

Attack-based Domain Transition Analysis

Susan Hinrichs

University of Illinois at Urbana-Champaign, shinrich@uiuc.edu

Prasad Naldurg

Microsoft Research, prasadn@microsoft.com

Abstract

SE Linux type enforcement policies are widely understood to be large and difficult for humans to understand. Several groups have created tool sets to aid SE Linux policy writers and maintainers in the task of working with SE Linux policies. We propose an attack-based policy model to look at the transitive domain transitions allowed in the policy. We have augmented Apol, a graph-based policy analysis tool, to implement global transitive domain translation analysis and more focused reduced transitive domain transition graphs between sets of suspect domains and sensitive domains.

1. Introduction

In SE Linux the main mandatory access control mechanism is implemented through Type Enforcement[5]. The primary entities involved in the access control are processes (representing users or subjects) labeled by *types* which are normally called *domains* and files which are also labeled by *types*. A set of access vector rules define what permissions processes labeled by particular domains have on files and other operating system entities labeled by types. With an appropriate set of rules, the type enforcement mechanism can effectively compartmentalize sets of subjects from one another when running on the same system.

In the default case, a process is labeled with a particular domain, and all new processes created from that process (by executing other programs) are labeled by the same domain. Thus, if a subject is subverted (through malware, stolen password, or idle console), the type enforcement rules constrain the set of files the attacker can read and write.

However, in any realistic system, there will be requirements for one process to create another process with different domain labeling. This is obviously needed when a system is booted so the initial process can create all the normal daemons in the appropriate domains. Similarly, the login process will need to create user domains appropriate for each logged on user. In other cases, programs like “passwd” need to run at a different domain so it can access the privileged information contained in the password file.

In SE Linux, this change in domain labeling is called a *Domain Transition*, and in general, the domain transi-

tion is a good thing which enables controlled transitions between controlled type compartments. The Information System Security Officer (ISSO) should understand which program calls which other programs and thus understand which domain transitions are acceptable. However, a couple problems could arise with respect to domain transitions:

- In a complex policy, there could be misunderstandings that results in program call chains ending in unexpected domain transitions. For a normally running program such a transition might be benign, but it reveals a least privilege error that could be exploited.
- An attack could subvert a program running in a particular domain and leverage the domain transitions in the system to expand the set of files the attacker has access to by invoking other programs in an unexpected manner and using the domain transition mechanisms to continue the attack through other domains.

At least three access vector rules are required to enable the domain transition.

1. The source domain has execute access to the executable file type.
2. The target domain has endpoint access to the executable file type.
3. The source domain has transition access to the target domain.

The `type_transition` rule specifies the default domain assigned to the new process.

```

type_transition <source domain>
  <file type>: process
  <target domain>

```

This rule indicates that a process running at <source domain> can create a process labeled with <target domain> when it executes a file labeled with <file type>. In addition to the `type_transition` rule, an application can also explicitly identify the domain for a new process by using the `setexeccon` system call.

Figure 1 shows a sequence of domain transitions. The solid arrows show the allowed transition from one domain to another. Each domain has a set of types that it can read and another set of types it can write. While the type sets probably overlap, it is likely that each new domain transition the set of accessible types will grow.

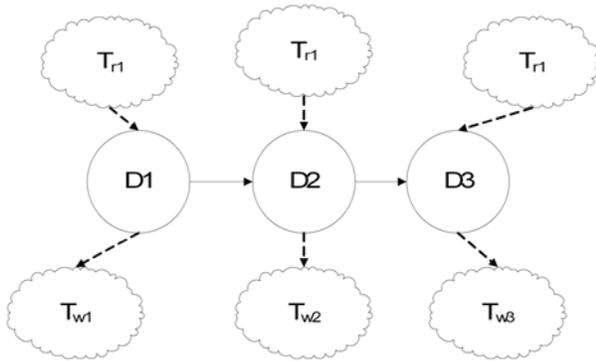


Figure 1: Example domain transition chain. Each domain is shown with the set of types it can read T_{rx} and the set of types it can write T_{wx} .

In this paper, we describe a global transitive domain transition analysis (DTA) algorithm that builds on the DTA algorithm implemented in Apol[1]. We describe an attack-based approach for analyzing the resilience of a SE Linux policy. Using the attack-based approach and the global DTA algorithm, we identify some graph techniques to aid the ISSO in strengthening his policy and making reasonable trade-offs between security and system functionality. We end our paper by sharing some results of this approach on the standard strict SE Linux policy and outlining future directions.

2. Related Work

One can broadly divide work on system security analysis into two camps. One camp represented by the work of Sheyner, et. al. [3] uses logic to model the system, in

this case the network structure and host vulnerabilities. Then they pose questions about the system (potential attacks) and use model checking technology to determine if the model satisfies the question. This approach is very rigorous and is well suited to situations where there is very detailed information about the system and potential vulnerabilities, and the end-user can pose clear queries.

In the SE Linux domain, the policy analysis tool Slat[2] uses the model checking approach. With Slat the ISSO can determine whether assertions about his policy hold true or not. If the assertion does not hold up, Slat will show a counter example from the policy.

The other broad camp of system security analysis tools use graphs more directly. Archipelago[4] is an example of a graph-based analysis tool used to analyze the security of a network of computers. Graph algorithms may be used with these tools, and the graph based tools tend to also provide the end-user with visual insight into the security structure of the system by displaying different views of the graph. In SE Linux, Apol[1] takes the graph approach. Apol presents the end user with a graphical view of his policy access vector rules. It calculates transitive information flows based on the access vector rules. Apol also performs an on demand DTA that calculates the immediate domain transitions to or from a specified domain. Given a domain, it will search the entire policy to determine which domains the selected domain can transition.

We follow the graph-based approach and attempt to give the end-user intuition from the graph structure in addition to analysis. However, the domain transition information could also be incorporated into a model-checking style of analysis.

3. Global Domain Transition Analysis

One could iteratively apply the Apol DTA to calculate the global DTA, but such a technique will perform multiple global policy traversals for each domain. We used this approach initially, and it took tens of minutes to compute the DTA for all domains in the system starting with the strict policy.

If the goal is to calculate the DTA for all domains in the system (or many domains in the system), the obvious thing to do is take an initial pass on the policy to gather rules into three groups. We used STL multimaps and sets to quickly and efficiently implement these containers.

```

For each type_transition rule R1
  For each access rule R2 in ExecuteGroup keyed by R1.src
    For each rule R3 in EntryGroup keyed by R1.target
      If R2.type == R3.type
        For each rule R4 in TransitionGroup keyed by R1.src
          If R4.src == R1.src and R4.target == R1.target
            AddEdge(R1.src, R1.target, R1.type);
            Break to type_transition loop

```

Figure 2: Psuedo code for calculating transitive domain transition.

1. ExecuteGroup: Rules that enable a domain to have execute rights to a type. This group is keyed by the domain.
2. EntryGroup: Rules that enable a domain to have entry point rights to a type. This group is keyed by the domain.
3. TransitionGroup: Rules that enable one source domain to transition to a target domain. This group is keyed by the source domain.

Figure 2 shows how the edges are calculated. The current algorithm assumes that only transitions specified by a type_transition rule will be allowed. As noted earlier this is overly restricted and would not consider transitions directly specified by applications using the setexeccon call.

With this algorithm, calculating the DTA graph for all domains in the strict policy takes under a minute. In the version of the strict policy we worked with, there were 806 domain transitions specified in the policy. Our tool emitted the global domain transition graph in XGML form, and we used yEd from yWorks (<http://www.yworks.com>) to display the resulting graph. With the “classic hierarchy” layout, we were able to see the basic structure of the global domain transition graph as a rather shallow tree (see Figure 3). The global domain transition graph can also provide other interesting statistics about the type enforcement policy such as identifying the *source* domains, domains that have only outgoing transition edges, and the *sink* domains, domains that have only incoming transition edges.

4. Attack-based Analysis

The SE Linux policies tend to become quite large for installations that attempt to implement fine-grained compartmentalization between users and applications. We explore an attack based technique by examining susceptible programs we think are likely to be sub-

verted either through malware or through lack of end-user awareness (e.g., leaving himself logged in at public terminals), and by identifying programs that have access to very sensitive data. Ideally, it should be impossible for susceptible programs to access the same data available to the sensitive programs except in very constrained circumstances.

By examining the domains associated with the set of susceptible programs (the suspect domains) and the domains associated with the set of sensitive programs (the sensitive domains), we use DTA to determine what paths can be used for a suspect domain to transition to a sensitive domain. We also use graph analysis to assist the user in evaluating transitions to remove to increase separation between the two sets of domains.

In the base case, if there are domains that are members of both the suspect set and the sensitive set, the type enforcement will allow data sharing between those members of the two sets. Clearly the solution in that case is to create a new domain to separate the sensitive programs using the shared domain from the suspect programs using the same domain.

In the less trivial case, the global domain transition graph G can be reduced to only include the domain transition paths that start from domains in the suspect set and end with domains in the sensitive set. This subgraph can be calculated as follows:

1. Compute the set of nodes reachable from nodes in P . Start with P_0 , the set of nodes in P .

$$P_{i+1} = P_i \cup \{n \mid \exists x \ni P_i \wedge (x, n) \ni E(G)\}.$$

Therefore, P_* is the transitive closure of all nodes reachable from P .

2. Similarly, we calculate the transitive closure T_* of all nodes that can reach nodes in T . We calculate this much the same way we calculated P_* except we consider edges leading into T_i when constructing

$$T_{i+1} = T_i \cup \{n \mid \exists x \ni T_i \wedge (n, x) \ni E(G)\}$$

3. We compute the set of nodes in the reduced subgraph as $P_* \cap T_*$. The edge is included in the subgraph only if both endpoints are in the subgraph.

By performing a breadth first traversal and not revisiting nodes already in the closure set, the calculation of steps 1 and 2 can be performed in time linear to the number of nodes and edges of the global graph. Figure 4 shows a reduced domain transition graph concentrating on a suspect set containing `pppd_t`.

If the reduced graph is empty, the ISSO knows that there is no way for control to flow directly from a suspect domain to a sensitive domain. Otherwise, the ISSO can use the reduced domain transition graph to identify a set of minimum or near minimum edge-cut sets, which correspond to the transitions that need to be removed to achieve true domain separation. One set may represent transitions that are easier to mitigate than the others. One might also give preference to one set over another by analyzing the length of the paths being cut. A long domain transition path may be rationalized as being too long to be practicably useable.

Means to mitigate a transition include

- The transition is not really needed and can be removed completely.
- Additional program analysis can be performed to ensure that the target transition is not open to abuse.
- Insertion of a proxy or a transformation procedure that acts as a high assurance gateway and is executed by any program transitioning between the adjacent domains on the cut edge.
- Additional tools or scripts can be configured to pay special attention to the audit messages emitted by these suspect transitions. Ideally, specific sequences of audit messages can be identified to separate legal domain transitions from illegal.
- The transition can be guarded by a Boolean and turned off when the ISSO determines the system is under attack or otherwise, running in a less secure environment.
- The target domain of the translation could be cloned. The cloned target domain accesses a subset of the data needed by the domain transition chain, and the subset data is deemed to be a satisfactory risk by the ISSO.

Instead of looking at minimum edge cut sets, we could also examine minimum node cut sets. Cutting a node would involve separating a domain, i.e., creating two domains and creating two separate sets of access rules for each new domain. It seems unlikely that we could completely divide the types accessed by a single domain into two disjoint sets accessed by each new domain.

5. Example DTA Scenario

Figure 3 shows the transitive closure over all domain transitions in the strict policy. With 297 nodes and 863 edges, the graph only reveals a gross structure to the end-user. It shows that the domain transitions form a relatively shallow tree.

Figure 4 shows a reduced domain transition graph that concentrates on the transitions rooted from the `pppd_t` domain. You can see that the `pppd_t` domain can transition into several of the mail oriented domains. This reduced graph enables the ISSO to concentrate of a sub portion of the domain transition graph. For example, if the ISSO learned of an exploit on the `ppp` daemon, he might want to examine the domain transitions from the `pppd_t` domain to identify potential attack paths. In that case, the ISSO might determine that the functionality of allowing `pppd_t` to transition to `system_mail_t` is not worth the risk. Alternatively, he might create add Boolean guards to the transitions from `pppd_t` to `system_mail_t` and `postfix_master_t` so the transitions can be disabled during times when system attack seem more likely.

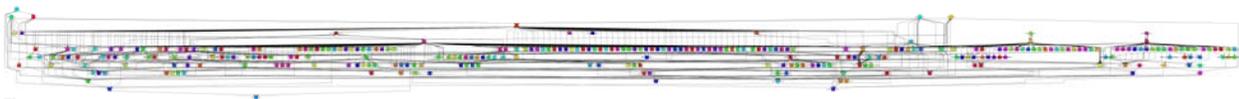


Figure 3: Global domain transition graph for the strict policy.

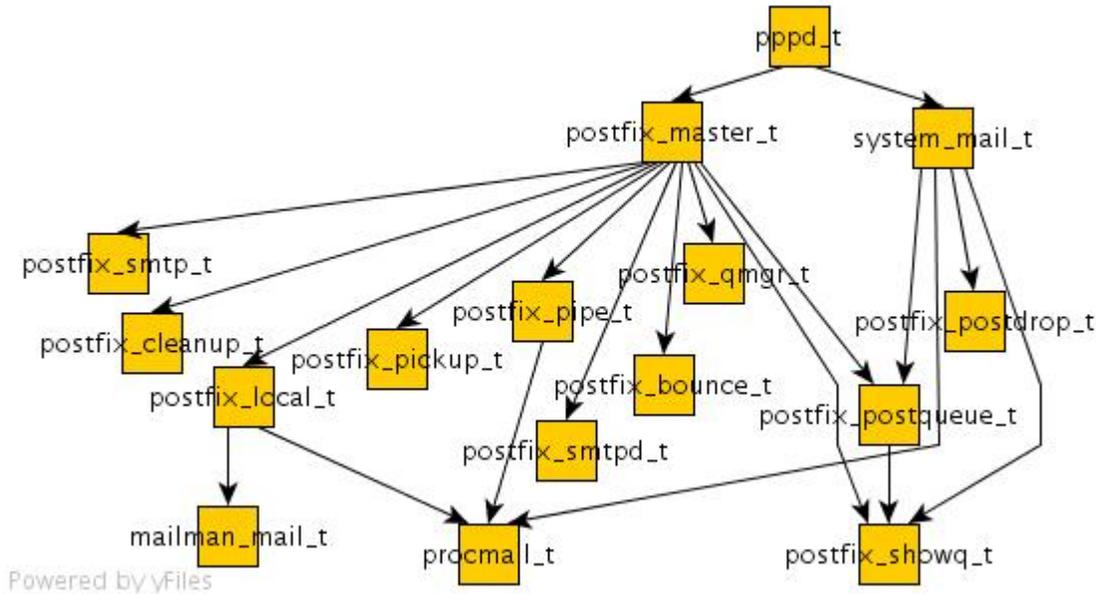


Figure 4: Reduced DTA graph from the pppd_t domain.

6. Conclusions and Future Work

We have only just started to break the surface in using the domain transition graphs to examine how attacks might propagate in a system. We need to examine real policies with this approach to get a feel for which mitigation approaches make sense in the real world.

The domain transition analysis can also be integrated with information flow graphs to get a better gauge of how information confidentiality leaks and integrity problems could flow in the presence of an attacker. Suppose the access rules enable an information chain of domains and types from domain 1 to domain N. If there are also domain transitions along the chain, a single attacker could in theory push the information along the path.

The reduced domain transition graphs show that if you can ask the right question, displaying graphs to the end user can help him understand his system. We used an existing graph editor. Integrating such a graph layout tool into other SE Linux policy tools could be beneficial.

Understanding domains transitions can aid the ISSO in understanding how his SE Linux system operates. By taking an attack-based approach, the domain transition graph can identify vectors of attack and help the ISSO modify his policy or closely examine critical domains.

References

1. F. Mayer, "Tools and Techniques for Analyzing Type Enforcement Policies in Security Enhanced Linux", in *Proceedings 2003 Annual Computer Security Applications Conference*, 2003.
2. J. Guttman, A. Herzog, J. Ramsdell, C. Skorupka, "Verifying information flow goals in Security-Enhanced Linux", *Journal of Computer Security*, vol 13, 2005, pp. 115-134.
3. O. Sheyner, J. Haines, S. Jha, R. Lippmann, J. Wing, "Automated Generation and Analysis of Attack Graphs", in *Proceedings of 2002 IEEE Symposium on Security and Privacy*, 2002.
4. T. Strang, et. al., "Archipelago: A Network Security Analysis Tool", in *Proceedings of LISA XVII*, San Diego, CA, 2003.
5. Peter Loscocco and Stephen Smalley. "Integrating Flexible Support for Security Policies into the Linux Operating System". In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference (FREENIX '01)*, June 2001.