Enhanced Security for Monitoring Services in Large Scale Distributed Systems

Florin Pop, Alexandra Arcalianu, Ciprian Dobre, Valentin Cristea
Faculty of Automatics and Computer Science, University Politehnica of Bucharest, Romania
Splaiul Independentei 313, Bucharest 060042, Romania
Emails: florin.pop@cs.pub.ro, alexandra.arcalianu@cti.pub.ro, ciprian.dobre@cs.pub.ro, valentin.cristea@cs.pub.ro

***Abstract*—Today computing systems employ high interaction between people and services. It is more than obvious that todays society depends every day more on such computing systems. Furthermore we cannot deny that consequences of their failures can have high cost, from financial resources loss to even human lives loss. Modern large scale distributed systems services must provide guarantees for protecting services against malicious threats. So security is becoming more important, as services can become victims of malicious attacks that can result in code injection that can affect critical components on a large scale. Because of such catastrophic consequences it has become mandatory for developers to provide security guarantees for users, usually in the form of protection mechanisms that can protect the system against the malicious behavior of services. In this paper we propose one such solution to provide security mechanisms and guarantees against security threats in large scale distributed environment. Such mechanisms consider the needs of modern distributed service-oriented applications. We propose security policies designed to handle security guarantees imposed by such applications. In the end, we present experimental results that reflect the capacity of the proposed security mechanisms and policies to correctly handle a wide range of security threats.**

I. Introduction

Distributed systems are environments where services are deployed to provide various functions. Their dependability is today regarded as a key issue by the research community. One way to assure dependability is through security. In this paper we propose an enhanced security architecture designed for monitoring systems. The reason of securing monitoring services is because they are essential in distributed systems to provide near real time information about the state of the system, which is further used to take important decisions throughout all the system.

Even if today's security efforts for protecting services start from the premise that the security mechanisms provided by the mainstream of operating systems are enough, we propose a different approach. The discretionary access mechanisms of the current operating systems have their weak points that can be fixed using mandatory access control solutions. Our solution proposes security policies for two monitoring services one that can run as standalone and one that need a container to run in. In order to protect the computer system against malicious behavior of the two services we sandbox them to run in domains using SELinux [10] that is an implementation of the mandatory access control model.

To ensure the security of the monitoring services, the proposed solution sees them as processes running within the operating system. We evaluate the use of a policy designed to allow a monitoring service to collect information from proc directory and send it using datagrams, whilst denying other actions. Likewise, we evaluate a second policy that only allows a monitoring service hosted by a web services container to send information to a proxy service. The effect of the security policies propose is that the two services targeted are allowed to do only the actions that are specified in the policy.

When building security policies for services that interact with the two monitoring services secured by our policies it is sufficient to consider only the damages that can be done by the former. This happens by adding in a secure system the two services while the system remains secure.

The main research finding is that the damage made by a malicious behavior of the two monitoring services on the computer system that hosts them can be confined using the security policies proposed. That is why when designing a balanced set of security mechanisms for protecting a monitoring service it is mandatory to include the one that provides mandatory access control at the operating system level.

Not less important is the fact that security is an essential characteristic for a dependable system along with reliability, availability, safety and maintainability and that is why the interest of research community of computer science for the domain of security is growing every day. We focus on enhancing security for monitoring services because they are essential for the normal functioning of a system by providing information about the state and the resources of the system. In a distributed system the Grid monitoring services are an important component through them it is possible that the best resources are delivered to a certain computational set of tasks.

The manner proposed to solve the stringent problem of services security is motivated by the present results in the field of security research. It is widely known that in order to assure security for a computer there are four main directions to be followed and each one acts at a different layer. These four layers are: application layer, hardware layer, network layer and OS layer. From all these layers the solution of securing services proposed by this paper lies at the OS layer. We have opted for the OS layer and not for any of the other layers for the following reasons. Compared with a solution at the application layer a solution at the OS layer has less overhead. Also choosing to device a security policy at OS layer has definitely lower costs than securing at hardware layer. Beside

all these an OS security solution is more flexible than a network solution because of the high number of constraints from the network layer.

For the modern computing systems the security features provided by the wide majority of operating systems are not enough to assure safe execution of untrustworthy services. One significant warning in this sense appeared in 1998, when the article The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments warned that as long an application is not properly secured by the operating system it is not free from failure. For confining the damage made by malicious applications it is mandatory to protect the computing system with some security features that are unanimously accepted at theoretical level but poorly used in the mainstream of the operating systems.

Even if public awareness of the need of security is every day increasing, still today security efforts do not know significant advances. One important reason for this is the assumption that adequate security for applications can be achieved using the existing security mechanisms of the mainstream operating systems. There are some security features that lack from these operating systems and we encourages their usage. Along the years, to provide security at the OS layer there has been a continuous research that constantly discovered new security mechanisms. It is a known fact that most computer systems of our days are constructed using OSs that implement Discretionary Access Control (DAC) mechanisms [7]. As any other security solution DAC is not a perfect one. The main issue of DAC, as the name suggests, is that the access permissions are at the discretion of the users and this can lead to serious security breaches.

Because protecting a system only with what DAC offers is not possible at the OS layer, it is necessary to draw on the concepts of Mandatory Access Control (MAC) theoretical model [7]. Domain separation, type enforcement and roles are some of the pillars upon which improved security solutions can be build in the OS. There are implementations of MAC that currently are integrated in OS over the DAC mechanism with very good results. Examples of such implementations are SELinux, AppArmor and Smack. The todays leader solution for providing security is SELinux that was designed with the idea to create a MAC solution flexible enough to support a wide variety of security policies in real-world environments. Having as drawback its high complexity it is often replace with simpler MAC implementations such as AppArmor [8] which restricts the access of programs to a limited set of files and is a path based type of security. Smack [9] like SELinux provides security labels for the inodes of files and not for their path but is much simpler than SELinux because is intentionally design to solve smaller security problems.

From all these three solutions, in order to design an enhancing security for monitoring services, we have chosen SELinux. It provides various MAC mechanisms such as type enforcement control, role based control or multi level security control. Also, compared with AppArmor or Smack, there is at least one distribution of OS on which SELinux is a mature security solution. There is not a security solution that can cover all the issues. A balance of multiples security solutions must be found in order to assure safety for a system. The effort of securing monitoring services will not offer a strong protection until it is completed with new features of the security at the OS layer, features that today are used by a narrow community.

The paper is organized as follow. The security issues with which monitoring services are confronted on a single computer system are identified in the Section II. Section III covers the first goal of proposing a security policy for a standalone application and implementing it with SELinux. The second goal of creating a security policy for a web services container is reached in Section IV. The test scenarios highlighting the most likely attacks to occur and demonstrating the protection provided by each of the two security policies proposed in this paper can be found in Section V. The findings of current research and future work are discussed in Section VI.

II. Security Issues for Monitoring Service

Beside the security mechanisms from the OS layer a computer system is sometimes protected against abnormal behavior of the services often used in distributed system by the security policy particular to each service. Even if usually services are not written by security specialists they are however provided with security policies more or less effective.

One simple example of such a policy is that of the monitoring service using ApMon API. There are two paths of protecting the monitoring service accordingly to [1]. The first alternative is to establish a password between MonALISA services and the application which uses the ApMon API for monitoring the local system. Every datagram send from the local host to MonaLISA service must contain that password. The second alternative is to configure the MonALISA service to receive datagrams from IPs within an ACL.

Also the computer system in question is also protected against untrustworthy services thorough the security policy of the container which runs the services, of course for those services that need such a container. An special interest towards securing shows the Tomcat web services container as stated in [3] and [2]. For example version six of the Tomcat container has its one Java Security Manager which allows the web browser to run the applets in sandboxes in order to avoid untrusted code to access files from the local system or to avoid that the applet connects to a different host from where it has been loaded and so on.

Other security policies like those specified by the Grid Security Infrastructure (GSI) defined in [4], [5] as a complex combination of security solutions like credentials using standard X.509 certificates, authentication algorithm defined by Secure Socket Layer protocol, rights delegation through Proxy certificates, all destined to assure security in the Grid environment. The set of solutions are provided by the Globus container which is the support for Grid services. Grid services are a special type of web services with further characteristics like life cycle management and the property of having a state associated. The Grid monitoring services are extremely

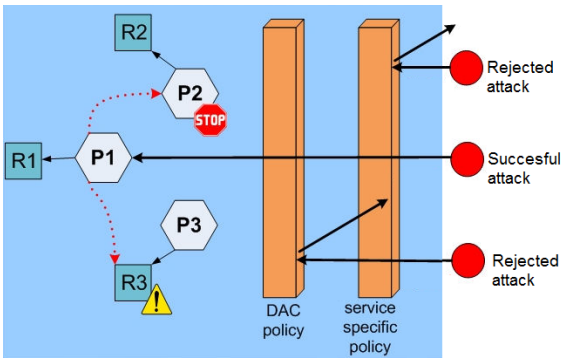

Fig. 1. Malicious attacks against services

important in a Grid system because they provide valuable information about the resources within that Grid and providing secure for this type of services is vital.

In this paper we propose a possible solution for enhancing security for monitoring services in distributed systems. The solution was found following the principle according to which a system is as secure as its weakest chain link. That is why the distributed systems were decomposed into smaller components, until the dimension of a single computer system. The security issues of the monitoring services did not delay in clarifying. In [6] Foster established four critical features for Grid monitoring services. They must be reliable, timely, scalable and secure. The last of the features remains valid regardless of the monitoring services type.

In order to secure a computer system it is mandatory to provide solutions for each one of the four layers: application layer, OS layer, network layer and hardware layer. Security mechanisms at the application layer have the advantage of high granularity, while protecting sensitive information and permit construction of complex policies. The disadvantage of a such mechanism is the high overhead. The OS is the one that mediates accesses initialized by applications to hardware components. Therefore, access control mechanisms at the OS layer can provide high level of granularity for protecting processes, files, sockets and so on. The security mechanisms from hardware layer are usually the ones with the highest costs, but as advantage have the small overhead the security policy being implemented directly in hardware. The network layer security has to be provided to ensure that only valid data packets are accepted and malicious traffic is not allowed to interact with OS or applications.

According to all presented aspects we want to demonstrate that for assuring the services security at the OS layer, beside the common solution of DAC mechanisms, there is place for improvement. The solution proposed here does not intend to remove the mechanisms protecting the services already exiting at OS layer or any of the other layers, but rather to create an orthogonal security policy which will enhance the security degree at the OS layer.

By choosing to device a security policy at the OS layer for services we analyze services as simple processes running on a computer system. By reducing the service to a simple process we can localize better the security issue with which is confronted a service from the OS point of view. Let's consider three services, regarded as processes, run in a computer system (see Figure 1). Each of these processes have associated some resources, that can be files, sockets and so on. For example process P_i has permissions to access the resource R_i . When running these three processes are protected first by the DAC mechanism of the OS and by a specific policy for every service. The latter policy is particular to each service and sometimes can even not exist. These two walls of protection are the barrier between processes and malicious attacks. There are various situations. A desirable case is the one of an attack stopped by the security policy specific to service. One least desirable is a malicious attack that crosses the policy particular to the service and it is rejected by the DAC mechanism. But there are attacks that can be blocked by neither one of the two security walls. Such attack affects the process P_1 and the process starts to behave maliciously. It kills process P_2, accesses the process P_3's resource and furthermore can compromise the entire system.

The proposed solution is enclosing the attacked service inside a sandbox that does not contain the other two processes. In this case even it begins to have an abnormal behavior the actions of the service are limited by the sandboxs boundaries and the process P_1 can neither kill process P_2 nor accessing P_3's information that is not suppose to touch.

As the prefix from its name suggests, SELinux is an enhanced security solution its purpose not being to replace other solutions but to improve the current security policies. Even if SELinux is a very powerful security solution it is not used quite often for protecting computers inside a distributed system mainly because of its complexity. We come in preventing this attitude by proposing security policies for two softwares that are frequently used in dependable systems. On one hand we specify a security policy for an application using the ApMon API. This API permits monitoring of a local host and sending information about the system to a MonaLISA service. We have chosen this type of application because monitoring is an usual task perform inside dependable systems and in order that the monitoring information be useful it is high demanding not to be tainted. Another reason for choosing this API is because it is provided by the open source community.

On the other hand we intend to enhance security for a middleware solution. This is an important software component for any distributed system, not only because it is largely used, but because it is the reference implementation of container of servlets and JSPs. We aim to secure the Tomcat regarded as a whole, as a container. This step is necessary to secure services, because most services used in computers systems need a container in order to achieve their activities. The problem with this approach was that the requirements of the services running inside Tomcat are far too different to permit a general enhancing security for services. This is why the better solution was to focus on securing Tomcat as a specific container for one single service. The chosen service was a proxy service.

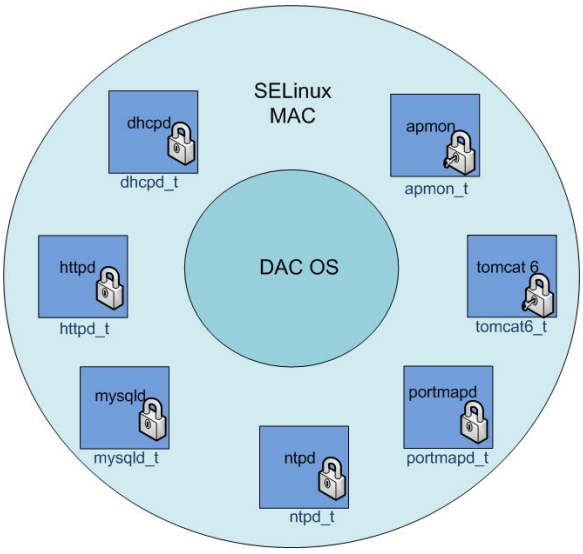

Fig. 2. Services running locked in sandbox

The importance of this research relies on the fact that it encourages enhancing security for software often used in distributed systems especially by the research community. Over the DAC security layer provided by any Linux OS there is the MAC security layer assured by SELinux. The SELinux solution already confines twelve daemons inside specific domains. These twelve daemons are enumerated in system requirements chapter. Anything else beside those daemons acts in the OS as if the SELinux layer does not exists. This is the case of an application that uses ApMon API or the tomcat web services container. These two example run in the so called unconfined domain from the SELinux point of view, that is they are fenced only by the DAC layer.

Figure 2 presents results from our research: a computer system with an enhanced security status because beside been protected from the damage made by the twelve daemons further it offers protection against damage made by the Tomcat container and the monitoring application that uses ApMon API. This is done by locking them in specific sandboxes according with their activities and they can not reach anything outside the sandbox's limits.

III. Security Policy for a Standalone Application

This section presents the first goal to propose a solution for protecting the system that hosts a service against the damage that can be made by this service if at a certain moment has a malicious behavior. The service in question is a standalone application that monitors the system and sends the monitoring data to a collecting destination. If the monitoring service has been break down by an attacker at a certain point will start to not behave as stated in its specification. The consequences of this behavior range from a disclosure of confidential information to severe problems that do not permit the system to work properly. As stated before the purpose is not replacing the already security solutions for monitoring services with a new one, but rather to enhance the existing

Object(Resource)	Object Class	Type
/bin/apmon	file	apmon_exec_t
/bin/destinations_s.conf	file	apmon_conf_t

The types from SELinux policy for apmon

Role	Domain	Type	Permissions
demo_r	apmon_t	apmon_conf_t	read, open
	demo_t	apmon_exec_t	execute

The domains from SELinux policy for apmon

Fig. 3. The types and domains from SELinux policy for apmon

level of security in those places where there is room for improvement.

ApMon is a flexible set of APIs that provides for an application the possibility to send monitoring information to MonALISA services. The system monitoring data include the current values for parameters like CPU load, the number of processes currently running, the amount of free memory, disk usage and so on. All these values are obtain from `/proc` filesystem. The monitoring data is sent to one or more hosts running MonALISA services using UDP datagrams. The use of the UDP protocol brings the advantage of high communication performance. Also it is not a problem if some of the datagrams get lost because the main concern of this API is to obtain statistical information about a system and not to record all the data about a system. The fact that for communication it is used the UDP protocol is extremely important for the policy built in proposed solution.

In order to ensure security for the APMon there is a policy that filters the ApMon datagrams that can be received by the MonALISA service. The first alternative is to establish a password between the Apmon and MonALISA entities and every datagrams must contain that password. The second alternative is to configure MonALISA service to receive datagrams from IPs within an ACL. As can be observed the above policy does not protect at all the localhost from which the monitoring information is gathered. That is why this paper proposes a policy which provides protection for the system that hosts the apmon application.

Given the above specification here are some security attacks that should not affect the system. An attempt to access other files except those from `/proc` filesystem from the part of a malicious apmon application should be denied. Also using TCP socket to bind and listen inside apmon application must be stopped. These attacks and some other are discussed in the test scenarios section and there we will see how the proposed security specification responds to attacks.

The first step in providing security for any application in a SELinux system is to close the application inside a sandbox that is its own domain. The role of the domain is to confine any damage made by the application. In order to obtain that it is mandatory to establish the domains and the types required

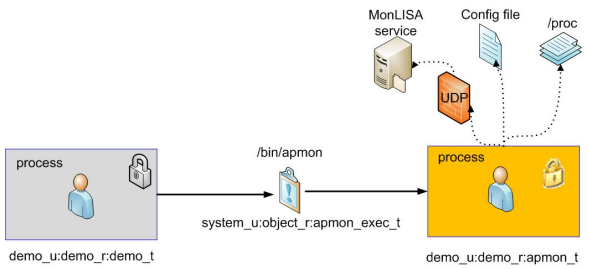

Fig. 4. Transition from demo_t to apmo_t domain

by the security specification. Of course we will use domains and types already defined in the reference policy but we will define also our own, specific for the apmon application. The new types introduced by our security policy are assigned to the resources which the application must have access to, resources that can not be found default in a Linux computer. For the default resources of the system like etc directory or proc directory the reference policy provides already types and domains. The types proposed for securing apmon application with SELInux are organized in Figure 3.

The entry point of the `apmon_t` domain is any executable file labeled with the type `apmon_exec_t`. Once a process executes this file it transitions in the `apmon_t` domain and runs only under the allow rules of the domain as can be seen in the Figure 4. Here the executable file is `/bin/apmon` and the allowed actions are reading the configuration file, accessing `/proc` contents and sending monitoring information to MonALISA service using UDP protocol. For individualize the configuration file among other files in the system we must labeled it with a specific type, in this case we have chosen the `apmon_conf_t` type.

Once the roles, domains and types have been set the next step is to code them into policy. This is done through the use of the TE rules and RBAC rules. AVRs, transition rules and RBAC rules are used to authorized access permissions described in the two tables from Figure 3.

IV. Security Policy for a Web Services Container

The purpose of this paper is to enhance security for monitoring services. These services are the ones running usually in distributed systems. Some of the services do not need special middleware to run for example those running as simple applications. But there are some which need a container in order to function such as web services. That is why we need to extend our enhancing security research and on web services containers (see Figure 5).

The services can vary from proxy services, file transfer services to monitoring services, equally the requests of these services can differ. What a service is allowed to do and what not differ sometimes too much for treating enhancing security for monitoring web services unitarily and independent from the container. A proxy service may need access to some type of files, while a monitoring service wants for example access to other types. Both run as java applications requiring searching

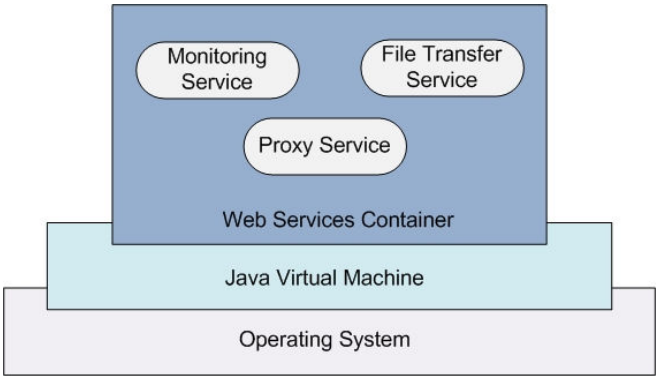

Fig. 5. Web Services inside Container

and loading dynamic libraries, memory execute permissions and network access. But one web service for monitoring requires restrictions as network interaction only with its service replicas for example meanwhile the file transfer web service must send a certain file over the network.

If we choose to create a generic security policy for the container and considering as a must that it has to host a wide range of web services we must allow the container to do almost anything and such security policy would not be protective at all. This situation is similar even if we refer at a generic security policy only for monitoring web services. That is why proposing a security specification for container means consequently particularizing the container of web services for a specific service. The different requirements of web services that often are contradictory from the perspective of deciding what the container is allowed to do and what not, gives no alternative than choosing one single service targeted by our security specification.

The version 6 of the Tomcat web services container is focused on security and all the security parameters can be found and set from catalina.policy file. Precisely this file provides fine grained access control to a Tomcat server administrator through the built in security model of Java. As explained in [17] when starting Tomcat in security mode the grants from this file allow services running inside Tomcat to access a specific set of directories. The problem highlighted also in [17] is that the Tomcat instance is not the only one having access to those directories but the entire computer system. It is vital that those directories be protected by the operating system and therefore the scope of our research. We intend to protect Tomcat resources not only through the usual discretionary access mechanisms but also using mandatory access controls.

The target of our security specification is to protect the local system hosting the web container from failings of the monitoring service and to stop attempts which are not included in the official actions of the monitoring service because these attempts have the potential of security attacks.

We have already established that we can not device a generic security policy for the Tomcat web services container because our policy must be built only by allowing the services to behave as specified and doing nothing else extra. If we want to

Object(Resource)	Object Class	Type
/usr/sbin/tomcat6	File	tomcat6_exec_t
/usr/…/java	File	java_exec_t
/usr/…/WEB-INF/classes/proxyConfig.xml	File	tomcat6_conf_t

The types from SELinux policy for Tomcat container

Fig. 6. The types from SELinux policy for Tomcat container

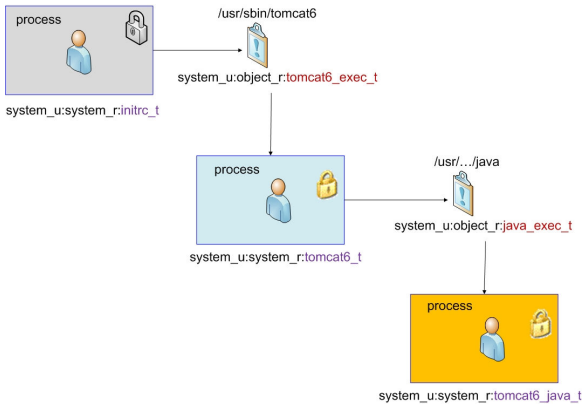

Fig. 7. Process transition from initrc_t to tomcat6_java_t domain

secure Tomcat as container for wide variety of web services we will end up by proposing a policy that allows everything and such a policy is absolutely useless. For this reason we restrict the Tomcat container as a web services container that hosts only one monitoring service and that service is the monitoring and balancing from the replication solution discussed above. From the OS perspective this restriction is reflected as a process with the following actions allowed and anything else denied. First the constraint container will run as a java process with the usual behavior allowed: searching, reading and shared libraries, accessing information about kernel state and network within proc directory, accessing tmp directory.

We have already established what the process represented by the Tomcat container inside which the monitoring component of the proxy service is allowed to do. It is high time to device a SELinux security policy according to those allowed actions. The first step is defining the domains and types of the policy that will help to sandbox the restraint Tomcat container. The Tomcat process will run in local system as daemon that means it will be started from `initrc_t` domain. The resources that trigger the existence of domains created by our policy are those from Figure 6.

The main types introduced by our policy are the ones from Figure 6. The type `tomcat6_exec_t` is the type for the Tomcat running script and its execution determines the first domain transition from Figure 7. The type `tomcat6_conf_t` is the type that protects the configuration file of the monitoring component of the proxy service from being accessed by unknown processes. According to our policy in the system the only process that can access this type of files must be in `tomcat6_java_t` domain. This is the case of the services running inside Tomcat and the monitoring service is the only service running inside it.

V. Experimental Scenarios

This section demonstrates how the security specifications for the application apmon using ApMon API and for the Tomcat container hosting only the monitoring service enhance the security of a computer system. The scenarios present situations in which attacks from malicious code can be contained and avoid the damage of the whole system though validating the two policies proposed. It is true that by an attentive filtering of network traffic most of the following attacks would be discovered and new protecting mechanisms can be enabled, but the damage can not be avoided in the first instance. With the SELinux policies proposed the damage can be avoided.

All the actions that do no correspond to the SELinux allow rules are logged as AVC denials. An AVC message from a specific action is the result of not finding a rule that allows the action in the Allow Vector Cache. As the name suggests this cache vector of rules is used to minimize the overhead introduced by the SELinux rules in the OS. A useful tool provided by Fedora 13 that inspects the log file for SELinux AVCs is setroubleshooter which interprets the AVC messages and displays them in an easier manner to understand.

For testing the SELinux security policies it is not enough to load them in the system. It is also necessary to configure the SELinux identity for the users accounts, precisely to map the SELinux user from the policy with the usual Linux user. Also for working properly the SELinux policy for Tomcat container hosting only a monitoring service needs besides being loaded to have configured the range of ports 40000-40003 to the port type specified in policy. All these actions can be run through scripts constructed by us, loadPolicyApmon and loadPolicyTomcat6 using mainly the semanage tool.

A. Test scenario 1

The first scenario for any application protected by a SELinux policy is to verify if the application still works. That is if the policy specifies enough rules for the application to be able to perform its task. As long as the set of rules specified by the policy is not complete the targeted application can not work properly.

This scenario follows the next steps. The apmon application sends information about the system inspecting `/proc` directory. Precisely this application is an example of using ApMon API as a thread in background which collects and sends monitoring data. MonaLISA farm has received information about the monitored system. Our policy allows the monitoring service apmon to perform its background job of collecting information about the local system from `/proc` directory and sending them to the MonALISA service.

B. Test scenario 2

This scenario verifies that the system equipped with the policy for apmon application will deny the request of con-

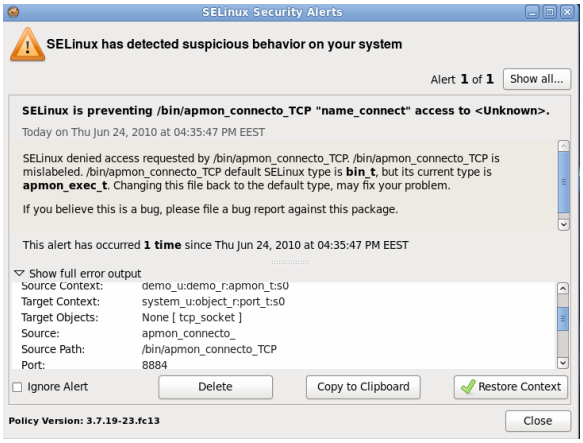

Fig. 8. The malicious apmon application is denied connect access to port 8884

necting on a TCP port from the part of the apmon application and will log the denial appropriately. This scenario stands for an application code attack. The malicious executable `apmon_connecto_TCP` is the result of inserting some code in the original apmon code. This extra code tries to establish a TCP connection on a certain port. This should not be allowed because our policy should only grant permission to create TCP socket and to inspect the local IP addresses, but should deny any attempt to connect on a TCP port.

The connect attempt has failed as can be seen in Figure 8. In this way the policy for apmon application has avoided that information from local host reach a destination other than MonaLISA service from the IP address just mentioned.

The source of the attempt can be found in the source context field from the AVC `demo_u:demo_r:apmon_t`. The SELinux alert shows that the application running under `apmon_t` domain has no permission to connect to port 8884 which is a resource of type `port_t` as the target context suggests.

C. Test scenario 3

The third scenario verifies that the system protected by the proposed policy for apmon application will deny any request of binding and listening on a TCP port received from the part of the apmon application. This scenario simulates also like the previous scenario a code attack. That is the original apmon code is modified in such a manner that when running beside its usual task will try to become TCP server and to potentially accept connection from outside the system. It is more than clear that such attack will have unconfined damages if the system is not protected by a correct policy.

The policy served this time for blocking possible connections from outside in the system without the knowledge of the user. This conclusion can be drawn looking at source context that surprises the `/bin/apmon_listenTCP` running in `apmon_t` sandbox when it attemps to bind on port 8884. The socket of type TCP is the targeted object that the application is trying to reach for binding but there is rule for allowing that.

D. Test scenario 4

Through our security specification for the Tomcat web services container hosting only a monitoring service we have transformed this container in a sandbox of safety for the monitoring service. Now it is high time to perform some tests on our SELinux policy in order to validate it. The first target of the test scenarios is to suppose a malicious attacker has taken control over the monitoring service component of the proxy service running on a computer system protected by our policy. He has the possibility to permit remote connection on local host system by just binding on some ports, other than those from the range 40000-40003, this being the allowed range to use for the monitoring service.

The SELinux policy proposed by us stands up and blocks the service attempt to bind on port 8884. This time the SELinux alert attentions that the process from the domain `tomcat6_java_t`, that is Tomcat, forces an access on a resource that is the port 8884 with `port_t` type. Our policy allows only the use of ports of `tomcat6_port_t` type for services running inside Tomcat.

E. Test scenario 5

Another objective of fencing all the actions made by Tomcat web services container is to contain the damage of the following situation. The monitoring service inside Tomcat has suffered an attack and as a consequence it behaves maliciously and tries to send valuable information about requests received from the proxy service somewhere else than it is supposed to. The test service is a simple one. When request is invoked through the browser this service intends to connect on a different port than those from the range 40000-40003. As example it has been chosen the port 9090 as port to connect to.

The reaction of the security policy is as expected thus the attempt of connecting on port 9090 has failed because the action was not specified as an allowed action. As in the previous scenario source and target involved are the `tomcat6_port_t` domain as source context and `port_t` type as targeted context. The test service trying to connect is identified by the SELinux alert as the java process trying to connect to port 9090 although is has not permissions to do that.

F. Test scenario 6

Continuing with the set of tests it would be helpful to know our security policy responds to an attempt of an exec inside tomcat. Let suppose the proxy is down again and the malicious attack wants to execute something. To simulate this scenario we constructed a test service that creates a process that executes the well known shell command `ps`, using `Runtime.getRuntime.exec` method.

As in the previous scenario setroubleshooter does help at well and it is necessary to check the log file. Here we found out why the test service has failed in executing the `ps` command.

The reason of the AVC message from the log file is that the java application is denied to execute the bash file. The denial is destined for our test service because as can be seen bellow the domain of the process is `tomcat6_java_t` and the services running inside tomcat container are the only one marked with this domain. As expected looking at the denied execute the type of the bash file is `shell_exec_t`, that is a type indicating that this file is an executable.

VI. Conclusions and Future Work

Throughout this paper we have proposed a solution for limiting the damage made by an abnormal behavior of a monitoring service on a single computer system. Even if our results are limited are limited on a single computer however the impact of this limitation can only signify a security enhancing and at distributed system level. Also if the two secured monitoring services are integrated in a system already considered secure then the whole system could be considered secure.

The main finding of this research is that at OS layer of security there is room for security enhancement because there is a fact that most of the computer systems implement at the OS layer only the discretionary access controls. This paper is somehow opposite to the security efforts of our days, characterized by the limited supposition that the security offered by the existing operating systems is quite enough and for enhancing security it must be searched somewhere else. An adequate protection against the malicious action of the monitoring services has been provided through the mechanisms of SELinux like type enforcement and role based access control at the OS layer.

When enhancing security for monitoring services that do not need a container to run in and run as simple applications on a system the security policy is simpler to implement, understand and debug because the requirements of the container do not have to be analyzed.

An important finding is that devicing mandatory access control security policies for a container of services that is intended to run in several services would lead at a security specification that allows almost anything. Such security specification would be useless and thats why the containers function has to be limit to hosting one single service. On this new configuration the security policy must be applied.

As future work the proposal of security enhancing at the OS layer using mandatory access mechanisms can be extended over other monitoring services and over services in general. In this way a computer system will be safe from the damage made by not only the twelve daemons by default restricted by a specialized SELinux security OS distribution and the two new protected monitoring services proposed in this paper. Also the current work can be continued with implementing mandatory access control security policies for services with using other technical solutions like AppArmor or Smack. Even though to be successful on this direction it is mandatory that the Smack security solution become trustworthy and mature on at least one operating system distribution. If possible it should be included by default in the mainstream of some operating systems distributions like SELinux is on Fedora and AppArmor is on Ubuntu distribution.

Acknowledgment

The research presented in this paper is supported by national projects: "SORMSYS - Resource Management Optimization in Self-Organizing Large Scale Distributes Systems", Project CNCSIS-PN-II-RU-PD ID: 201 (5/28.07.2010) and "DEPSYS - Models and Techniques for ensuring reliability, safety, availability and security of Large Scale Distributes Systems", Project CNCSIS-IDEI ID: 1710 (618/15.01.2009). The work has been co-funded by the Sectorial Operational Program Human Resources Development 2007-2013 of the Romanian Ministry of Labor, Family and Social Protection through the Financial Agreement POSDRU/89/1.5/S/62557.

References

[1] Iosif Legrand, Ramiro Voicu, Catalin Cirstoiu, Costin Grigoras, Latchezar Betev, and Alexandru Costan. 2009. Monitoring and control of large systems with MonALISA. Commun. ACM 52, 9 (September 2009), 49-55. DOI=10.1145/1562164.1562182

[2] Mathew Moodie, Pro Apache Tomcat 6. The Expert's Voice in Java Technology, Apress, USA, 2007.

[3] Vivek Chopra, Sing Li, Jeff M. Genender, Professional Apache Tomcat 6, Wrox/Wiley Pub., 2007, ISBN: 978-047-175-3-612.

[4] May Phyo Oo and Thinn Thu Naing. 2007. Access Control System for Grid Security Infrastructure. In Proceedings of the 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Workshops (WI-IATW '07). IEEE Computer Society, Washington, DC, USA, 299-302.

[5] F. Siebenlist, R. Ananthakrishnan, D. E. Bernholdt, L. Cinquini, I. T. Foster, D. E. Middleton, N. Miller, and D. N. Williams. 2009. Enhancing the earth system grid security infrastructure through single sign-on and autoprovisioning. In Proceedings of the 5th Grid Computing Environments Workshop (GCE '09). ACM, New York, NY, USA, , Article 13 , 8 pages. DOI=10.1145/1658260.1658278

[6] Ian Foster and Carl Kesselman (Eds.). 2004. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[7] Sylvia Osborn, Ravi Sandhu, and Qamar Munawer. 2000. Configuring role-based access control to enforce mandatory and discretionary access control policies. ACM Trans. Inf. Syst. Secur. 3, 2 (May 2000), 85-106. DOI=10.1145/354876.354878

[8] Mick Bauer. 2009. Paranoid penguin: AppArmor in Ubuntu 9. Linux J. 2009, 185, pages.

[9] Bryan D. Payne, Reiner Sailer, Ramon Caceres, Ron Perez, and Wenke Lee. 2007. A layered approach to simplified access control in virtualized systems. SIGOPS Oper. Syst. Rev. 41, 4 (July 2007), 12-19. DOI=10.1145/1278901.1278905

[10] Boniface Hicks, Sandra Rueda, Luke St.Clair, Trent Jaeger, and Patrick McDaniel. 2010. A logical specification and analysis for SELinux MLS policy. ACM Trans. Inf. Syst. Secur. 13, 3, Article 26 (July 2010), 31 pages. DOI=10.1145/1805874.1805982