A Learning-Based Approach for SELinux Policy Optimization with Type Mining

Said Marouf
University of North Carolina at Charlotte
smarouf@uncc.edu

Doan Minh Phuong
University of Engineering and Technology, VNU at Hanoi
phuongdm@vnu.edu.vn

Mohamed Shehab
University of North Carolina at Charlotte
mshehab@uncc.edu

1. INTRODUCTION

One of the major steps towards enhancing the security of the Linux operating system was the introduction of Security Enhanced Linux (SELinux) [1], developed by the U.S. National Security Agency. SELinux is a kernel Linux Security Module (LSM) that adds Mandatory Access Control (MAC) to a regular Linux system with Discretionary Access Control (DAC) [2]. SELinux supports Type Enforcement (TE), Role Based Access Control (RBAC), and Multi-Level Security Levels (MLS).

In this paper we focus on SELinux type enforcement policies, in SELinux policies involve complex administration and management [3]. The complexity comes from the high granularity provided by SELinux, such high granularity requires administrators to keep track of all attributes that comprise a SELinux policy, e.g. types, domains, roles, labels, etc. We believe that reducing the number of attributes thus simplifying SELinux policies, is a great step towards making administrators' lives easier. Within a SELinux policy, administrators define pairs of Type-Type mappings, these pairs declare what accesses are allowed between the two types (Note that a type on a process is called a *Domain*). We focus on reducing SELinux application access to minimum set of types used by the application, hence reducing the number of Domain-Type associations within a policy. We believe this will lead to: 1)Simplifying SELinux policies, hence help administrators keep track of all factors comprising a SELinux policy. 2)Increasing the security of a SELinux policy by removing unnecessary Domain-Type mappings within a policy after analyzing an application's behavior, hence following the least privileged rule.
We propose a learning-based approach for monitoring an application's behavior through system calls. Analyzing system call logs allows us to improve upon an application's policy within SELinux by investigating the actual types accessed by the application's domain vs. types originally requested. Our approach is inspired by the Role Mining Problem [4], i.e. finding the optimal User-Role and Role-Permissions set, where in our case the goal is to find the optimal Domain-Type and Type-Resource set.

2. SELINUX POLICIES

There has been much research has been done in investigating SELinux policy analysis and verification [5, 6, 7, 8]. Related to automatically generating SELinux policies, the MITRE Corporation [9] proposed an approach that guides administrators in building new policies. Another research area is related to simplifying SELinux policies, Yokoyama et al. [10] suggested a framework that splits policies according to different phases of an application's execution. In their work they showed that the Apache policy can be split into three parts each mapping to a different phase.Another interesting area is related to verifying SELinux policies and their consistency, Ahn et al. [11] introduced an approach and tool that verifies SELinux policies and helps in detecting any potentially unwanted information flow.

To our knowledge there is no learning-based approach to simplifying SELinux policies. We believe that SELinux policies can be simplified considerably based on learning the behaviors of applications within SELinux. In addition we can prevent such applications from accessing resources/processes that they potentially will not use or need. Deciding on what resources an application needs is based on our learning approach. The learning approach is based on monitoring the behaviors of a particular application through the set system calls it invokes. Because this is a learning-based approach, a policy can be modified various times according to the changes in its behavior. We also believe the administrator should have the final decision on the changes, hence we consider recommending policy changes rather than actually making them.

Taking a close look at Domain-Type mappings within a SELinux policy, we notice a similarity with the Role Mining Problem (RMP), i.e. RMP tries to optimize the User-Role and Role-Permissions mappings, in a similar fashion one can optimize the Domain-Type and Type-Resource mappings within an SELinux policy. This optimization is applied to the Domain-Type mappings deemed necessary using our learning-based approach.

Another issue within SELinux is that default policies especially those with unconfined types (unconfined_t) are allowed access to more resources than they will potentially ever need. In a case like this, it is useful to learn an application's be-

havior patterns, then deduce the set of Domain-Type mappings necessary for it to operate properly, and use these mappings within a newly configured SELinux policy. This allows for a more secure system, even in a default Targeted mode SELinux configuration.

So, our goal behind this research is to 1)Simplify SELinux by reducing Domain-Type mappings within an application's policy, 2)Adapt policies according to their actual access control needs based on a learning approach, and 3)Optimizing the Domain-Type mappings resulting from the learning process, based on the RMP which we call the Type Mining Problem.

The following is an example of a SELinux allow rule. The rule states that *domain_t* has access to files of type *type_t*, with permission $read\_file\_perms$. Here we say the domain *domain_t* is mapped to the type *type_t*.

allow domain_t type_t:file {*read_file_perms*}

2.1 Default Policy Behavior

SELinux can operate in two modes, Strict, and Targeted. In Strict mode, applications are denied access by default. Whereas in Targeted mode, by default applications are given the unconfined_t domain/type, which has full access to all SELinux types. In Targeted mode, restrictions are enforced only in cases where developers provide a policy for their applications, or if an administrator explicitly writes and enforces a policy on a certain application. Figure 1 and 2 illustrate the process of assigning the proper type/domain at installation and run time respectively.
Our approach focuses on enhancing the default behavior of SELinux Targeted mode by inferring new policies. These policies are based on learning and monitoring the behavior of an application during run time, analyzing accesses made, and creating an enhanced version of the default policies.

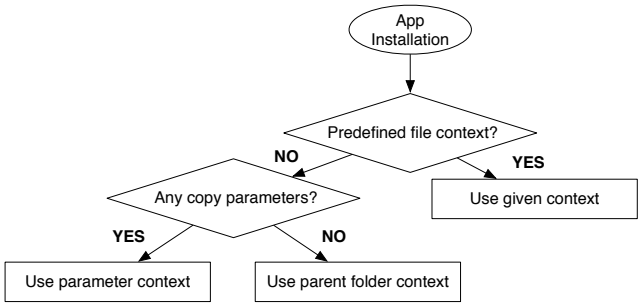

Figure 1: Type Enforcement (Installation Time)

3. OUR PROPOSED APPROACH

We define the domain to type mapping as:

DEFINITION 1. *(Domain-Type Mapping) With in a policy, when a domain $d_n \in D$ is allowed access to a resource $o_k \in O$ of type $t_i \in T$, there exists a mapping $m(d_n, t_i)$ between d_n and t_i, where D is the set of Domains in SELinux, O is the set of resources, and T is the set of Types.*

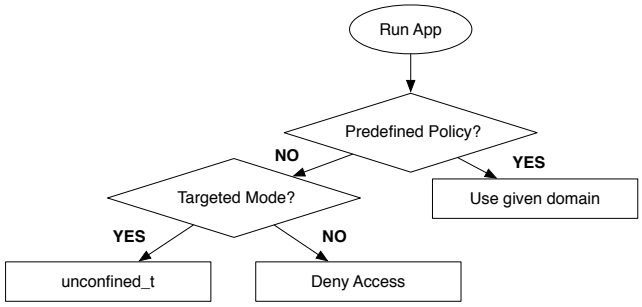

Figure 2: Domain Enforcement (Run Time)

For the sake of simplicity let us consider the set of mappings M_{d_n} for a single domain d_n. Also consider $T_{d_n} \subseteq T$ which is the set of types that domain d_n is originally mapped to, i.e. before applying any optimization. Figure 3 shows a set of different domain and type mappings before optimization. Notice the mapping $m(d_n, t_m)$ between d_n and t_m, this mapping is a potential candidate for optimization as we will see in subsection 3.1.

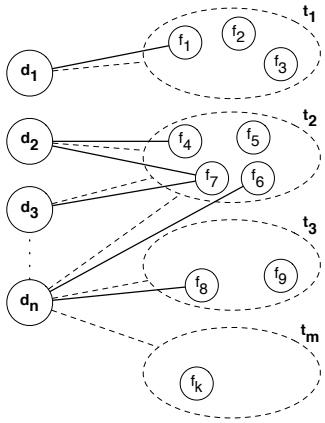

Figure 3: Domain-Type & Type-File mappings

3.1 Learning-Based Approach

Consider a domain d_n, which initially has a set of mappings M_{d_n}. These mappings are specified by an application's (i.e. its process) default SELinux policy. The matter of fact is that an application at certain times will not need access to all the resources allowed within its policy, hence what accesses it needs are different from what it actually utilizes. Figure 3 shows d_n is mapped to t_2, t_3, and t_n (via the dashed lines). Using a learning approach, we can filter out the unnecessary allowed accesses within an application's policy. This can be achieved by monitoring an application's behavior through system calls it makes while running. We use the modified *strace* command provided by Polgen [9] which allows us to record the security context (user:role:type) of resources accessed via system calls, hence we know the type associated with a certain resource. Figure 3 shows that the application with domain d_n only accessed files f_6, & f_8 (via the hard lines), which means the mappings $m(d_n, t_3)$, and $m(d_n, t_2)$ are the necessary mappings and there is no need for $m(d_n, t_m)$. At this point one could remove the mapping $m(d_n, t_m)$ from the application's policy, i.e. removing the

rules associated to this mapping.
Figure 4 shows a potential new mapping set, notice for domain d_n the type t_m is removed, hence d_n will have a new domain-type mapping set M'_{d_n}. We also propose to generate new types from existing ones, these new types will be sub-types that are associated to only the resources accessed by a particular domain. The type t_2 in Figure 3 can be sub-typed into t'_2 where t'_2 does not include the file f_5.

DEFINITION 2. *(Sub-Type) A type t_s is a sub-type of t_p if $O_{t_s} \subseteq O_{t_p}$, where O_{t_s} is the set of resources with type t_s, and O_{t_p} is the set of resources with type t_p.*

We believe the Domain-Type mappings will change at different times, hence we suggest an adaptable approach which monitors applications frequently and accordingly their policies could be adjusted.

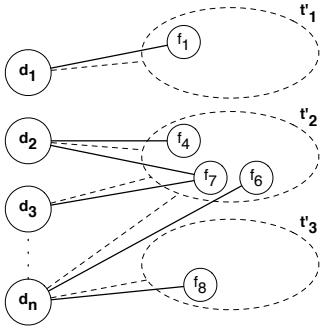

Figure 4: Post Domain-Type & Type-File mappings

3.2 The Type Mining Problem

After applying the learning approach and finding the necessary domain-type mappings, we end up with a set of new sub-types T'. To optimize this set and its mappings to domains and resources, we refer to the Role Mining Problem (RMP) [4]. The RMP tries to optimize the User-Role and Role-Permissions mappings, in a similar fashion one can optimize the Domain-Type and Type-Resource mappings within an SELinux policy. That is, we can optimize the Domain-SubType mappings between the domains D and the resulting sub-types T'. For example, in Figure 4 type t'_3 could probably be merged with type t'_2, given the resulting type $t_{2,3}$ does not invalidate the original accesses intended for both t'_2 and t'_3. As in RMP, an optimal Role set must be found that results in a maximum approximation to the original User-Permissions intended. Similarly in our approach, we must optimize the type set T' such that we maximize the approximation of accesses given to domains on certain types. We call this the Type Mining Problem.

DEFINITION 3. *(Type Mining Problem) given a $u \times k$ binary matrix A representing Domain to Resources mappings, the Type Mining Problem is the process of finding two matrices B and C, where B is a $u \times m$ matrix that represents the Domain to Type mappings, and C is a $m \times k$ matrix that represents the Type to Resources mappings, such that $A = B \times C$ and m is minimal.*

We also consider a more complex scenario where we look into possible Type to Type mappings, e.g. domains that access other domains (rather than types on resources) that also access certain types. This will add an extra level of mappings to explore.

4. POLICY INFERENCE

Optimizing default SELinux policies relies on the ability to infer policy rules from the Strace logs we collect by monitoring an application's behavior. The Strace logs are a collection of single line traces, each trace representing a single system call made by an application. A typical log trace contains a system call's name & parameters (e.g. object accessed), the security context of the application calling the system call (specifies the domain), and the security context of object accessed (specifies the type being accessed by the domain), see Figure 5:

```
Structure:
sys_call([object ≪security_context≫], ...)  =
return_value ≪domain_security_context≫

Example:
getcwd("/root/Desktop" ≪root:object_r:user_home_t:s0≫, 4096) =
14 ≪root:system_r:unconfined_t:s0-s0:c0.c1023≫
```

Figure 5: Log Trace Structure & Example

Inferring an optimized version of a default policy happens through two steps:

1. Rule Filtering: Filter out unnecessary *allow* rules that are not used by an application. Filtering is based on:
   - The domain-type accesses extracted from the log (e.g. <*unconfined_t - user_home_t*> in Figure 5). Hence, we can compare these domain-type pairs to the set of domain-type pairs for unconfined_t in the original SELinux reference policy. We can remove *allow* rules that do not have a corresponding domain-type pair.
   - The domain-type actions. These are the actions made by system calls (e.g. read, write, etc.). We use these actions to determine what *object-class* permissions are necessary within an *allow* rule. *object-classes* represent certain objects, e.g. *"file"* is an *object-class* for files. For example, an application might only need to *open* and *read* from the *object-class "file"* for type t_i, in this case we can remove any *write* permissions given within *allow* rules for the type t_i and *object-class "file"*.
2. Type Optimization: Using the logs collected for a certain application, we follow the process explained in subsection 3.2. This relies on finding the actual objects accessed via system calls. These objects can be extracted from the system call attributes, e.g. an *open* system call takes a *file path* as an attribute, the file at this path is considered the accessed object.

5. CONCLUSION

Our proposed approach is based on two main techniques. The first, learning an application's behavior through system calls and discovering the necessary domain-type mappings,

based on the accessed resources. This leads to a set of new sub-types. Second, optimizing the resulting subtype sets. We borrow from the Role Mining Problem, and map it to our problem, which we call the Type Mining Problem. The approach will potentially simplify policy administration, and provide more secure policy configurations that adapt to an application's environment.

6. REFERENCES

[1] Security Enhanced Linux, "http://www.nsa.gov/research/selinux, (03/10/2010)."

[2] SELinux in Ubuntu, "https://wiki.ubuntu.com/selinux, (03/12/2010)."

[3] D. Zhang, K. Ramamohanarao, and T. Ebringer, "Role engineering using graph optimisation," in *SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies*, (New York, NY, USA), pp. 139–144, ACM, 2007.

[4] J. Vaidya, V. Atluri, and Q. Guo, "The role mining problem: finding a minimal descriptive set of roles," in *SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies*, (New York, NY, USA), pp. 175–184, ACM, 2007.

[5] G. Zhai, W. Ma, M. Tian, N. Yang, C. Liu, and H. Yang, "Design and implementation of a tool for analyzing selinux secure policy," in *ICIS '09: Proceedings of the 2nd International Conference on Interaction Sciences*, (New York, NY, USA), pp. 446–451, ACM, 2009.

[6] B. Hicks, S. Rueda, L. St.Clair, T. Jaeger, and P. McDaniel, "A logical specification and analysis for selinux mls policy," in *SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies*, (New York, NY, USA), pp. 91–100, ACM, 2007.

[7] B. Sarna-Starosta and S. D. Stoller, "Policy analysis for security-enhanced linux," in *Proceedings of the 2004 Workshop on Issues in the Theory of Security (WITS)*, pp. 1–12, April 2004. Available at http://www.cs.sunysb.edu/~stoller/WITS2004.html.

[8] T. Jaeger, R. Sailer, and X. Zhang, "Analyzing integrity protection in the selinux example policy," in *SSYM'03: Proceedings of the 12th conference on USENIX Security Symposium*, (Berkeley, CA, USA), pp. 5–5, USENIX Association, 2003.

[9] The MITRE Corporation, "Polgen: Guided automated policy development. url http://www.mitre.org/tech/selinux, (03/05/2010)."

[10] T. Yokoyama, M. Hanaoka, M. Shimamura, and K. Kono, "Simplifying security policy descriptions for internet servers in secure operating systems," in *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, (New York, NY, USA), pp. 326–333, ACM, 2009.

[11] G.-J. Ahn, W. Xu, and X. Zhang, "Systematic policy analysis for high-assurance services in selinux," in *POLICY '08: Proceedings of the 2008 IEEE Workshop on Policies for Distributed Systems and Networks*, (Washington, DC, USA), pp. 3–10, IEEE Computer Society, 2008.