Third International Symposium on Information Assurance and Security

Enhanced Availability and Security by Rate Control Using Extended Policy Framework in SELinux

Pravin Shinde, Priyanka Sharma, Srinivas Guntupalli
CDAC, Mumbai
{pravin,priyanka,srinivas}@cdacmumbai.in

Abstract

In this paper we discussed an extension to Security Enhanced Linux (SELinux) to build a more available and secure system that has the capability to contain and mitigate Denial of Service (DoS) attacks by exercising rate control over resource usage. We presented an extended structure to compliment Mandatory Access Control policies of SELinux. Using this extension a system's resource usage by various entities can be kept under control, leading to a more available system.

1 Introduction

Attacks on computing systems can be broadly classified into two categories. First, attacks that exploit the existing vulnerabilities in the systems and make the systems do what they are not expected to do. The second category being, attacks that abuse the privileges granted and hog the system and prevent genuine users from accessing the services, the system is supposed to provide. Flood based DoS attacks fall into the second category. Methods available in the literature for protecting systems against the attacks that belong to the first category suggest proper access control mechanisms and creating systems with out any vulnerabilities. And protecting systems against the second category of attacks requires control on usage of system resources.

DoS attacks are malicious attempts to make a system or network unusable and can be realized in several ways. Flooding a network service with requests, consuming too many resources in a system are a few of them. A network-based attack intentionally saturates the system resources with increased network traffic. An assault coordinated across many hijacked systems by a single attacker is called as Distributed DoS (DDoS). The malicious workload in network-based DoS attacks comprises network data-grams or packets that consume network buffers, CPU processing cycles, and link bandwidth. When any of these resources form a bottleneck, system performance degrades or stops, impeding legitimate system use. Overloading a Web server with spurious requests, for example, slows its response to legitimate users.

[8] classifies DoS attacks into Protocol Attacks and Brute Force attacks based on the vulnerability they are trying to exploit . Protocol Attacks try to exploit a particular vulnerability present with the protocol. Brute-force attacks are performed by initiating a vast amount of seemingly legitimate transactions. Fixing the protocol vulnerabilities pushes the Protocol Attacks into Brute Force category. To build protection against Brute Force attacks need an early detection of such attacks. The method we presented in this paper builds a robust system which works against both kinds of attacks by containing them and creating a more available systems.

The organization of the paper is as follows. In section 2 i.e., related work, we did literature survey and presented the existing methods to detect and mitigate DoS attacks. In Section 3 we presented Security Enhanced Linux (SELinux) in which we had built an extension to mitigate DoS attacks followed by the approach we had used, and experiments and results in Sections 4 and 5.

2 Related Work

The problem of detecting and protecting against DoS has been investigated for quite some time. [10] discusses the existing strategies that generally fall into two categories .

- Solutions that suggest use of adding additional resources to mitigate the attack
- Solutions that attempt to differentiate between legitimate and malicious (or anomalous) use

While the former approach is often used in practice to mitigate the effects of an ongoing attack, most of the academic literature has focused on the latter category.

Solutions that suggest use of additional resources typically involve outsourcing some of the jobs to keep the system free from the extra load. [16] proposes use of secure

0-7695-2876-7/07 $25.00 © 2007 IEEE
DOI 10.1109/IAS.2007.17

IEEE computer society

entities called bastions to do the outsourced job. Some of the solutions in this category also advocate extra control to hosts over other network entities. [6] proposes a bi-fold solution based on the Internet Indirection Infrastructure, and an IP-based solution in which hosts insert filters at the last hop IP router.

Solutions that try to differentiate between legitimate and malicious use can be largely classified into categories,

- Solutions that build models by learning the normal behavior of the usage and validate the regular usage against such model

- Solutions that find anomalies by monitoring continuous changes in the usage without any assumption of normalcy. Here, sudden and significant change in the use is considered as an attack

Both of the above approaches suffer from potential to generate false alarms during *flash events*, which are due to unexpected legitimate use. Many similar approaches that aim at differentiating genuine requests from malicious, suffer from potential false alarms.

Most of the available solutions in the literature suggest topological modification or modifications at intermediate devices to mitigate DoS. [12] suggests a technique to detect DoS attacks by monitoring a wide IP address space for incoming unsolicited back-scatter packets. Such packets are a non-collocated victims response to several spoofed vulnerability and flooding attacks. The back-scatter packets source address is that of the victim, but the packets destination address is randomly spoofed. An attack that uses uniformly distributed address spoofing leads to a finite probability that any monitored address space will receive back-scatter packets. Such attacks can be detected using this analysis. [9] discusses statistical approaches to monitor activity variation to detect DoS attacks.

Most of the solutions concentrated on novel techniques to detect DoS attacks than trying to mitigate them. Some of the solutions to mitigate such attacks are very specific a certain kind of DoS attacks. SYN cookies were introduced by D.J.Bernstein to solve DoS attacks exploiting the vulnerability in Transmission Control Protocol [5]. Ingress filtering had been introduced to tackle DoS attacks done using spoofed IP addresses and later on it became an IETF standard [3].

The solutions presented in the above paragraph are specific to a particular vulnerability in the system or network or a protocol. But there is not generic solution that can work against the entire class of DoS attacks that are realized by brute force. We present a solution in this paper that provides support in operating system itself to mitigate such attacks and keeps the system more available and secure.

3 SELinux

Security-Enhanced Linux (SELinux) is an implementation of mandatory access control using Linux Security Modules (LSM) in the Linux kernel, based on the principle of least privilege [2], [4].

3.1 Reference Monitor

SELinux is based on the fundamental model for characterizing access control in operating systems called, reference monitor proposed in Anderson report [7], [11]. In a reference monitor, as shown in Figure 1 the operating system classifies resource into active and passive entities. Active resources such as processes are called subjects and passive resources such as files are called objects. The reference monitor mechanism controls access among these subjects and objects according to the specified security policy [15]. Access control decisions are based on security attributes associated with each subject and object. For example, in standard Linux, subjects have real and effective user identifiers, and objects have access permission modes that are used to determine whether a process may open a file.

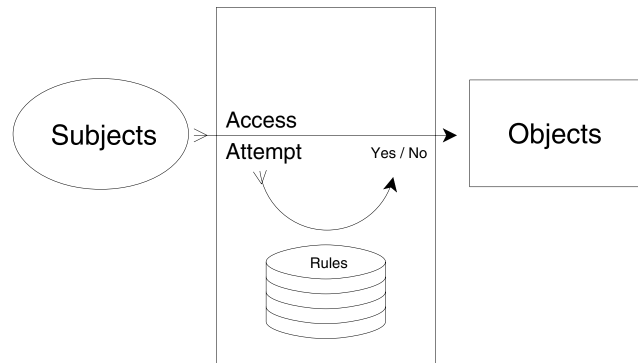

Figure 1. The Reference Monitor

Every operating system implements some form of reference monitor and applies access control on subjects and objects. In standard Linux, subjects are generally processes, and objects are the various system resource used for information sharing, storage, and communication. In general, the security policy rules enforced by the reference monitor are fixed and hard-coded, whereas the security attributes that these rules use for validation can be changed and assigned.

3.2 Discretionary Access Control

Discretionary Access Control(DAC) is a form of access control that usually allows authorized users to change

the access control attributes of objects, thereby specifying whether other users have access to the object [13]. Nearly all modern operating systems have some form of user-identity-based DAC. In Linux, the owner-group-world permission mode mechanism is prevalent and well known.

In such environment processes inherit the privileges of the users who created them and if a program is corrupt or malicious, it can misuse the privileges assigned to it. DAC assumes a benign environment, where all programs are trustworthy and without any flaws, and works well only in such environment.

3.3 Mandatory Access Control

To address the problem of malicious and flawed software, Mandatory Access Control (MAC) has been introduced as a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity [1].

SELinux enabled kernel enforces MAC policies that confine user programs and system servers to the minimum amount of privilege they require to do their jobs. This reduces or eliminates the ability of these programs and daemons to cause harm when compromised (via buffer overflows or misconfiguration). This confinement mechanism operates independently of the traditional Linux access control mechanisms.

3.4 Security Attributes

Typically such policy enforcement(MAC) is done in accordance with the policies specified by the administrator of the system In SELinux access control is based on some type of access control attribute associated with objects and subjects. This access control attribute is called a security context. All objects (files, inter-process communication channels, sockets, network hosts, and so on) and subjects (processes) have a single security context associated with them. A security context has three elements: user, role, and type identifiers. The usual format for specifying or displaying a security context is as follows:

user:role:type

The *type* identifier is the primary part of the security context that determines access. The *user* and *role* identifiers in a security context have little impact in the access control policy for type enforcement except for constraint enforcement which were discussed later in the paper.

3.5 SELinux Architecture

Figure 2 describes the architecture of SELinux which comprises of Security Server(SS) and Access Vector Cache(AVC) as main components and LSM hooks as the interface [14]. SS provides general interfaces for obtaining security policy decisions, enabling the rest of the module to remain independent of the specific security policies used. AVC provides caching of access decision computations obtained from the SS to minimize the performance overhead of the SELinux security mechanisms. It provides interfaces to the hook functions for efficiently checking permissions and it provides interfaces to the SS for managing the cache.

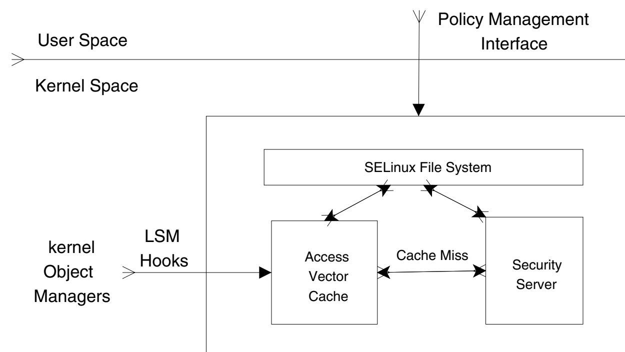

Figure 2. SELinux Architecture

3.6 Features of SELinux

- Type Enforcement (TE): The majority of SELinux policies are a set of statements and rules that collectively define the type enforcement policy. SELinux monitors access attempts by processes to every resource and an access succeeds only if there is at least one TE rule allowing that access.

- Role Based Access Control (RBAC): Roles act as a supporting feature to TE. RBAC in SELinux further constrains TE by defining the relationship between domain types and users to control Linux users privileges and access permissions.

- Multi-Level Security (MLS): To process information with different sensitivities differently MLS is also supported by SELinux.

4 Our Approach

SELinux policy framework mandates administrators to specify the behavior of a process in terms of, what kind of

access is required for a process to run. As SELinux enforces MAC, every running process requires a policy associated with it. These policies are typically static. Once they are specified, they are read during boot, and loaded into the system. When a process requests for any kernel object, that request is trapped by SELinux and verified against the rule base corresponding to the process and the object and access is granted or denied according to the policy.

We extended SELinux policy framework to address the problems discussed in the earlier section, that are related to DoS attacks realized through over-usage of resources. Using this extension one can specify not only whether access is allowed or not between two security contexts, but also the rate at which such access is allowed.

4.1 Why SELinux is Chosen ?

SELinux has already been accepted in the main-line Linux kernel due to several features like [4]

- Clean separation of decision making from enforcement
- Well-defined policy interfaces, flexible policy language
- Efficient caching of access decisions
- Control over process initialization and inheritance and program execution
- Control over all kernel objects

SELinux intercepts access to kernel objects in the system using LSM hooks. By referring to a policy database, it takes a decision, which will be enforced by kernel object managers. As already fine grained control over all the events in the kernel is with the SELinux, we found, adding few more parameters and extending the policy framework leads to a more available and secure systems.

4.2 Proposed Extension

Using SELinux policy language, one can specify the kind of access that is granted to a subject on an object. LSM hooks and SELinux enable the kernel to enforce such policy specification. But, there is no facility in SELinux policy language to specify the rate of access allowed between a subject and an object. The rate of a resource usage becomes important in mitigating some of the DoS attacks discussed in the earlier sections.

We propose to add one more parameter to the rule structure used by SELinux policy language, that specifies the rate of access allowed. This parameter is used to control the rate of a resource usage. Rate control is enforced along with TE. TE is achieved in SELinux using Access Vector(AV) rules. AV rules are the rules that specify accesses among different types of kernel objects. The syntax of an AV rule is as follows,

rule_name type_set type_set : class_set perm_set.

Between the type_sets that belong to some class, a permission can either be allowed or denied. we added rate control parameter, to the AV rules, that specifies the rate at which access should be allowed between kernel objects of gives types. The extended SELinux controls the rate of access between a subject and an object. If the rate crosses the allowed rate in the AV rules, access would be temporarily denied.

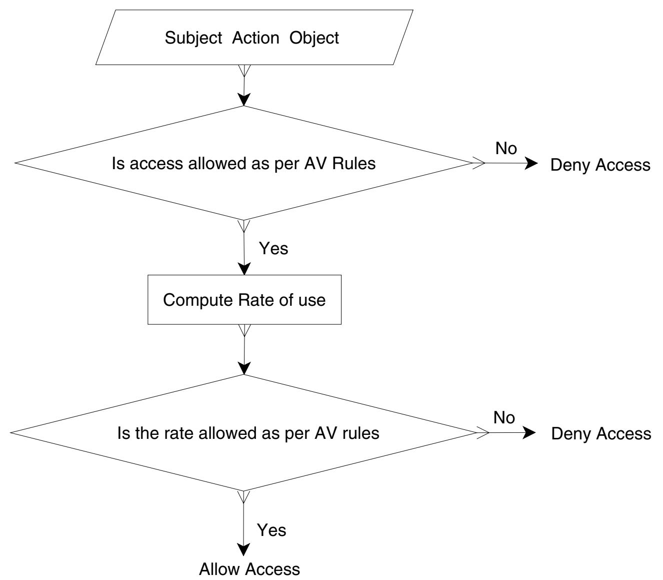

Figure 3. Flow of Control in the New Model

Figure 3 shows a flowchart of the control flow in system with the extension. When a subject tries to perform an action on a given object, LSM hooks divert the flow to SELinux, which refers to policy database to make a decision. If the policy doesn't allow the requested access, SELinux passes the message 'deny' back to kernel object managers. If the access is allowed as per the AV rules, extended SELinux computes the rate of such requests and compares the rate with the allowed rate specified in the AV rules. If the rate is with in the allowed limits, access is allowed or else denied by communicating the same to the kernel object managers.

4.3 Design Details

The variables that are part of the extension are loaded along with the SELinux policy file. In memory binary policy is maintained by SS. As every access request for kernel object is intercepted by SELinux, it would be time consuming to check with SS every time. To optimize the performance, AVC has been introduced by SELinux. AVC is a cache of the binary policy which consists of the access control information of the recently used subject/object domains. Whenever a request comes to SELinux about a particular domain, it exports the complete policy related to that domain into AVC. So, further requests about that domain can be served by AVC itself.

We exploit the existence of AVC in the same way as original SELinux does. The rate control information about the domains lies in the binary policy. Whenever a request for a domain is intercepted, rate control information would also get exported to AVC along with other policy information. Unlike original SELinux, the extension developed needs dynamic information for the enforcement of the extended policy. We maintain this dynamic information also in AVC itself along with the binary policy. In the AVC structure itself, we maintain the temporal information pertaining to the rate of usage of that particular access.

4.4 Implementation Details

In the AVC of SELinux we included the new parameter along in the records that contain subject, object and allowed access fields. The value of this parameter indicates the allowed rate of use of such access by the subject. To maintain the current usage of the access, we used moving window and smoothing average. According to the configured size of the time, a window is created and number of accesses made in that time window is considered as the current usage. Whenever the current usage exceeds the allowed rate, it is considered as anomalous and access is denied. To take care of genuine *flash events*, we used smoothing average where we give only partial weight to the exact usage in the current time window and partial weight to the recent time windows. If the rate exceeds the allowed rate after smoothing, that indicates growth in several windows and denying the access is more justified.

5 Experiment & Results

We have chosen to monitor *fork* and *accept* calls for the experiment. We aim to measure the turnaround time of our test-script (which comprises of a set of instructions) in the normal system and during the flood *fork* and *accept*. In a normal system, the turnaround time of any process should not be unpredictable when we run it multiple times. If the impact of an attack is contained in a system, turnaround time of concurrent processes will not deviate much from what it normally used to be. We have chosen to measure the same in our experiment.

We created a process that calls *fork* continuously and ran our test-script concurrently. we ran test-script multiple times to observe the variation in turnaround time. We did this experiment with and with-out the extension that we had built. As the process continued to *fork*, it consumed more and more system resources and system was spending more time with this and was unable to attend other processes. This is evident from the growing turnaround time of the test-script. Turnaround time of the test-script had grown with time when we ran the experiment with out our extension. When we controlled the rate at which this process can *fork*, we were able to reduce the rate of growth of turnaround time of the test-script. We repeated the experiment with *accept* call of a TCP server and got the same results.

We can conclude from these experiments that when the rate of resource usage is controlled in one domain, the system will be more fair towards the other processes in the system. Figure 4 and Figure 5 explain the results graphically.

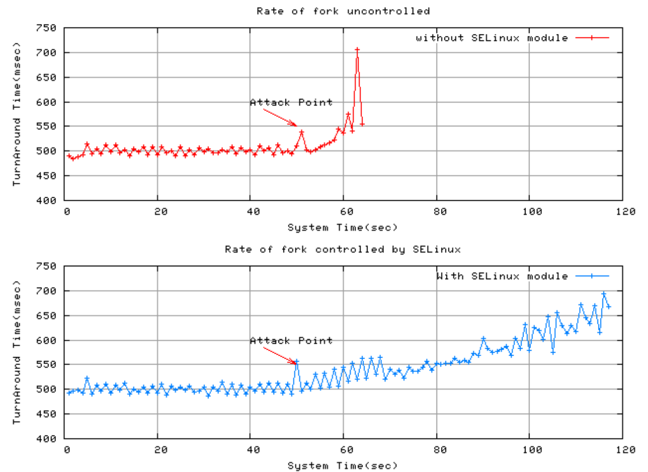

Figure 4. Variation in Execution Time in *fork* experiment

Figure 4 shows the effect of flood of calls to *fork* on the turnaround time of test-script. X-axis shows the system time and Y-axis shows the turnaround time of the test-script. First graph of Figure 4, which is of the system not guarded by our extension, shows the sharp increase in execution time and abrupt stop that indicates the system's 'ran out of resource' status. Second graph of Figure 4 is of the system guarded by the rate control extension. It shows the system's availability well after flood of calls to *fork*. As the

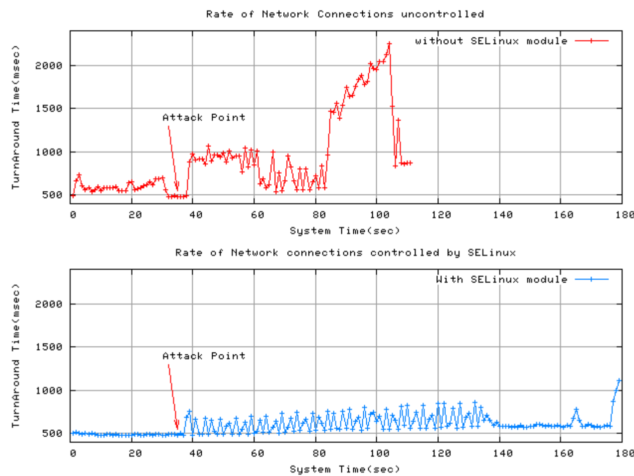

Figure 5. Variation in Execution Time in *accept* experiment

rate of calls to *fork* is kept under control, there is not much variation in the turnaround time of the test-script. There is a slow hike in the turnaround time, as we controlled only rate of calls to *fork* but not the total number of processes that are forked. But, it is evident that controlling the rate had stabilized the turnaround time of other processes, hence, led to a more available system.

In the second experiment we tried to control response to TCP connection requests by controlling *accept* calls. In Figure 5, it is evident that when there was no control over the rate of *accept*, turnaround time of the test-script has grown arbitrarily and when the rate is controlled, there was no significant effect on the turnaround time. When the experiment was done with out rate control extension, system became unusable after responding to several connection requests. When the rate of calls to *accept* is controlled, system stability is improved, which is evident from the graph in Figure 5.

6 Future Work

The extension with the support of the experimental results shows the advantage of controlling access among some domains to keep access times among other domains and in turn execution times of other processes predictable and consistent and lead to more available systems. This framework can be made more generic and robust, so that it can work with all kinds of calls that are being intercepted by SELinux. Currently we used thresholds which are created through human expertise. This process can be automated by machine learning where a module can learn the thresholds.

7 Conclusion

There are a lot of methods available in the literature that are aimed at detecting and protecting against DoS attacks. DoS attacks by their very nature are difficult to be mitigated. But intelligent early detection, and containing them will lead to more secure and available systems. The method we presented in this paper aims at controlling the resource usage by any entity in the system and restricts the spread of effect of an attack to other processes in the system, hence improves the availability of the system.

References

[1] Mandatory_access_control from wikipedia. In *http://en.wikipedia.org/wiki/Mandatory_access_control*.

[2] National secuirty agency - security enhanced linux. In *http://www.nsa.gov/selinux/info/faq.cfm*.

[3] Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing. In *http://www.ietf.org/rfc/rfc2267.txt*.

[4] Security-enhanced linux from wikipedia. In *http://en.wikipedia.org/wiki/SELinux*.

[5] Syn cookies. In *http://en.wikipedia.org/wiki/SYN_cookie*.

[6] D. Adkins, K. Lakshminarayanan, A. Perrig, and I. Stoica. Taming ip packet flooding attacks. In *In HotNets-II. ACM Press*, 2003.

[7] J. P. Anderson. Computer security technology planning study. Technical report, Oct. 1972.

[8] G. Carl, G. Kesidis, R. R, and S. Rai. Denial-of-service attack-detection techniques. In *IEEE Computer Society*, February 2006.

[9] L. Feinstein. Statistical approaches to ddos attack detection and response. In *Proc. DARPA Information Survivability Conf. and Exposition*, pages 300–314, 2003.

[10] S. M, M. Greenwald, C. Gunter, S. Khanna, and S. Venkatesh. Mitigating dos attack through selective bin verification. In *Secure Network Protocols, (NPSec). 1st IEEE ICNP Workshop on*, pages 7–12, 2005.

[11] F. Mayer, K. MacMillan, and D. Caplan. *SELinux by Example: Using Security Enhanced Linux*. Prentice Hall.

[12] D. Moore, G. Voelker, and S. Savage. Inferring internet denial-of-service activity. In *Proc. Usenix Security Symp., Usenix Assoc.*, 2001.

[13] U. S. D. of Defense. Trusted computer system evaluation criteria. *DoD Standard 5200.28-STD*, December 1985.

[14] S. Smalley and C. Vance. Implementing selinux as a linux security module. Technical report.

[15] R. Spencer, S. Smalley, P. Losocco, M. Hibler, D. Andersen, and J. Lepreau. The flask security architecture: System support for diverse security policies. Technical Report UUCS-98-014, 1998.

[16] B. Waters, A. Juels, J. A. Halderman, and E. W. Felten. New client puzzle outsourcing techniques for dos resistance. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 246–256, New York, NY, USA, 2004. ACM Press.