

Usage Control Enforcement in Linux NFS Server

H. Aboelsoud M

Department of Computer Engineering
Military Technical College
Cairo, Egypt

Abstract—Servers in distributed environments are the main targets of malicious attacks. As a consequence, the need for securing and protecting resources residing on such servers is considered a major and continuous challenge. However, traditional access control models are not suitable for regulating access in today's highly dynamic, distributed, network-connected, and open computing environments as they are usually not expressive enough to capture important security requirements such as continuity of decisions (ongoing controls) and mutability of attributes, besides lacking of important decision factors like obligations and conditions. Hence, the usage control (UCON) comes as a novel and promising approach to overcome the inadequacies of traditional access control models [1]. However, applying UCON in modern distributed environments is usually introducing complex usage scenarios and new challenging issues as discussed by Grompanopoulos et al. [2]. This paper, taking into account Grompanopoulos's UCON challenges, studies usage control enforcement in distributed file systems, and take Linux Network File System (NFS) as a case-study. This work follows the approach proposed in [3] to present UCON based on the schema of the OM-AM [4] (Objectives, Models, Architectures, Mechanisms) engineering design philosophy by focusing on the architectures and mechanisms layers. An enforcement architecture design following the Sandhu's UCON_{ABC} model [5] is proposed and a prototype implementation in the Linux NFS server, on top of the existing DAC mechanism, is also proposed as a proof of concept. The implementation includes modifications to the Linux NFS server through the `nfsd` loadable kernel module (LKM), which handles the main functionality of the Linux NFS server. Security and performance analysis were conducted to ensure that our system enforced the UCON policies as expected and to measure the additional overhead for making UCON checks compared with an unmodified kernel (vanilla kernel).

Keywords—Access Control, Operating System (OS) Security, Usage Control, and NFS Security.

I. INTRODUCTION

Distributed file system objects like files and directories are examples for valuable and sensitive system resources that need to be protected by OS which uses access control mechanisms to protect and control access to them. Traditionally, access control has dealt only with authorization decisions on users' access to target resources. The most widely used traditional access control models are [6]: Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role Based Access Control (RBAC).

In DAC, objects or data are owned by a user (owner) and permission to act on them is given at the discretion of the owner [7]. DAC is widely implemented in many systems (e.g. UNIX, Windows, etc.) because of its flexibility and ease of

implementation. In MAC, access is based on labels assigned to subjects and objects and access decisions are made beyond the control of the individual owner of the object [8]. MAC is implemented in UNIX-based systems through Domain and Type Enforcement (DTE) and implemented in Windows through a security feature called Mandatory Integrity Control (MIC). In RBAC, access is granted based on the roles individual users have in their organization based on their job functions [9], RBAC is implemented in windows OS using groups.

Other modern access control models include Trust Management (TM) and Digital rights management (DRM). TM [10] is a model to authorize unknown entities in an open environment, but it deals only with static entities, whose characteristics do not change in time. DRM [11] concerns on controlling and tracking access to and usage of already disseminated digital objects at client-side, it mainly focus on intellectual property rights protection.

However, current classic access control models are not suitable for regulating access as they are usually dedicated to specific target problems (ad-hoc solutions) and not comprehensive enough to cover the broad traditional models. Hence, UCON comes as a unified framework to extend traditional access control models in a way that make it suitable for new challenges in the computer security [1]. UCON encompasses traditional access control models, TM, DRM and other enhanced access control models, by integrating authorizations, obligations, and conditions and providing the properties of decision continuity and attribute mutability [5].

This research studies a UCON model called UCON_{ABC} model, describes enforcement architecture following this model and implements it in the Linux NFS server, on top of the existing DAC mechanism, as a proof of concept. This work discussed here is not meant to replace the underlying OS controls, but to offer an additional decision level to them, more accurate, flexible and consistent. This work follows the OM-AM engineering design philosophy which allows describing UCON in four relatively independent layers beginning from the high-level specification till low-level enforcement mechanisms and implementation. The main focus of this work is on the architectures and mechanisms layers.

An important issue in designing secure systems is to decide at which level security mechanisms should be placed. Typically, the organization of distributed systems consists of separate layers for applications, middleware, OS services and the OS kernel. In this work the decision was made to implement the proposed work in the OS kernel layer because of the following reasons:

- i. Typically, an access/usage control decision is made and enforced by a reference monitor [12], principles that should be considered when it is implemented are [13] being tamper-proof and always-invoked which is reasonably easy to achieve by implementing it in the kernel layer.
- ii. Doing all the work inside the kernel has a good impact on the performance as transitions between user-mode and kernel-mode cost time and resources.

This paper is organized as follows. Section 2 describes the background, including the NFS architecture and UCON_{ABC} model. Section 3 introduces the related works. Section 4 describes the proposed work, including the proposed architecture, implemented prototype details, some policies expressing covered UCON_{ABC} core models, and considerations made in this work for Grompanopoulos's UCON challenging issues. Section 5 provides security and performance analysis. Section 6 gives some conclusions and presents future work directions.

II. BACKGROUND

In this section, first, the NFS architecture is reviewed showing the existing access control mechanism then the UCON_{ABC} model is reviewed.

A. NFS Architecture

NFS is one of the most widely-used server-based distributed file systems. The model underlying NFS is a client/server model which implements a communication protocol that provides the clients transparent access to a file system that is managed by a remote server [14]. In UNIX based systems NFS is generally implemented following the layered architecture shown in figure 1. As can be seen clients are offered a common interface for different file systems called the Virtual File System (VFS). The main idea of the VFS is to hide the differences between these file systems by abstracting common tasks of them so now the users can interact with the VFS no matter what type of file system they are accessing.

At client side, when a process make a NFS request the VFS interface passes it to a separate component known as the NFS client, which takes care of handling access to files stored at the remote NFS server, Then NFS client implements the NFS file system operations as RPCs to the server [14]. On server side, after The NFS server receives the incoming client request The RPC stub unmarshals the request and the NFS server converts it to regular VFS file operations that are subsequently passed to the VFS layer which in turn translating them to the appropriate operations within the local file system in which the actual files are stored [14].

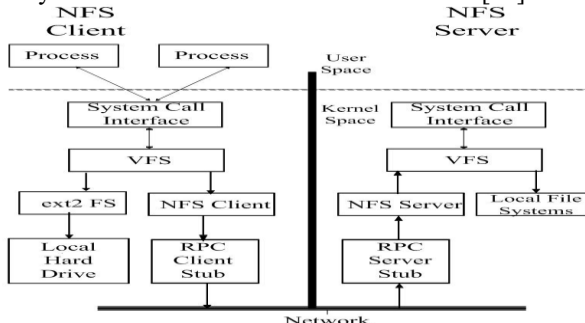


Fig.1. The NFS Architecture for UNIX-Based Systems [14].

In NFS systems, after a client has been authenticated, it is necessary to check whether that client has sufficient access rights that make such a request carried out by the server. This can be done by checking each access request against group of predefined policies to decide whether the request granted or denied. Current Linux NFS servers control the access to the shared resources residing on it by using a simple form of ACL maintained by the security administrator in which the access rights (read, write) of the exported NFS volumes are defined [15]. An entry in ACL typically looks like this: Directory machine1 (option11, option12) machine2 (option21, option22)

Where Directory is the directory you want to export, machine1 and machine2 are the client machines that are allowed to access the directory and optionxx is the option listing for each machine describing what kind of access that machine have.

In several cases this solution is not enough and needs to be enhanced, there are various access/usage control scenarios that cannot be achieved using this classical method. For example, suppose that only N clients can access an object simultaneously, some resources are accessible only during business hour or monitoring whether a certain processes are running or not at NFS client side. Such scenarios are not possible with the current access control mechanism. Hence, applying UCON_{ABC} in NFS environment is necessary to cover the requirements that such scenarios and others may request.

B. The UCON_{ABC} Model

UCON_{ABC} model proposed by Sandhu et al. [5] formalizes the UCON notion based on the concepts of authorization (A), obligations (B), and conditions (C) and also introduces new features like continuity (ongoing controls) and mutability of attributes, it encompasses and enhances traditional access control models (e.g., DAC, MAC, RBAC, etc.), TM, and DRM and goes beyond them in its definition and scope.

The UCON_{ABC} model consists of eight components [1] as shown in figure 2.

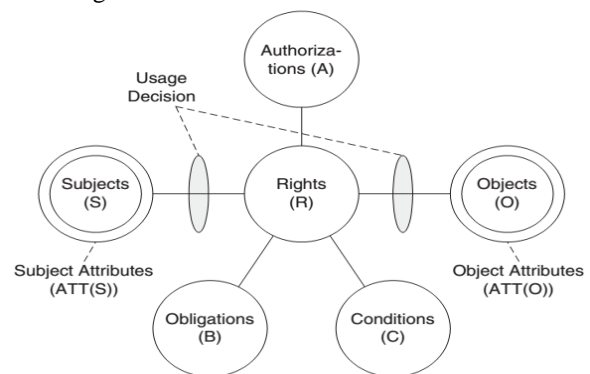


Fig.2. UCON_{ABC} Model Components [1].

Next, we briefly discuss the core components of UCON_{ABC} model giving examples on them in the context of NFS environment.

Subjects and Objects: Subjects are entities that request the usage of other entities (objects). When applying UCON in NFS environment, the subjects are the NFS client machines and objects are the shared files and directories residing on NFS server.

Rights: Rights enable access of a subject to an object in a particular mode [1]. In NFS, examples of rights are read and write permissions. Because files and directories are different entities, the meaning of these permissions assigned to each differs slightly. In case of directory, the read permission allows the user to list the files in the directory and the write permission allows the user to add, rename and remove a directory entry [14].

Attributes: Both subjects and objects have attributes which are properties that can be used during the access decision process. In NFS, examples of subject attributes include subject's identity, roles and security clearance whereas examples of object attributes include security labels, object's type and ACLs.

Decision factors: Authorizations, obligations and conditions are UCON_{ABC} model decision factors that are used to determine whether a subject should be allowed to access an object in a particular mode:

- Authorizations: are evaluated based on subject and object attributes and the requested right to determine whether to grant the requested right or not. The evaluation of the authorization decision can be performed prior to the usage of an object (pre-authorization) or during the usage (ongoing authorization). An example for an authorization policy in NFS systems is the permission of a NFS client to read a shared file/directory residing on the NFS server.
- Obligations: are mandatory actions that a subject must perform before, during or after the usage of an object. An example for an ongoing obligation in NFS systems is that a NFS client has to keep certain process running on his machine while he is logged into NFS service.
- Conditions are subject and object independent environmental or system-oriented restrictions that have to be satisfied before or during the usage process. Examples for conditions in NFS systems are accessible time period for a NFS client and processor load on NFS server.

The main novelties of UCONABC model are continuity of usage decisions and mutability of subject and object attributes. UCONABC recognized continuity of decision where usage decision is not only checked before an access, but also throughout the period of the usage process and the usage can be terminated if some specified policies are not satisfied [1].

The subject and object attributes can be classified into immutable and mutable attributes. Immutable attributes are modifiable only by administrative actions but are "immutable" in that the system does not modify them automatically [1]. Unlike immutable attributes, mutable attributes have to be updated as side-effects of a subject's usage on objects and do not require any administrative action for updates. These updates, in turn, may affect current or future usage decisions. For example, a subject's e-cash balance has to be decreased by the value of a digital object as the subject uses or accesses the object [1].

III. RELATED WORK

In [5] the UCON_{ABC} model proposed by (Park and Sandhu, 2004) formalizes UCON notion based on the concepts of authorization (A), obligations (B) and conditions (C) with unique properties of access decision continuity and attribute

mutability. Several approaches based on the UCON_{ABC} model for the OS protection are proposed in [16,17,18].

In [16], a simple but effective usage control model UCON_{KI} (the UCON kernel integrity) is proposed for OS kernel integrity protection, it concerns accesses to sensitive kernel objects (e.g., kernel text, system calls table, interrupt descriptor table) in a real-time manner.

In [17], the general requirements of trusted usage control enforcement in heterogeneous computing environments are identified and general platform architecture is proposed to meet these requirements. The overall goal of their approach is to build a "virtually closed" and trusted subsystem for remote usage control policy enforcement.

In [18], Teigao et al. define a usage control model based on UCON_{ABC} and describes a framework to implement it in OpenBSD 4.1 UNIX kernel to control the usage of local files. The prototype evaluation shows that the proposed model is feasible, straightforward, and may serve as a basis for more complex usage control frameworks.

In [2], a number of challenging issues faced when UCON is applied in modern computing environments were discussed in the context of suitable representative usage scenarios. The results of this study revealed various limitations in contextual information handling, lack to support complicated usage modes of subjects on objects, the lack of a feasible obligation fulfillment mechanism, and weaknesses in utilizing information concerning previous or current usages of system resources.

As the role that security plays increases more and more in distributed systems our work comes to increase the level of security applied in Linux NFS server, that form the basis for many distributed systems and applications, by providing the advantages offered by the UCON_{ABC} model and taking into account Grompanopoulos's UCON challenges. So, our work may serve as a solid base for more advanced research and developments in the security era of the modern distributed systems where sharing data is an essential process to these distributed systems.

IV. THE PROPOSED WORK

As the main focus of this work is on the architectures and mechanisms layers of the OM-AM methodology, this section describes the proposed architecture design and the details of the prototype implementation. Then, some policies expressing covered UCON_{ABC} core models are presented with its pseudo code. Finally, considerations made in this work for Grompanopoulos's UCON challenging issues are discussed.

A. The proposed Architecture

Figure 3 shows an overview for the architecture. As can be seen, the proposed architecture is not intended to substitute the current NFS access control system, but to extend it by offering an extra decision level to it. Our system components are represented by the colored boxes to distinguish them from the original components of the NFS system

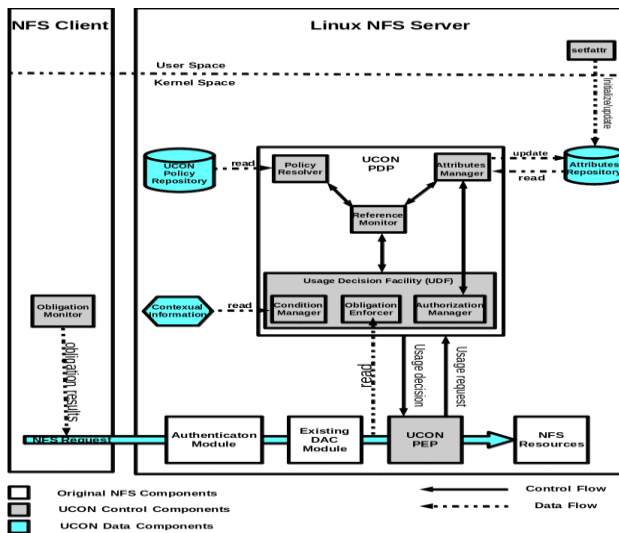


Fig.3. The Proposed Architecture.

The proposed architecture includes three main components which are a Policy Enforcement Point (PEP), a Policy Decision Point (PDP) and data sources components. The PEP component task is to intercept every NFS client request after it is accepted by the original NFS access control mechanism and then forwards this request to the PDP component which then in turn decides whether to accept or deny the client request by matching it against a set of UCON policies retrieved from the UCON policies repository component, then PDP responds with the decision to the PEP which enforces it.

The PDP component is composed of a number of internal sub-components, which are:

- **Reference Monitor (RM)** is the core component which represents a gateway for all the usage decisions.
- **Policy Resolver (PR)** is a parser responsible for retrieving the policies from the UCON policy repository and resolving them into an internal representation to be used by the PDP.
- **Attribute Manager (AttM)** is responsible for collecting subject (NFS client) and object (requested file) attributes from the attributes repository to use them in the process of evaluating the policy and it is also responsible for updating these attributes before, during and after the usage process.
- **Usage Decision Facility (UDF)** includes the following functional modules:
 - **Authorization Manager (AuthM)** As the authorization decisions is based on subject and object attributes, this component communicates with the attribute manager to get the attributes required in the process of evaluating the usage policy.
 - **Obligation Manager (OBMAN)** As operating in a distributed file system environment requires that monitoring the fulfillment of obligations is done at client side and verifying them is done at server side, in the proposed architecture the obligation manger is divided into two sub-components:
 - **Obligation Monitor (OBMON)** exists at the NFS client side, its role is monitoring whether the client has satisfied each required obligation

or not. It can monitor whether certain processes (e.g., antivirus, antispysware, etc.) are running or not to satisfy that the client machine meets specific security requirements. Then, moving obligations data to the NFS server by setting an array of Boolean values (true or false) and inject this array into the NFS client request.

- **Obligation Enforcer (OBE)** exists at the NFS client side, its role is extracting the obligations Boolean array from the NFS client request fed to the PDP and verifying the fulfillment of each obligation by checking its corresponding Boolean flag.
- **Condition Manager (CM)** which is responsible for gathering external condition information, to be used in the policy evaluating process, like current time and resources usage (e.g., processor, memory, etc.). The PDP invokes the condition manager whenever needed if the security policy requires the evaluation of a certain condition.

There are also some external data sources components which provide our PDP component with the needed information:

- **UCon Policy Repository (UPR)** stores the UCON policies and provides them to the PDP to be evaluated.
- **Attributes Repository (AR)** stores the subjects and objects attributes and provides them to the attributes manager to be used in the process of usage decision evaluation.

When NFS server receives a request from a NFS client, it authenticates the request and then the current NFS access control system authorizes the request by inspecting the ACL to decide whether the access should be allowed or not. If the request is accepted then our PEP component intercepts and forwards it to the PDP where most of the work is done.

At first, the RM extracts subject ID and object path from the NFS request and pass this information to PR to retrieve the corresponding UCON policies from the UPR. After the PR loads the policies from the repository it parses them and create the equivalent data structures that are used in the policy evaluation process. According to the policy rules created by the policy resolver the RM invokes the appropriate usage decision components from UDF to evaluate the policies. For example, to evaluate the authorization policy the authorization manager is invoked which in turn communicate with the attributes manager to fetch the appropriate subject and object attributes from the AR, these attributes are used in the decision evaluation process and they may be updated as a as a consequence of usage process and these updates may cause reevaluation of the policy by the PDP which may revokes an ongoing access. The RM may also invoke the condition and obligation managers whenever needed if the retrieved policies require the evaluation of a certain condition or obligation.

If any policy rule is violated during the decision evaluation process made by UDF the RM responds immediately with a deny decision to the PEP. Otherwise, it returns the PEP an allow decision. Based on the PDP's decision, the PEP then enforces the received result by either blocking the NFS request or making it continue its way to the requested object.

B. Implementation Details

The development environment used involved the following:

- Red Hat Enterprise Linux (RHEL) Server 7 (Kernel 2.6.36.1): used as the target OS for development and running the NFSv4.1 server.
- Virtual Machine Manager 0.8.4: used for running the virtual machines that represents the NFS Clients.
- Emacs 24.3: is an extensible customizable text editor used for editing the C source code.
- GNU Compiler Collection (GCC) compiler 4.7.2: open source command-line software designed to act as a compiler for Linux-based OSs.

The implementation details of the components presented in the proposed architecture:

PEP and PDP components

- The PEP and PDP components are implemented in the nfsd loadable kernel module (LKM) which handles the main functionality of the NFS server.
- The PEP insertion points are injected in some relevant NFS file operations functions (e.g., nfsd_open, nfsd_close, nfsd_read, and nfsd_write) after they are accepted by the original NFS DAC mechanism.
- Usage session for every file is initiated with the call to nfsd_open function and terminated with the call to nfsd_close function.

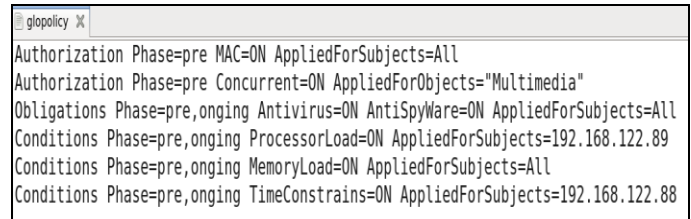
RM component

It calls the ExtractClientID function at the beginning of the code to extract the identifier of the NFS client who makes the request, gets as parameter pointer to svc_rqst structure (data of the NFS client request) and then calls the ExtractObjectPath function to extract the path of requested file, gets as parameter pointer to file structure (data of the requested file). Then, it sends these data as parameters to the UCON_PolicyResolver function to retrieve the corresponding policies from the UPR. Finally, based on the retrieved policies it calls the appropriate usage decision components to evaluate the policies.

UPR component

All policies are stored in plain text files, to simplify the job of PR component, in the /etc/uc4nfs/repository directory on the NFS server, this directory has two files which are:

- A file called (revoc_list) which is a plain text file represents the client's revocation list, it is the first file checked before and during the usage session. If any NFS client appears in this file its request is rejected immediately without any further checks.
- A global policy file named (glopolicy) which contains usage policies for NFS clients. The format of this file is shown in figure 4, as can be seen the simplicity and the clearness are considered in this format to make it suitable to be parsed inside the Linux OS kernel. The file consists of group of entries, each entry contains a policy which starts with the usage decision factor (Authorization | Obligation | Condition) followed by a series of (key, value) pairs used by the policy resolver for evaluating the usage policies.



```
glopolicy X
Authorization Phase=pre MAC=ON AppliedForSubjects=All
Authorization Phase=pre Concurrent=ON AppliedForObjects="Multimedia"
Obligations Phase=pre,onging Antivirus=ON AntiSpyWare=ON AppliedForSubjects=All
Conditions Phase=pre,onging ProcessorLoad=ON AppliedForSubjects=192.168.122.89
Conditions Phase=pre,onging MemoryLoad=ON AppliedForSubjects=All
Conditions Phase=pre,onging TimeConstrains=ON AppliedForSubjects=192.168.122.88
```

Fig.4. UCON Policies in glopolicy File.

PR component

At first, it reads the policies from UPR component. Then, a series of calls to some C library string functions (e.g., strsep(), strtok() functions) are used to split the policies into meaningful tokens, then a series of calls to data types converter functions (e.g., atoi() function) are used to convert the strings to the appropriate data types. The retrieved policies are filtered based on the NFS client identity and the requested object to get only the appropriate policies to be evaluated by UDF component.

AR component

The object mutable/immutable attributes (e.g., classification and type) are persistently stored in the Linux extended attributes which are properties organized in (name, value) pairs associated with file system objects. On Linux, specifically, there are four extended attribute classes: security, system, trusted and user. In our case we use the extended security attributes which are supported by the local file system ext4 used to export the NFS volumes to the clients. The subject mutable/immutable attributes (e.g., clearance and allowed time interval) are stored in the /etc/uc4nfs/clients/client-id file where each NFS client has its own attributes file named with the client identifier (IP or machine name).

AttM component

It is responsible for updating and retrieving the subject (NFS client) attributes by reading and parsing the client specific attributes file, besides retrieving the object (requested file) attributes through system call getxattr() which retrieves the value of the object extended attribute identified by the attribute name and associated with the given object path in the file system to use them in the process of evaluating the policy and it is also responsible for updating these attributes before, during and after the usage process through system call setxattr().

AuthM component

Pre-authorization control is implemented by making modifications to nfsd_open function while ongoing-authorization controls are implemented by making modifications to nfsd_read and nfsd_write functions. In the process of evaluating the policy communications with the attribute manager are made for requesting and updating the attribute values of the NFS clients and objects.

OBE component

In the nfsd LKM, svc_rqst structure contains the data of the NFS request like the client who makes the request. A modification made to this structure by adding a 4-byte pointer to Boolean array to be checked by the obligation enforcer in the process of verifying the fulfillment of obligations.

OBMON component

The implementation of this component is not covered in this work.

CM component

To get the current processor and memory usage, queries are made to the /proc/stat file which contains some stats about the kernel and exists in the procfs file system which is a special file system in Linux OS that represents system information in a hierarchical file-like structure [19]. To get the current time, at first a call to the do_gettimeofday() function is made which returns the time of day expressed as seconds and microseconds and stores it in a timeval structure, then a call to time_to_tm() function is made to convert it into a human readable time format like Hour:Min:Sec:Msec format and store it in a tm structure.

C. Proposed UCON Policies

UCON_{ABC} is actually a family of core models with several parameters. Based on the three decision factors authorizations (A), obligations (B), and conditions (C), along with mutability of attributes (immutable(0), pre-update(1), on-update (2), post-update (3)) and continuity of enforcement (pre or on-going) the model space is enumerated as shown in table 1 [5] to define the 16 basic UCON_{ABC} core models.

TABLE 1: THE 16 BASIC UCON_{ABC} MODELS WITH OUR COVERED MODELS.

	0 (immutable)	1 (pre-update)	2 (ongoing-update)	3 (post-update)
preA	Y	Y	N	Y
onA	Y	Y	Y	Y
preB	Y	Y	N	Y
onB	Y	Y	Y	Y
preC	Y	N	N	N
onC	Y	N	N	N

Cases that are not likely to be suitable in practice are marked as “N”. If decision factor is “pre”, updates are expected to occur only before or after the right is exercised and there is little reason to have ongoing updates [5]. For condition models, evaluation of condition cannot update attributes as it just checks current environmental or system status [1]. In reality, many real-world systems will use some combination of these models. In our case the gray cells represents the core models that we covered in this work. In the following, some policies expressing covered UCON_{ABC} core models, formally defined in the original UCON_{ABC} model proposal [5], are presented with its pseudo code to show the effectiveness of applying UCON_{ABC} model in NFS environment:

Policy 1. MAC policy, UCON_{preA0}:

MAC policy is an example for pre-authorization control that is performed in open NFS operation by calling UCON_MAC function that gets the clearance and classification security labels by calling UCON_GetSubjectAttribute and UCON_GetObjectAttribute functions which are part of the attribute manager module. Then, based on the open mode (read or write) Bell-LaPadula’s [20] security properties (simple and star property) are utilized for allowing or denying the request.

Pseudo code:

```

UCON_MAC (NFSReq.File): check if a subject s has the right to read/write to an object o.
Input: A NFS client request structure.
Input: A File structure.
Output: Returns true if s has the right to read/write o, otherwise return false.
BEGIN Function
NFSClientID = ExtractClientID (NFSReq)
ObjectPath = ExtractObjectPath (File)
ClientClearance = UCON_GetSubjectAttribute(NFSClientID,"Clearance")
ObjectClassification = UCON_GetObjectAttribute (ObjectPath"Classification")
IF (OpenMode equals read) AND (ObjectClassification dominates ClientClearance) THEN
    Return false
ELSE IF (OpenMode equals write) AND (ClientClearance dominates ObjectClassification) THEN
    Return false
ENDIF
Return true
End Function
    
```

Policy 2. A limited number of concurrent accesses, UCON_{preA13}:

This policy limits the concurrent access to a given object to N clients. If N+1 client requests access, the access will be denied until the number of concurrent clients falls under the Limit N. Though this policy can be applied on any type of files but it is very suited for the multimedia files. As, NFS is a server-based distributed file system, where model underlying it is a client/server model in which NFS server responses to requests come from clients, playing a multimedia file makes NFS server responses to a large number of read requests and this number is linearly increasing in case of concurrent access by N clients. Thus, the number of concurrent accesses should be limited to make a reasonable load on the NFS server. To implement this policy, the UCON_ConcurrencyCheck function, which is part of the authorization manager module, is called in the NFS open file operation and the UCON_SetObjectAttribute function is called in the NFS close file operation to decrement the number of concurrent clients after the usage session.

Pseudo code:

```

UCON_ConcurrencyCheck (File): limit concurrent access to a given multimedia object.
Input: A File structure.
Output: Returns true if concurrency test successes, otherwise return false.
BEGIN Function
ObjectPath = ExtractObjectPath (File)
FileType = UCON_GetObjectAttribute (ObjectPath,"FileType")
IF (FileType equals multimedia) THEN
    CurrentUsage = UCON_GetObjectAttribute (ObjectPath,"CurrentUsage")
    MaxConcurrentUsage = UCON_GetObjectAttribute (ObjectPath,"MaxConcurrentUsage")
    IF(CurrentUsage < MaxConcurrentUsage) THEN
        UCON_SetObjectAttribute (ObjectPath," CurrentUsage", CurrentUsage+1)
        Return true
    Else
        Return false
    END IF
    END IF
    Return true
END Function
    
```

Policy 3. Revocation by appearing in a revocation list, UCON_{preA0onA0}:

In our work for every usage controlled file the usage session starts when NFS file open operation is called and terminates when NFS file close operation is called. Consequently, pre-authorization control is performed in open operation while ongoing-authorization controls are performed in read and write NFS operations. Such controls can be realized by checking a revocation list before the access is

allowed and repeating this check during the usage session with every read or write request. If any NFS client appears in this revocation list before the access or during the usage session the usage right is denied or revoked immediately. The following function is called in open, read, and write NFS operations.

Pseudo code:

UCON_RevocationList (NFSReq): Check whether a subject s should be denied/revoked or not before and during the usage session by checking a revocation list file.
Input: A NFS Request.
Output: Returns true if subject s should be denied/revoked, otherwise return false.
BEGIN Function
NFSClientID = ExtractClientID (NFSReq)
FileHandle = OpenRevocationListFile(Path, ReadMode)
IF (FileHandle equals null) THEN
 PrintError (ErrorMessage)
 Return true
END IF
Array A = LoadListInArray(FileHandle)
CloseRevocationListFile(FileHandle)
Foreach (element e in A)
 IF (NFSClientID match e) Then
 Return true
END Foreach
Return false
End Function

Policy 4. Temporal usage control, UCON_{preC0onC0}:

This policy restricts usage based on time which is an environmental factor that is independent from individual subjects and objects. Pre-condition control is performed in open NFS operation while ongoing-condition controls are performed in read and write NFS operations. Such controls can be realized by calling UCON_TimeCheck function which is part of the condition manager module.

Pseudo code:

UCON_TimeCheck (NFSReq): Check if a NFS client has the right of access/usage to an object at a certain time.
Input: A NFS request.
Output: Returns true if NFS Client has the right, otherwise return false.
BEGIN Function
NFSClientID = ExtractClientID (NFSReq)
StartTime = UCON_GetSubjectAttribute (NFSClientID, "StartTime")
EndTime = UCON_GetSubjectAttribute (NFSClientID, "EndTime")
CurrentTime = UCON_GetCurrentTime()
UCON_ConvertFormat(CurrentTime)
IF ((StartTime ≤ CurrentTime) AND (CurrentTime < EndTime)) THEN
 Return true
Else
 Return false
END IF
End Function

Policy 5. Processor usage limitation, UCON_{preC0onC0}:

This policy restricts usage based on processor usage which is a system factor that is independent from individual subjects and objects. Pre-condition control is performed in open NFS operation while ongoing-condition controls are performed in read and write NFS operations. Such controls can be realized by calling UCON_ProcessorCheck function which is part of the condition manager module.

Pseudo code:

Check_Processor_Usage_Constrains(NFSReq): Check if a NFS Client has the right of access/usage to an object at a certain processor usage.
Input: A NFS request.
Output: Returns true if NFS Client has the right, otherwise return false.
BEGIN Function
NFSClientID = ExtractClientID (NFSReq)

ProcessorLimit = UCON_GetSubjectAttribute (NFSClientID, "ProcessorLimit")
FileHandle = UCON_OpenFile ("/proc/stat", ReadMode)
IF (FileHandle equals null) Then
 PrintError (ErrorMessage)
 Return false
END IF
Fields = UCON_ReadFields(FileHandle)
CurrentProcessor = UCON_GetCurrentProcessorUsage(Fields)
IF (CurrentProcessor < ProcessorLimit) THEN
 Return true
Else
 Return false
END IF
End Function

D. UCON challenging issues considerations

The results of a study made by Grompanopoulos et al., revealed various limitations and a number of challenging issues faced when UCON is applied in modern computing environments and it also, discussed some solution approaches to these challenging issues. In this section considerations made in this work for most of these challenges are discussed.

Challenge1. Contextual information handling

The condition evaluation in UCON is a complicated process, especially in systems with a large number of condition variables, which make its implementation result in a very complicated usage control system. Based on the notice in [5], stating that there is a vague line differentiating which information should be assigned to attributes and which to condition variables, a proposed solution [2] to the above issues could be achieved through the assignment of contextual information to subject and object attributes.

The above mentioned solution is considered in our work by assigning of contextual information like time and processor usage to the attributes of NFS clients as mentioned in the previous section.

Challenge2. Keeping information about system usages

By utilizing attribute mutability, UCON becomes capable to support consumable rights and history-based access control. So, the recording of the system objects usages, by using the information regarding previous or current usages of them, is required in UCON systems. A proposed solution [2] to this issue could be achieved through the association of use attributes with system's objects.

The above mentioned solution is considered in our work by setting an upper bound limit on the number of concurrent usages of a given multimedia file by N clients. This has been achieved by association of CurrentUsage attribute with the multimedia file to track the number of concurrent usages of it.

Challenge3. Managing obligation enforcement

There is lack of a feasible obligation fulfillment mechanism as mentioned in [21]. In this work an obligation fulfillment mechanism is proposed at the architecture layer and partially implemented in the proposed prototype. As NFS is a distributed file system, so there is a need to split the obligation manger component into two sub-components:

- i. Client side sub-component (obligation monitor) which is responsible for the process of monitoring the fulfillment of obligations at client side and sending the obligation results to the NFS server by injecting them into the NFS request data.
- ii. Server side sub-component (obligation enforcer) which is responsible for the process of verifying the

obligation results at server side and enforcing the obligation policies.

V. TESTING AND ANALYSIS

Test plans for validating our work for correct functionality, verifying the enhanced security, and measuring its performance overhead were developed.

A. Validation Test

Validation testing was performed by using Pynfs tool [22] as it knows how to parse and generate the protocol itself, so it can talk directly to the client or server to be tested. This makes it particularly well-suited for testing responses to unusual error conditions, protocol conformance, and correctness of new features.

Test Results

Our enhanced Linux NFS server has gone through 640 tests made by Pynfs to be validated and the results obtained show that there are no unusual errors or problems with our enhancements.

B. Verification Test

Verification testing was performed to ensure that our system enforced the UCON policies as expected. NFSTEST_POSIX [23] is an open source tool comes as part of the NFSstest package; some modifications are made to this tool to make it suitable for verifying NFS file system level access using positive and negative testing.

Testing Environment:

- NFS Server: a machine running RHEL Server 7 configured to export (/home/ucontest) directory with the following Condition variables:
 - CurrentTime = 15 (3 PM)
 - Processorload < 20%
- NFS clients: 2 machines running Ubuntu 15.04.
- Original DAC mechanism on the NFS server is configured to give 2 NFS clients read/write permissions on the exported directory.
- Subjects and objects are set as the following:
 - 5 files are created on the NFS Server mounting point (/home/ucontest), with the attributes shown in table 2.
 - 2 NFS clients with the attributes shown in table 3.

TABLE 2: THE OBJECTS ATTRIBUTES.

Object	MAC attribute (classification)
File1	Normal
File2	Normal
File3	Secret
File4	Secret
File5	Secret

TABLE 3: THE SUBJECTS ATTRIBUTES.

Subject	MAC attribute (clearance)	Conditional attribute	Temporal attributes
Client1	Top Secret	Maxprocessorload:30%	2:6 (PM)
Client2	Normal	Maxprocessorload:40%	4:6 (PM)

The following test scenarios are performed to verify our work:

Test1 Scenario

- i. Mounting the exported directory (/home/ucontest) on the 2 NFS clients.
- ii. Running the modified NFSTEST_POSIX tool on the 2 NFS clients.

Test1 Results

As shown in figures 5 and 6, the test results produced by NFSstest_posix tool are matched with the expected results shown in table 4 which prove how our system is able to effectively apply the appropriate UCON policies on the 2 NFS clients.

TABLE 4: THE EXPECTED RESULTS.

Subject	Read		Write	
	Pass	Fail	Pass	Fail
Client1	5	0	0	5
Client2	0	5	0	5

```
*** Verify POSIX API open() on NFSv4.1
PASS: access - file1 access allowed with mode read
PASS: access - file2 access allowed with mode read
PASS: access - file3 access allowed with mode read
PASS: access - file4 access allowed with mode read
PASS: access - file5 access allowed with mode read
FAIL: access - file1 access denied with mode write
FAIL: access - file2 access denied with mode write
FAIL: access - file3 access denied with mode write
FAIL: access - file4 access denied with mode write
FAIL: access - file5 access denied with mode write
TIME: 1.251562s

10 tests (5 passed, 5 failed)
```

Fig.5. NFSTEST_POSIX Tool Results on Client1 Machine

```
*** Verify POSIX API open() on NFSv4.1
FAIL: access - file1 access denied with mode read
FAIL: access - file2 access denied with mode read
FAIL: access - file3 access denied with mode read
FAIL: access - file4 access denied with mode read
FAIL: access - file5 access denied with mode read
FAIL: access - file1 access denied with mode write
FAIL: access - file2 access denied with mode write
FAIL: access - file3 access denied with mode write
FAIL: access - file4 access denied with mode write
FAIL: access - file5 access denied with mode write
TIME: 1.316778s

10 tests (0 passed, 10 failed)
```

Fig.6. NFSTEST_POSIX Tool Results on Client2 Machine

Test2 Scenario

- i. Putting an AVI video on the exported directory and playing it using movie player program on the client1 machine
- ii. Pushing the processor load on NFS server above 50%

Test2 Results

The movie player program stopped playing the video after a few seconds which means that PEP of our system succeeded in enforcing ongoing-condition policy related to Processor usage limitation.

C. Performance Test

The objective of performance testing procedure is to measure the additional overhead for making usage controls checks by our proposed work compared with an unmodified kernel. The following sections describe file system benchmarks that are used to evaluate the performance of our enhancements to the NFS server, and the results of performance testing.

File System Benchmarks

The following file system benchmarks are used to evaluate the performance of our enhancements to the NFS server:

- **Am-utils** The first file system benchmark we used to measure overall file system performance was Am-utils (The Berkeley Automounter) [24] version 6.1b3 which contains 430 files and more than 60,000 lines of C code.

This benchmark configures and compiles the am-utils software package inside a given directory.

- **NFSSTONE** The second file system benchmark we used was a NFS-specific benchmark called NFSSTONE [25]. This traditional benchmark performs a series of 45522 file system operations on a mounted NFS file system, mostly executing system calls, to measure the number of operations (NFSStones) a NFS server can serve in a second.

Test Results

Figure 7 shows the results of am-utils benchmark, the time values presented are the average for 10 executions; coefficient of variation = 3.6%. As can be seen, results show a small difference between all of the tests, the difference between the fastest and slowest results = 1.86%.

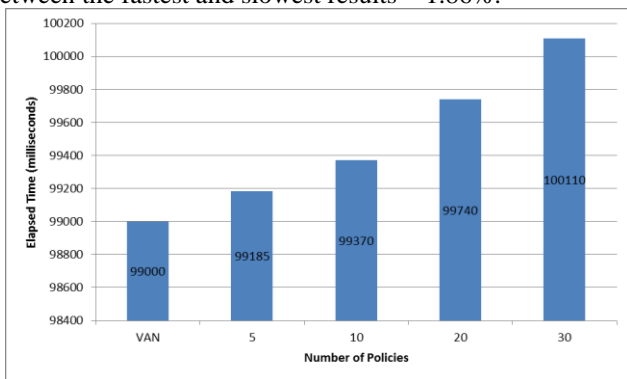


Fig.7. Results of the Am-utils Benchmark.

Figure 8 shows the results of the NFSSTONE benchmark, the time values presented are the average for 10 executions; coefficient of variation = 3.9%. As can be seen, results show a small difference between all of the tests, the difference between the fastest and slowest results = 3.18 %.

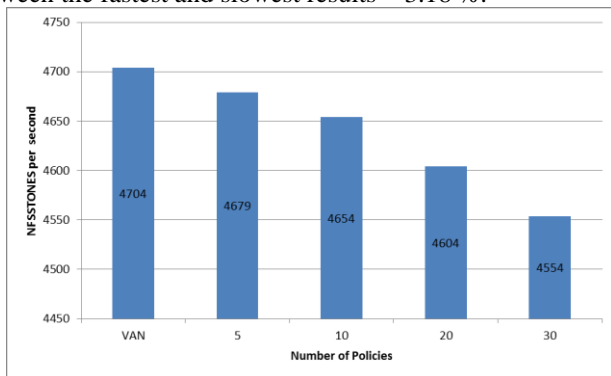


Fig.8. Results of the NFSSTONES Benchmark.

In summary, we evaluated our NFS server enhancement using both NFSSTONE and Am-utils, results showed that our security enhancement has an acceptable overhead over the vanilla kernel.

VI. CONCLUSION AND FUTURE WORK

This paper has presented an enforcement architecture design following the Sandhu's UCON_{ABC} model and implements it in the Linux NFS server. Our approach has concentrated on the architectures and mechanisms layers of OM-AM framework of security engineering. Our proposal shows the viability of implementing UCON_{ABC} model in Linux NFS server, by providing advanced UCON policies that allows various access/usage control scenarios that cannot

be achieved using the existing classical method. Our work does not aim to replace the existing access control mechanism in NFS, but rather to extend it by offering an extra decision level to it.

This work has been implemented in NFSv4.1 but the same architecture design could be implemented on all versions of NFS with minimum possible changes. Although there is space for performance optimizations but our work causes reasonable impact on the overall system performance. We believe that this work addresses the specific problems identified within the traditional access control solutions by implementing UCON_{ABC} model in a flexible way that meets the modern security requirements of the NFS.

Our future work includes the following:

- We plan on moving our implementation to the VFS layer. This way, the advantages of UCON_{ABC} model could be gained for other underlying native file systems such as EXT3 on local hosts, or with other distributed file systems.
- We plan to explore methods to improve the performance. One of these methods could be using one of the policies caching techniques to avoid reading them continuously from the disk.

REFERENCES

- [1] Sandhu, Ravi, and Jaehong Park. "Usage control: A vision for next generation access control." Computer Network Security. Springer Berlin.
- [2] Grompanopoulos, Christos, and Ioannis Mavridis. "Challenging issues of UCON in modern computing environments." Proceedings of the Fifth Balkan Conference in Informatics. ACM, 2012.
- [3] J. Park, Usage control: A unified framework for next generation access control, Ph.D. Thesis, George Mason University, Fairfax, VA, USA, 2003.
- [4] R. Sandhu, Engineering authority and trust in cyberspace: The OM-AM and RBAC way, in: In Proceedings of 5th ACM Workshop on Role-Based Access Control, ACM, 2000, pp. 111-119.
- [5] Park, Jaehong, and Sandhu Ravi (2004). The UCON_{abc} usage control model. ACM Trans. Inf. Syst. Secur., 7:128-174.
- [6] S.D.C. di Vimercati, S. Paraboschi, P. Samarati, Access control: Principles and solutions, Softw. Pract. Exper. 33 (5) (2003) 397-421.
- [7] Russell D, Gangemi GT. Computer security basics. Sebastopol, CA:O'Reilly and Associates; 1991.
- [8] Ramachandran R, Pearce DJ, Welch I. AspectJ for multilevel security. In: The 5th AOSD workshop on aspects, components, and patterns for infrastructure software (ACP4IS). Bonn, Germany; 2006. p. 1-5.
- [9] SANDHU, R., COYNE, E., FEINSTEIN, H., AND YOUMAN,C. 1996. Role based access control models. IEEE Computer 29, 2.
- [10] D. Artz, Y. Gil, A survey of trust in computer science and the semantic web, Web Semant. 5 (2) (2007) 58-71.
- [11] H.L. Jonker, S. Mauw, J.H.S. Verschuren, A.T.S.C. Schoonen, Security aspects of DRM systems, in: 25th Symposium on Information Theory in the Benelux, 2004, pp. 169-176.
- [12] Security frameworks for open systems: Access control framework. Technical Report ISO/IEC 10181-3, ISO (1996).
- [13] J. P. Anderson. Computer security technology planning study volume II, ESD-TR-73-51, vol. II, electronic systems division, air force systems command, hanscom field, bedford, MA 01730. <http://csrc.nist.gov/publications/history/ande72.pdf>, Oct. 1972.
- [14] Andrew, Tanenbaum S., and Steen van Maarten. Distributed systems-principles and paradigms (2." 2007).
- [15] S. Shepler, M. Eisler, and D. Noveck. Network File System (NFS) Version 4 Minor Version 1 Protocol. RFC 5661 (Proposed Standard), January 2010. URL <http://www.ietf.org/rfc/rfc5661.txt>.
- [16] M. Xu, X. Jiang, R. Sandhu, X. Zhang, Towards a VMM-based usage control framework for OS kernel integrity protection, in: SACMAT'07: Proceedings of the 12th ACM Symposium on Access Control Models and Technologies, ACM, New York, NY, USA, 2007, pp. 71-80.

- [17] He, Ruan, Marc Lacoste, and Jean Leneutre. "Virtual security kernel: A component-based os architecture for self-protection." Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on. IEEE, 2010.
- [18] Teigão, Rafael, Carlos Maziero, and Altair Santin. "Applying a usage control model in an operating system kernel." Journal of Network and Computer Applications 34.4 (2011): 1342-1352.
- [19] <https://en.wikipedia.org/wiki/Procsfs>.
- [20] Bell, D. and LaPadula, L.: Secure computer systems: Mathematical foundations and model. MITRE Report, 2(2547) (November 1973).
- [21] A. Lazouski, F. Martinelli, and P. Mori. Usage control in computer security: A survey. Computer Science Review, 4(2):81 – 99, 2010.
- [22] <http://www.citi.umich.edu/projects/nfsv4/pynfs>.
- [23] <http://wiki.linux-nfs.org/wiki/index.php/NFStest>.
- [24] <http://www.am-utils.org>.
- [25] B. Shein, M. Callahan, and P. Woodbury. NFSSTONE: A network file server performance benchmark. In Proceedings of the Summer USENIX Technical Conference, pages 269–275, Baltimore, MD, Summer 1989.