SPTrack: Visual Analysis of Information Flows within SELinux Policies and Attack Logs

Patrice Clemente[1], Bangaly Kaba[1], Jonathan Rouzaud-Cornabas[2], Marc Alexandre[1], and Guillaume Aujay[1]

[1] ENSI de Bourges – LIFO
88 Bd Lahitolle, 18020 Bourges, France
`Patrice.Clemente@ensi-bourges.fr`
[2] LIP – INRIA – ENS Lyon
9 rue du Vercors, 69007 Lyon, France

Abstract. Analyzing and administrating system security policies is difficult as policies become larger and more complex every day. The paper present work toward analyzing security policies and sessions in terms of security properties. Our intuition was that combining both visualization tools that could benefit from the expert's eyes, and software analysis abilities, should lead to a new interesting way to study and manage security policies as well as users' sessions. Rather than trying to mine large and complex policies to find possible flaws within, work may concentrate on which potential flaws are really exploited by attackers.

Actually, the paper presents some methods and tools to visualize and manipulate large SELinux policies, with algorithms allowing to search for paths, such as information flows within policies.

The paper also introduces a complementary original approach to analyze and visualize real attack logs as session graphs or information flow graphs, or even aggregated multiple-sessions graphs.

Our wishes is that in the future, when those tools will be mature enough, security administrator can then confront the statical security view given by the security policy analysis and the dynamical and real-world view given by the parts of attacks that most often occurred.

1 Introduction

In the field of computer security, system administrators become more and more confronted with large and complex security policies without tools to analyze those policies. Even on security hardened systems enforcing Mandatory Access Control mechanisms like GrSecurity or SELinux, very precise and accurate policies always allow numerous security breaches. In [1], the authors show that on a finely defined SELinux security policy, there remain over than 1 million of potential security breaches, using indirect sequences of interactions on the system. For example, a user can connect on a system and can gain privileges exploiting a flaw in a legitimate program; he can then access to confidential root data. Thus, to analyze such potential flaws, one should need to see and analyze policies, in the most natural way. As all flows rely on sequences of multiple interactions, the best way is to visualize them as visual graphs.

But visually analyzing policy graphs with thousands of nodes and edges is not that useful and easy. Users should be able to focus their attention of specific parts, properties or sub-paths in the policies. More than that, one can need to visualize policies in terms of security properties violations.

Thus, the paper focuses on an abstract view of confidentiality and integrity security properties: information flows. Our approach provides a method and a tool to track information flows within security policies.

Although the paper presents preliminary work, the approach should fulfill multiple objectives in the future: allowing to detect security breaches in the security policy and modify it in consequence.

Sometimes, analyzing policies is inefficient or lacks experimental or practical feedback. That is why in a second part, the paper faces the problem of system session logs visualization. It provide methods and tools to efficiently aggregate numerous system sessions into one graph, showing several interesting things. First, those graphs show events occurring often within particular sessions or within all sessions. The color of edges is related to that frequency. Applying that method to attack logs gathered from honeypots allow us to detect what potential flaws (of the policy) where actually exploited by real attackers. We concentrate on information flows during attacks. Those are valuable results, as we whish to be able in the future to react on those policies and concentrate on the more dangerous paths on the graph among the millions of flaws in the policy.

The paper is organized as follows. Section 2 surveys related work. Section 3 presents our approach for SELinux system policies visualization using interaction graphs, information flow graphs and tracking information flows within policies. Section 4 presents our approach for the visualization of SElinux system logs and, in particular, system logs of attacks. It introduces session graphs, multiple session graphs and multiple session information flow graphs in order to provide statistical views of attacks and information flows during attacks. Section 5 presents the tool and particular algorithms before concluding with perspectives in Section 6.

2 Related Work

2.1 System Policies Visualization

There have been some work done in the field of analysis and visualization of security policies. In [2], the author surveys more than 20 papers about security visualization, and the most parts of them are focused on policies for network tools. Only a part is related to operating systems: it deals with Rule based Resource Access Control (RBAC) security policies visualization [3], but do not provide any ways to track RBAC violation attempts for example. However, some other work on the visualization of system security policies exist. In [4], the authors deal with the hierarchical visualization of NTFS access control policies. Their approach, transforming Access Control Lists subtrees into (sub-)rectangles cannot be applied to whole operating systems policies. The authors of [5] provide a tool for the visualization and comparison of security policies, to determine their conflicts for example. But the paper do not deal with risks or flaws within a single policy alone. In [6], the authors provide a tool for analyzing and visualize security properties but only deals with small policies and do not confront visualization of static policies with real system execution, or attack sessions. The interesting work of [7] go a step further. The authors provide a tool to query about policies violation of simple security properties based on information flows, or others (e.g. separation of duties). But they do not provide a tool to render, explore and filter large policies. They neither consider the problem of confront policies with real system usage or attacks against it.

2.2 System Logs Visualization

Generally speaking, security monitoring or visualization often rely on network security. Visualization of system session logs neither attack logs do not make an exception to that. In [8], the author largely explain how to deal with information security visualization, but the attention is focused on network information. Even when talking about system events, it is focused on exploiting them to gain IPs of the attackers. In [9,10], the authors provides approach and a research prototype in order to deal with large amount of network data but not provide any consideration of system logs and data. Indeed, many work deal with very specific but deeply studied ways of visualizing networks logs and data, such as [11,12]. Many work, such as [13,14] provides interesting ideas for the visualization of attacks, such as displaying the attacks as a force directed graph or visualizing multiple attacks into one single representation. In [15] the approach is to interactively build the graph, which is quite impossible with large ones. In [16], the authors go a step further with providing statistical view of network attacks.

However, all work are strictly focused on network attacks, with logs collected from NIDS (Networks Intrusion Detection Systems). To our knowledge, there exists no work on the visualization of operating system logs of attacks based on HIDS (Host Based Intrusion Detection System) logs. We think this is a gap that should be filled regarding the objectives previously given: better comprehension of system attacks; further comparing system sessions with system policies; and comparing potential flaws in system policies with real exploits in system attacks. Those are our major motivations for this paper.

3 Visualizing Security Policies

In this section we present our method and related tool designed in order to allow the manipulation of real SELinux security policies, and Security Properties violations among them.

Figure 1 describes the functional architecture of our approach. Starting from a SELinux MAC (Mandatory Access Control) policy, we build a policy graph and possibly an information flow graph.

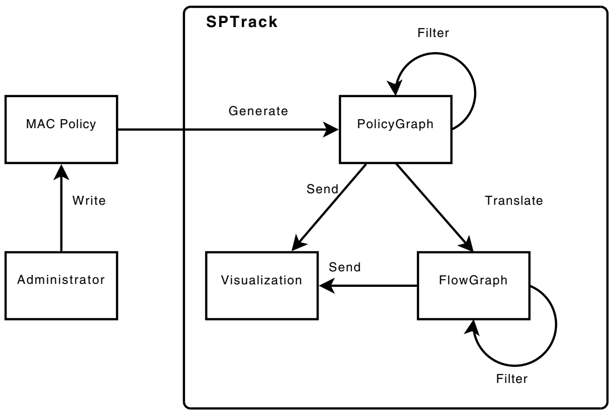

Fig. 1. Functional architecture

3.1 Security Policy Graph

On operating systems, there are entities performing interactions (i.e. operations) on other entities of the system. Under SELinux, such entities are called security contexts. The active ones, e.g. users, processes are called *subjects* and passive ones (files resources, sockets) are called *objects*. A SELinux security policy is defined by rules giving "security permissions" (e.g. read/write, exec) between subjects and objects, regarding the "security class" of the object (e.g. file). Security contexts are represented as a 3-uple of user, role an type related to a given entity. For example, many contexts are related to root. They can be for example like the following: user = $system\_u$ (i.e. acting as, or belong to), role = $object\_r$ (i.e. a passive entity), and type = $home\_t$ (i.e. the root's home directory). Such a context will represented as: $system\_u : object\_r : home\_t$.

We represent SELinux security policies as graphs of contexts where nodes are security contexts subjects or objects and where edges are interactions permissions. More precisely, each interaction is a couple of ($security\_class$, $security\_permissions$).

A complete SELinux policy is very complex to define and to administrate : thousands of contexts and interactions. Obviously, it is difficult to visualize in a useful way. This is why we introduce information flow graphs.

3.2 Information Flow Graph

Moreover, as we focus here of information flows related security properties, we introduce the notion of information flow graph (IFG). An information flow graph related to a given policy is a sub-graph representing only interactions (edges) able to transfer information between contexts.

Criticality level of interactions. We use a mapping table to build this subgraph regarding for each couple ($security\_class, security\_permission$) which type of flow it is and how much its ***criticality*** level is. The criticality refers to the potential erasability provided by the interaction (syscall) for the subject to alter the integrity or the confidentiality of the object. For example, the interaction couple $(signal, send)$ from a context a to another context b is seen as an information flow from a to b with a low level criticality. On the other hand, the context $user\_u$ $(user\_u : user\_r : user\_t)$ performing a $(file, write)$ operation to a context $user\_home\_t$ $(user\_u : user\_r : user\_home\_t)$ as a direct flow from $user\_u$ to $user\_home\_t$, called a $write\_like$ operation, is highly critical because it alter the integrity of the object. Typically, *read* and *write* syscall are the most critical, whereas *signals* for example are the less ones. Typically, when considering only highly critical flows, the resulting sub-graph of information flows is generally very smaller than the full one. The criticality level is used to color the graph from the potential danger of the flow. An example is given in the next subsection.

Such a vision is quite more interesting to focus on than any previous work. We defined 4 levels of criticality regarding the values of our mapping table. In terms of colors, the lowest criticality can be *green* (level 0 to 7), *blue* (8 to 15), *yellow* (16 to 25), *red* (26 to 40): the most dangerous flows.

For example, in order to maintain an acceptable confidentiality property between any *root* context $(system\_u : *)$ and the *user* $(user\_u : user\_r : user\_t)$, one would have to verify for any flow from $system\_u : *$ to any *user*'s exists above the *yellow* level. For a completely restrictive property, no flow at all should be possible.

Our visualization tool provides many classical search algorithms that takes into account the criticality level of the edges.

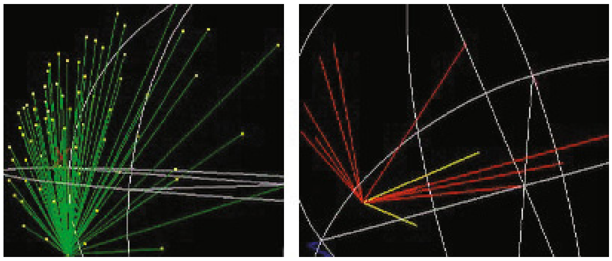

Fig. 2. Security policy graph vs information flow graph

The left part of Fig. 2 shows an example of the visualization of a small security policy with only 71 nodes and 71 edges. The nodes' labels have been removed in order to only show the reduction scale of using the IFG. The right part of the figure presents the corresponding information flow subgraph, recolored according to criticality values. On Fig. 2, only mid-critical (yellow) and highly critical (red) flows were kept from the left side.

With large IFG, it can be more visually helpful to filter graphs to only paths between nodes (or superset of nodes), especially for information flow paths. For example, in Fig. 3, the graph is filtered in order to display only information flows between two security contexts (*root_u* : *object_r* : *nscd_var_run_t* and *sysadm_u* : *sysadm_r* : *bootloader_t*), and the labeling of nodes is made only on the highly critical level (in red). The red path here is composed of 4 nodes (i.e. includes 2 intermediary nodes).

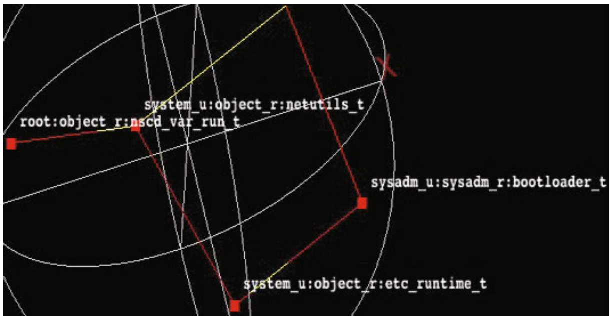
system_u:object_r:netutils_t
root:object_r:nscd_var_run_t
sysadm_u:sysadm_r:bootloader_t
system_u:object_r:etc_runtime_t

Fig. 3. A potential flow in a filtered policy

Thanks to that approach, the administrator can list the remaining security contexts involved in the information flows displayed. He can then select some of them to precisely analyze and track to/from what contexts can information flow occur with them.

For the real policy we used in our experiments, i.e., a SELinux Gentoo hardened policy, having 1703 security contexts and 175190 aggregated[1] edges between contexts, filtering should be guided by more concrete and actual information. That is why we worked on the analysis of real system sessions, and in particular, real attacks sessions, gathered on our high interaction honeypots. This is the purpose of the next section: visualizing system logs and in particular attacks system logs.

4 Visualizing Attack Session Logs

The purpose of this section is to present how we deal with attack session logs in order to create graph representing system sessions of attacks. Figure 4 describes the global system logs visualization architecture: the construction of sessions graphs (ITG), multiple session graphs and multiple session information flow graphs (see below).

Firstofall, we propose to factorize multiple sessions into one single graph. These session logs were gathered during two years from our high interaction

[1] Each edge contains all possible permissions between the 2 linked context, thus leading to an average compression ratio of the number of edges by 75%.

honeypots[2] and thus represents attack sessions, generated with SELinux under auditing mode. Each session logs has been formatted and stored on a large DB2 database with a unique session number. The database stores 130Gb of data divided into 200 millions lines, representing more than 31,000 attack sessions.

4.1 Session Graph

For each attack log, we are able to construct the corresponding so-called session graph. (SG). That graph contains all interactions made since the beginning of the attacks session (e.g. *login* via *ssh*) till its end (e.g. *logout*). The SG shares the same structure as the security policy graph detailed in the previous section. The main difference is that its edges represent interactions that were *actually* performed during the session.

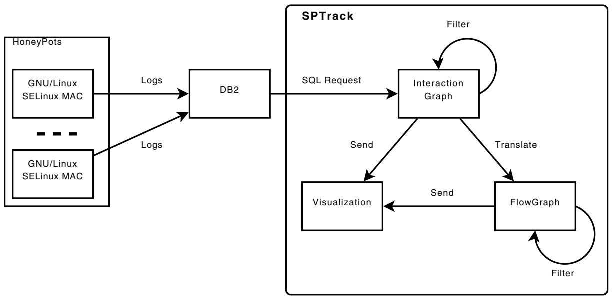

Fig. 4. Session visualization architecture

4.2 Multiple Session Graph

Sessions graphs are very useful to analyze session under our tool. Moreover, as we collected multiple attack sessions, we wanted to make some statistical visualization of multiple sessions, by using one single graph, called multiple session graph (MSG), synthesizing all sessions, with the hypothesis that this will let emerge repetitive behaviors of attackers, or very odd ones.

Our approach consider two criteria for the construction of the MSG, and in particular, for each edge of this graph: (a) the absolute number of occurrences of a given interaction among all the sessions where it occurs; (b) the number of sessions where this interaction occurs.

To ponder the first criterion, we took its logarithm as we thought that the number of sessions if more relevant for the consideration of all attackers behaviors. The following equation 1 is used to assign a score to each edge during the construction of the graph, i.e. during the parsing of the attack logs

[2] High interactions honeypots are real machines, not emulated ones, in order to capture attackers' activities.

stored on our database. *edge_number* represent the criterion (a) above, whereas *session_number* is second criterion (b).

$$score = \big(1 + \ln(edge\_number)\big) \times session\_number \qquad (1)$$

As a result for visual exploitation, we defined a color set related to four score ranges following a logarithmic scale, that can be intuitively explained as the following;

– blue: the interaction occurred only one time,
– green: the interaction occurred few times,
– yellow: the interaction occurred multiple times,
– red: the interaction occurred in all stored sessions.

In our experiments, 50 attack sessions that consist of 2 millions of lines of sessions logs were reduced to a graph with 150 nodes and 130 edges. Actuelly, the MSG aggregates as many sessions logs as we can have. In practice, it is easy to build the total graph of the 31,000 attacks sessions we gathered of the honeypots.

4.3 Multiple Sessions Information Flow Graph

In order to compare potential flaws in policies and real exploit by attackers, we introduce multiple session information flow graphs MSIFG. They are build following the same pattern as the policy IFG. We use the criticality value for each elementary operation. Higher the criticality of elementary operation is, higher the risk of a real information flow has happened on the system. The criticality level can be also used to reduce the graph size by only displaying the real sensitive operations (e.g. red edges) within all sessions. Thus, we are able to find critical paths within the aggregated sessions based on the computed score. Within figure 5, a path can be seen, between the SSH daemon (*system_u* :

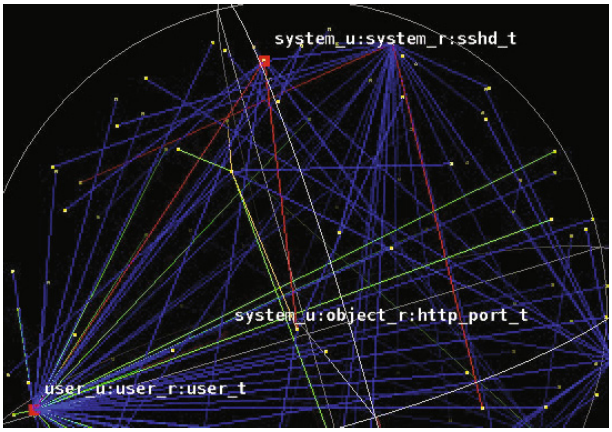

Fig. 5. Flow within a MSIFG

system_r : *sshd_t*) and user (*user_u* : *user_r* : *user_t*) through the the HTTP port (*system_u* : *object_r* : *http_port_t*) that goes. It seems to be a path of an attacker getting an interactive shell (*ssh*) via an *http* port. Such a path is quite surprising and would motivate the administrator to closer look at the entire session of that attack. He would then also be able to react adequatively. Moreover, he should track on the policy all possible paths between user and the SSH daemon for example.

5 Conclusion

In this paper, we have presented a global platform called SPTrack[3] for the visualization of both SELinux security policies and SELinux system logs. We have provided method and algorithms to apply the visualization tool to the detection and the visualization of possible information flows on the system regarding its security policy. To do that we transform interaction graphs into information flow graphs, where paths are information flows and edges' colors are criticality level of edges. In parallel, we have provided implemented algorithms to the visualization of SELinux system logs applied to attack sessions gathered from our high interaction honeypots. We have exhibited methods to aggregate thousands or even millions of attack sessions into one single MSG. By transforming MSGs into MSIFGs, we have finally provided ways to analyze the most/least usual information flows among numerous attacks, and furthermore we abled to track those flows given a criticality level.

Currently, our solution can only deal with security properties designed in terms of information flows. However, information flows can be sufficient to model many security properties (integrity and confidentiality of data, subjects, users, domains, binaries, groups). We are working to express more precisely what security property one can analyze. Moreover, we may extend the security properties available: the user should be able to specify which security property he wants to study in the policy/logs. Moreover, the user should be able to select a criticality threshold to those studies, in order to focus 'more dangerous' paths.

Those properties should also be non-information flow based properties, such as separation of duties (no modification of an object followed by its execution from the same user), or trusted path execution for examples (all execution are made within specific binaries repositories).

References

1. Briffaut, J., Lalande, J., Toinard, C.: Formalization of security properties: enforcement for mac operating systems and verification of dynamic mac policies. International Journal on Advances in Security 2(4), 325–343 (2010)
2. Tamassia, R., Palazzi, B., Papamanthou, C.: Graph Drawing for Security Visualization. In: Tollis, I.G., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 2–13. Springer, Heidelberg (2009)

[3] SPTrack is based on Walrus open source code [17].

3. Montemayor, J., Freeman, A., Gersh, J., Llanso, T., Patrone, D.: Information visualization for Rule-Based Resource Access Control. In: Proc. of Int. Symposium on Usable Privacy and Security (SOUPS), Citeseer (2006)
4. Heitzmann, A., Palazzi, B., Papamanthou, C., Tamassia, R.: Effective visualization of file system access-control. Visualization for Computer Security, 18–25 (2008)
5. Rao, P., Ghinita, G., Bertino, E., Lobo, J.: Visualization for Access Control Policy Analysis Results Using Multi-level Grids. In: 2009 IEEE International Symposium on Policies for Distributed Systems and Networks, pp. 25–28. IEEE (July 2009)
6. Wahsheh, L.A., Leon, D.C.D., Alves-Foss, J.: Formal Verification and Visualization of Security Policies. Journal of Computers 3(6), 22–31 (2008)
7. Xu, W., Shehab, M., Ahn, G.J.: Visualization based policy analysis: case study in SELinux. In: SACMAT 2008: Proceedings of the 13th ACM Symposium on Access Control Models and Technologies, pp. 165–174. ACM, New York (2008)
8. Marty, R.: Applied Security Visualization. Addison-Wesley Professional (2008)
9. Kolano, P.Z.: A Scalable Aural-Visual Environment for Security Event Monitoring, Analysis, and Response. In: Bebis, G., Boyle, R., Parvin, B., Koracin, D., Paragios, N., Tanveer, S.-M., Ju, T., Liu, Z., Coquillart, S., Cruz-Neira, C., Müller, T., Malzbender, T. (eds.) ISVC 2007, Part I. LNCS, vol. 4841, pp. 564–575. Springer, Heidelberg (2007)
10. McPherson, J., Ma, K.L., Krystosk, P., Bartoletti, T., Christensen, M.: Portvis: a tool for port-based detection of security events. In: VizSEC/DMSEC 2004: Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security, pp. 73–81. ACM, New York (2004)
11. Ma, K.: Cyber security through visualization. In: Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation, vol. 60, p. 7. Australian Computer Society, Inc. (2006)
12. Ball, R., Fink, G., North, C.: Home-centric visualization of network traffic for security administration. In: Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security, pp. 55–64. ACM (2004)
13. Mansmann, F., Fischer, F., Keim, D.A., North, S.C.: Visual support for analyzing network traffic and intrusion detection events using TreeMap and graph representations. In: Proceedings of the Symposium on Computer Human Interaction for the Management of Information Technology, CHiMiT 2009, pp. 19–28. ACM Press, New York (2009)
14. Hideshima, Y., Koike, H.: Starmine: a visualization system for cyber attacks. In: APVis 2006: Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation, pp. 131–138. Australian Computer Society, Inc., Darlinghurst (2006)
15. Noel, S., Jajodia, S.: Managing attack graph complexity through visual hierarchical aggregation. In: Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security, VizSEC/DMSEC 2004, p. 109 (2004)
16. Luse, A., Scheibe, K., Townsend, A.: A Component-Based Framework for Visualization of Intrusion Detection Events. Information Security Journal: A Global Perspective 17(2), 95–107 (2008)
17. CAIDA: Walrus - Graph Visualization Tool (2009), `http://www.caida.org/tools/visualization/walrus/`