">resource</AttributeValue>
   </Attribute>
  </Attributes>
 -<Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
  -<Attribute AttributeId="http://wso2.org/claims/role" IncludeInResult="false">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Role</AttributeValue>
   </Attribute>
  </Attributes>
 -<Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action">
  -<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" IncludeInResult="false">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">operation</AttributeValue>
   </Attribute>
  </Attributes>
 </Request>
```

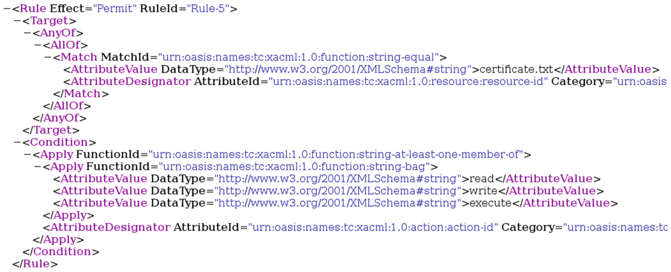
Fig. 5. User request sample

Policy Decision Point. The Policy Decision Point (PDP) receives Tom's request coming from the PEP. It needs to analyse if Tom fulfills the required conditions to read the file certificate.txt. The PDP will consult the Policy file to determine what actions Tom is allowed to perform on the file certificate.txt. Balana [24] provides us with an API call that allows us to create a PDP.

Policy Administration Point. To write policies, we have made use of the Simple Policy Editor. This policy editor is part of WSO2 Identity Server [25]. Simple Policy Editor allows anyone to create XACML 3.0 policies without an extensive knowledge of XACML language. However, an understanding of access control rules is required. Figure 6 is a sample of our policy file.

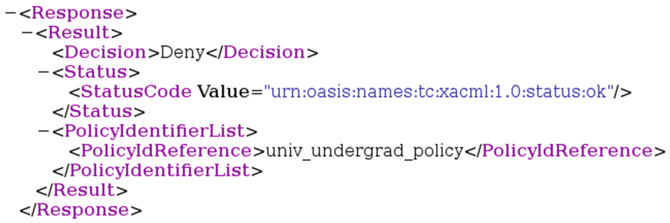
In addition to the Policy file, the PDP also consults the user-role assignment table. After determining Tom's role, which is undergraduate, and consulting the Policy file, the PDP reaches the conclusion to authorize Tom to read the desired file. The PDP passes that decision back to the PEP. That response is represented as a XACML file. A sample of the response XACML file is exhibited on Fig. 7.

The PEP then replies to Tom with a response granting him access to the file. Tom can now access the file server.



```
−<Rule Effect="Permit" RuleId="Rule-5">
  −<Target>
    −<AnyOf>
      −<AllOf>
        −<Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">certificate.txt</AttributeValue>
            <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" Category="urn:oasis
          </Match>
        </AllOf>
      </AnyOf>
    </Target>
  −<Condition>
    −<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-member-of">
      −<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">write</AttributeValue>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">execute</AttributeValue>
        </Apply>
        <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" Category="urn:oasis:names:tc
      </Apply>
    </Condition>
  </Rule>
```

Fig. 6. Policy file sample



```
−<Response>
  −<Result>
      <Decision>Deny</Decision>
    −<Status>
        <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
      </Status>
    −<PolicyIdentifierList>
        <PolicyIdReference>univ_undergrad_policy</PolicyIdReference>
      </PolicyIdentifierList>
    </Result>
  </Response>
```

Fig. 7. Response sample

Leader Election. Our objective is to protect the Access Control Module by regularly switching the Authorization Control Service providing the Access Control service at any given time. For the sake of demonstrating, we have chosen to switch the Authorization Control Service after every 10 min plus a random number of seconds. The random time is added to cancel the predictability of the time when the election takes place. An attacker knowing when the new leader is elected can schedule his attack accordingly.

An election is called after the end of term of the current Leader. In this instance, that term is set to ten minutes and some random seconds. The current leader being ACS1, it is the one responsible to call for an election. Using Java, the Leader Election is implemented according to the adaptation of the protocol presented in Sect. 4.2. At the end of the protocol a new leader is elected. This leader is different from ACS1, for instance ACS3 has been selected a the new leader. This is the server that is going to be responsible of providing the Authorization Control Service until next election. We made all Authorization Control Services probe the leader after every ninety seconds by sending a IsAlive message to it. This is done in order to detect the failure of the current leader.

Tom want to request another file stored on the file server, but the Authorization Control Service has been moved from ACS1 to ACS3. Any attacker who was in the middle of preparing an attack against ACS1 will be attacking the wrong Authorization Control Service, which is the intended goal of our architecture. However, Tom will also be sending his authorization request to the wrong server.

Migration. We have implemented the Migration Module as a mechanism to simply transfer the session history file and the policy file from the previous leader ACS1 to the elected leader ACS3. As stated in Sect. 4.2, the other access control modules are the same across all Authorization Control Services. The reason for transferring ACS1 Policy file to ACS3 is that while ACS1 was providing the Authorization service, policies, resources and users may have been updated. To avoid disruption in the access control service, ACS3 needs to have the most recent policy file.

Other alternatives to this migration can be envisioned. One option is to store the policies in a Policy Database, and replicate the database across all the Authorization Control Services accordingly. Another option would be to migrate the database from the current leader to the new leader at the end of an election. An additional option would be to use a single Policy database that would be shared by all the Authorization Control Services. This last option can create a potential issue by making that single policy database a single point of failure attractive for would be attacker.

5.3 Discovery Service

We need to let Tom know that the Authorization Control Server ACS3 has been elected as the new leader, and therefore he should send his request to ACS3. The Discovery Service allows us to achieve that goal through the adaptation of SLP protocol.

We have implemented the Discovery Service using a tool called OpenSLP [35] which is an open source implementation of the Service Location Protocol. OpenSLP can be used either in a three components mode or in a two components mode. In the first mode, we can have a User Agent (UA), a Service Agent (SA) and Directory Agent (DA). The User Agent is the Agent requesting services. The Service Agent is the Agent providing the services, while the Directory Agent is the repository of services. In a two component mode, we can have only the User Agent and the Service Agent. In this case, the Service Agent plays also the role of a Directory Agent (DA). For the sake of this demonstration, we have implemented the later option. We have installed OpenSLP and made sure that *slpd*, which is the OpenSLP daemon, is running.

Service Agent. Since our setting do not use a Directory Agent, the new leader will have to register its services with *slpd* upon being elected. The old Authorization Control Service, previously registered, is unregistered to avoid confusing users. The new leader registers its access control service by issuing a query in the form of a ServiceURL. The ServiceURL has the following form: *service:ServiceName://IPAddress:PortNo*, where ServiceName is the Authorization Service, IPAddress is the IP address of the new leader, and PortNo is the port where the Authorization Service is running.

User Agent. In our setting, the User Agent is Tom who needs to find the location of the new leader which is providing the access control service. In order to discover the Authorization Service, Tom's client sends a multicast packet with a ServiceURL. The Service Agent, which is the new leader, will verifies if Tom's query matches the registered service. In case of a match, the Service Agent replies to Tom with the ServiceURL informing him how to access the access control service.

6 Conclusion and Future Work

We have sketched a Moving Target Defense architecture aiming at defending an Access Control Reference Monitor. The design allows a master Resource Access Server and a master Authorization Control Server to be periodically and randomly switched to other ones. This mechanism allows the disruption of any ongoing attack on the Access Control Reference Monitor. This work opens a new direction in research on Moving Target Defense of an Access Control Reference Monitor. This architecture can benefit from some improvements. For instance, we do not believe that the election algorithm is an optimal one in term of computation and the number of messages exchanged during the election process.

Acknowledgement. This work was partially supported by funding from CableLabs, the US National Science Foundation under grant number 1650573, and the US Department of Energy under contract DE-NE0008571. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of CableLabs, the National Science Foundation, or the Department of Energy.

References

1. Abu-Amara, H., Lokre, J.: Election in asynchronous complete networks with intermittent link failures. IEEE Trans. Comput. **43**(7), 778–788 (1994)
2. Al-Shaer, E.: Toward network configuration randomization for moving target defense. In: Jajodia, S., Ghosh, A.K., Swarup, V., Wang, C., Wang, X.S. (eds.) Moving Target Defense, vol. 54, pp. 153–159. Springer, New York (2011)
3. Anderson, J.: Computer Security Technology Planning Study. Technical report ESD-TR-73-51, Electronic Systems Division, Hanscom Air Force Base, Hanscom, MA (1974)
4. Antonatos, S., Akritidis, P., Markatos, E.P., Anagnostakis, K.G.: Defending against hitlist worms using network address space randomization. Comput. Netw. **51**(12), 3471–3490 (2007)
5. Arghavani, A., Ahmadi, E., Haghighat, A.: Improved bully election algorithm in distributed systems. In: 2011 International Conference on Information Technology and Multimedia (ICIM), pp. 1–6. IEEE (2011)
6. Brunekreef, J., Katoen, J.P., Koymans, R., Mauw, S.: Design and analysis of dynamic leader election protocols in broadcast networks. Distrib. Comput. **9**(4), 157 (1996)
7. Burrows, M.: The chubby lock service for loosely-coupled distributed systems. In: Proceedings of the 7th Symposium on Operating systems design and implementation, pp. 335–350. USENIX Association (2006)

8. Castro, M., Liskov, B.: Practical byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. (TOCS) **20**(4), 398–461 (2002)
9. Castro, M., Liskov, B., et al.: Practical byzantine fault tolerance. In: OSDI 1999, pp. 173–186 (1999)
10. Compton, M.D.: Improving the quality of service and security of military networks with a network tasking order process (2010)
11. Copeland, C., Zhong, H.: Tangaroa: a byzantine fault tolerant raft
12. Evans, D., Nguyen-Tuong, A., Knight, J.: Effectiveness of moving target defenses. In: Jajodia, S., Ghosh, A.K., Swarup, V., Wang, C., Wang, X.S. (eds.) Moving Target Defense, vol. 54, pp. 29–48. Springer, New York (2011)
13. Ferraiolo, D., Cugini, J., Kuhn, D.R.: Role-based access control (RBAC): features and motivations. In: Proceedings of 11th Annual Computer Security Application Conference, pp. 241–48 (1995)
14. Ferreira, A., Chadwick, D., Farinha, P., Correia, R., Zao, G., Chilro, R., Antunes, L.: How to securely break into RBAC: the BTG-RBAC model. In: Annual Computer Security Applications Conference, ACSAC 2009, pp. 23–31. IEEE (2009)
15. Fraim, L.J.: Scomp: a solution to the multilevel security problem. IEEE Comput. **16**(7), 26–34 (1983)
16. Garcia-Molina, H.: Elections in a distributed computing system. IEEE Trans. Comput. **31**(1), 48–59 (1982)
17. Gilbert, M.D.M.: An examination of federal and commercial access control policy needs. In: National Computer Security Conference, 1993 (16th) Proceedings: Information Systems Security: User Choices, p. 107. DIANE Publishing (1995)
18. Guttman, E.: Service location protocol: Automatic discovery of IP network services. IEEE Internet Comput. **3**(4), 71–80 (1999)
19. Han, Y., Lu, W., Xu, S.: Characterizing the power of moving target defense via cyber epidemic dynamics. In: Proceedings of the 2014 Symposium and Bootcamp on the Science of Security, p. 10. ACM (2014)
20. Howard, H., Schwarzkopf, M., Madhavapeddy, A., Crowcroft, J.: Raft refloated: do we have consensus? ACM SIGOPS Oper. Syst. Rev. **49**(1), 12–21 (2015)
21. Huh, J.H., Bobba, R.B., Markham, T., Nicol, D.M., Hull, J., Chernoguzov, A., Khurana, H., Staggs, K., Huang, J.: Next-generation access control for distributed control systems. IEEE Internet Comput. **20**(5), 28–37 (2016)
22. Hunt, P., Konar, M., Junqueira, F.P., Reed, B.: Zookeeper: wait-free coordination for internet-scale systems. In: USENIX Annual Technical Conference, vol. 8, p. 9 (2010)
23. Inc, A.: Bonjour. https://support.apple.com/bonjour. Accessed: 26 Feb 2017
24. Info, X.: Balana. http://xacmlinfo.org/2012/12/18/getting-start-with-balana. Accessed: 26 Feb 2017
25. Info, X.: Wso2 identity server. http://xacmlinfo.org/category/wso2is/. Accessed: 26 Feb 2017
26. King, V., Saia, J., Sanwalani, V., Vee, E.: Scalable leader election. In: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm, Miami, FL, USA (2006)
27. King, V., Saia, J., Sanwalani, V., Vee, E.: Scalable leader election. In: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm, pp. 990–999. Society for Industrial and Applied Mathematics (2006)
28. Lamport, L.: The part-time parliament. ACM Trans. Comput. Syst. (TOCS) **16**(2), 133–169 (1998)
29. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. (TOPLAS) **4**(3), 382–401 (1982)

30. Lamport, L., et al.: Paxos made simple. ACM Sigact News **32**(4), 18–25 (2001)
31. Le Lann, G.: Distributed systems-towards a formal approach. In: IFIP Congress, Toronto, vol. 7, pp. 155–160 (1977)
32. Lee, J.W., Schulzrinne, H., Kellerer, W., Despotovic, Z.: z2z: discovering zeroconf services beyond local link. In: 2007 IEEE Globecom Workshops, pp. 1–7. IEEE (2007)
33. Martin, J.P., Alvisi, L.: Fast byzantine consensus. IEEE Trans. Dependable Secure Comput. **3**(3), 202–215 (2006)
34. Mohammed, I., Dilts, D.M.: Design for dynamic user-role-based security. Comput. Secur. **13**(8), 661–671 (1994)
35. OpenSLP: Service location protocol. http://www.openslp.org/. Accessed: 26 Feb 2017
36. ORACLE: Trusted Solaris Operating System. http://www.oracle.com/technetwork/server-storage/solaris/overview/index-136311.html
37. Perkins, C., Kaplan, S.: Service location protocol. In: ACTS Mobile Networking Summit/MMITS Software Radio Workshop (1999)
38. Perkins, C.E., et al.: Dhcp options for service location protocol (1999)
39. Presser, A., Farrell, L., Kemp, D., Lupton, W.: UPnP device architecture 1.1. In: UPnP Forum, vol. 22 (2008)
40. Rissanen, E., et al.: Extensible access control markup language (xacml) version 3.0 (2013)
41. Ruj, S., Stojmenovic, M., Nayak, A.: Decentralized access control with anonymous authentication of data stored in clouds. IEEE Trans. Parallel Distrib. Syst. **25**(2), 384–394 (2014)
42. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. Computer **29**(2), 38–47 (1996)
43. Sandhu, R.S., Samarati, P.: Access control: principle and practice. IEEE Commun. Mag. **32**(9), 40–48 (1994)
44. Sayeed, H.M., Abu-Amara, M., Abu-Amara, H.: Optimal asynchronous agreement and leader election algorithm for complete networks with byzantine faulty links. Distrib. Comput. **9**(3), 147–156 (1995)
45. Schell, R., Tao, T., Heckman, M.: Designing the GEMSOS security kernel for security and performance. In: Proceedings of the 8th National Computer Security Conference, Gaithersburg, MD (1985)
46. Shi, X., Lin, H., Jin, H., Zhou, B.B., Yin, Z., Di, S., Wu, S.: Giraffe: a scalable distributed coordination service for large-scale systems. In: 2014 IEEE International Conference on Cluster Computing (CLUSTER), pp. 38–47. IEEE (2014)
47. Singh, G.: Leader election in the presence of link failures. IEEE Trans. Parallel Distrib. Syst. **7**(3), 231–236 (1996)
48. Soundarabai, P.B., Thriveni, J., Manjunatha, H., Venugopal, K., Patnaik, L.: Message efficient ring leader election in distributed systems. In: Chaki, N., Meghanathan, N., Nagamalai, D. (eds.) Computer Networks & Communications (NetCom), pp. 835–843. Springer, New York (2013)
49. The TrustedBSD Project: Trustedbsd. http://www.trustedbsd.org
50. Veizades, J., Perkins, C.E.: Service location protocol (1997)
51. Wang, G., Koshy, J., Subramanian, S., Paramasivam, K., Zadeh, M., Narkhede, N., Rao, J., Kreps, J., Stein, J.: Building a replicated logging system with apache kafka. Proc. VLDB Endowment **8**(12), 1654–1655 (2015)
52. Wright, C., Cowan, C., Smalley, S., Morris, J., Kroah-Hartman, G.: Linux security modules: general security support for the linux kernel. In: Proceedings of the 11th USENIX Security Symposium, San Francisco, CA (2002)

53. Xu, J., Guo, P., Zhao, M., Erbacher, R.F., Zhu, M., Liu, P.: Comparing different moving target defense techniques. In: Proceedings of the First ACM Workshop on Moving Target Defense, pp. 97–107. ACM (2014)
54. Zhuang, R., DeLoach, S.A., Ou, X.: Towards a theory of moving target defense. In: Proceedings of the First ACM Workshop on Moving Target Defense, pp. 31–40. ACM (2014)