TOWARDS WORKING SET BASED APPROXIMATION OF LEAST PRIVILEGE PRINCIPLE FOR OPERATING SYSTEMS

Christian P. Praher

Institute for Information Processing and Microprocessor Technology (FIM)
Faculty of Engineering and Natural Sciences
Johannes Kepler University Linz
praher@fim.uni-linz.ac.at

Keywords

Security, Access Control, Principle of Least Privilege, Operating System, Working Set

Abstract

The principle of least privilege is becoming more and more important in access control as it can mitigate the consequences of attacks performed by malicious code or ill intended users. Establishing a strict least privilege access control policy is however very hard to achieve as it requires intensive administrative configuration. In this paper we propose a novel approach based on working sets that consider a user's past access control resource usage for approximating least privilege in an automated manner. We present the results of a first empirical analysis based on a simple variant of the model. The key concepts of an extended model which is currently being formalized and has been influenced by findings from the first data analysis is also provided.

1. Introduction

Contemporary main stream operating systems by default do not very well adhere to the principle of least privilege which states that every application should run with the least set of privileges needed (Saltzer, 1974). They rather employ an ambient authority (Watson, Anderson, Laurie, and Kennaway, 2010) security model in which a process runs in the security context of the user that started it. Consequently by default the entire logon session of a user runs with the same set of privileges, irrespective of the access rights really needed by the individual applications.

The reason for the lack of compliance with this important access control principle is rooted in the fact that true least privilege is hard to achieve. After all the permissions needed by an application are a direct result of the exercised program code. In order to know which privileges are needed for an application one would have to run the program so long as to execute every possible runtime execution path, or alternatively do a complex static analysis of the executable binary code first.

In this paper we propose a novel approach of approximation of least privilege. This method is based on the idea of working sets (Denning, 1968) created from past per user access control usage. The main fundamentals of our model are the assumption that from the large number of available applications and consequently permissions only a small subset is actively needed by the average

user. These most recent applications are what make up our (main) working set whereas other executables are not directly accessible anymore. Our second basis is the learning of typical application access control behavior inferred from runtime analysis of the exercised application permissions. First experimental evaluation conducted on a simple variant of our model make us confident that these ideas are applicable to multi purpose operating systems.

2. Related Work

Besides the mandatory operating system access control frameworks like e.g. TOMOYO Linux (Harada, Horte, & Tanaka, 2004) or Rule Set Based Access Control (RSBAC) (Ott, 2007) that focus on least privilege but require extensive administrative configuration, the only systems the author of this paper is aware of that directly rely on and incorporate the idea of working sets to achieve least privilege are Dynamic Sessions in Role Based Access Control (DSRBAC) and Working Set-Based Access Control for Network File Systems (WSBAC).

DSRBAC was developed by Mühlbacher and the author of this thesis (2009) as an extension to the RBAC model. In this model every role is associated with a time to live (ttl) value. A session is regarded as the working set of roles in which unused roles can expire and be re-added by the user. The choice of which role to expire in an active session is regulated by a well ordering according to a role's permission mightiness. As its name implies DSRBAC is tailored to the RBAC model and its inherent role concept.

WSBAC by Smaldone, Ganapathy, and Iftode (2009) introduces a per user working set for differentiating access to files on a Network File System (NFS) server depending on the location and the device of the user. Requests are treated differently coming either from within a trusted stationary in-house PC or a potentially insecure mobile device. After an administrative adjustable time, e.g. one day, only those files can be accessed from the mobile device that have been used on the workplace PC. The key difference between WSBAC and the model presented herein is the different scope of NFS server versus operating system and the associated difference of the items contained in the working set. Also WSBAC does not consider multiple working sets as well as it does not incorporate or mention a continuous refinement of the working set through a working set trimming function.

3. Basic Idea and First Experimental Evaluation

The goal of our model is to overcome the traditional ambient authority in operating system access control and instead create a positive security model (Ristic & Shezaf, 2008) which by default denies unknown or unusual access requests in contrast to simply allowing them.

Our main concept is a working set which represents the current access control locality on a per user basis. As every user interaction with the operating system (kernel) is realized through processes we regard processes as the basic content for our working set. There are two distinct phases in the system. The first one is an initial learning phase in which the working set of the user is established on a per session basis. After the learning phase a user may instantly only access those applications that have been executed in the learning phase and are currently in his/her working set. If the user runs and application which is not part of the current working set he/she explicitly has to consent its execution, e.g. through a command prompt similar to the Microsoft UAC consent prompt (Russinovich & Solomon, 2009). It is important to emphasize that the working set will

continuously be adjusted after the initial learning phase, by executing previously unknown applications and by a special trimming function for keeping the working set current and removing unneeded applications.

3.1. Considered Data and Particular Research Questions

In order to assess whether the idea of an application working set is appropriate for average operating system usage we conducted a first experimental study with three standard user clients over three months (from July to October of last year). The users are employees of a small trading company working in the positions of executive, salesperson and secretary/accountant. In this evaluation we were particularly interested in how intrusive the positive security model of the working set would be. Of special interest was thus how often applications not being part of the working set would cause a *privilege fault*, meaning need to be acknowledged by the user to be run. The second question was how well the working set could approximate the applications needed by the user on a per session basis.

The common operating system platform of all three clients was Windows XP[1] and the application usage was collected by means of a Windows service using the Windows Management Instrumentation (WMI) collecting every instance creation and instance deletion event of every process. An instance creation event is raised whenever a new process is started (e.g. through double-clicking the executable file in the graphical shell) and an instance deletion event is raised as soon as the process is terminated (e.g. by pressing the close button in the window title bar).

Similar to the concept of domain paths in TOMOYO Linux (Harada et al., 2004), applications are uniquely identified by their parent/child relation and position in the current process tree. E.g. the same process binary for the Firefox web browser would be regarded as an individual item in the working set depending on whether it was called by clicking an icon on the desktop (`/explorer/firefox`) or by opening an HTML attachment in the Thunderbird mail client (`/explorer/thunderbird/firefox`). Every such application path not contained in the current working set would cause a privilege fault.

3.2. Working Set Establishment and Trimming Function

The core of the analysis is a trimming function for establishing the fist working set in the initial learning phase and for keeping it as small as needed afterwards. The only predefined input needed by the function shown in figure 1 is the (administrative adjustable) number of learning sessions (`n_learn`). The algorithm is a mixture of Least Recently Used (LRU) and Least Frequently Used (LFU) and is based on a simple scoring system for deciding whether an application should be part of the next session working set or not.

The algorithm is straight forward. Every first seen application is initialized with a value equal to the number of sessions for learning (`n_learn`). If the application was contained in the previous working set and also exercised in the current session, its score gets incremented. If otherwise the application is part of the working set but was not run in the current session its score gets decremented. To account for sporadically used applications a frequency factor is considered for re-faulting applications which means applications that have been part of the working set but have been

[1] As far as the “outdated”' operating system Windows XP, which was given, is concerned it is worth mentioning that the described concept is not dependent on any particular OS

removed. They are awarded a score which is a multiple of `n_learn` and the number of times they have been removed from the working set.

```
n_learn = Administrative specified number of sessions for learning phase
session_ts = Current session timestamp (auto increment, initially 1)
session_applications = Set of applications executed in current session
all_applications = Set of all applications ever executed
is_learning = (session_ts <= n_learn)

for application in all_applications:
  if (application.new_in_session(session_ts)):
      application.set_score(n_learn)
  elif (application.faulted_in_session(session_ts)):
      application.set_score(application.get_fault_count() * n_learn)
  elif (application in session_applications):
     application.set_score(application.get_score() + 1)
  else:
     /*Application was not in current session */
     application.set_score(application.get_score() - 1)

  /*Establish and trim the working set*/
  if (not is_learning and application.get_score() <= 0):
     remove_from_ws(application)
  else:
     add_to_ws(application)
```

Figure 1: Working Set Establishment and Trimming Function

3.3. Results of the Evaluation

For the analysis of the collected data the learning threshold (`n_learn`) was set to 5. This seems reasonable given the fact that the work week of the observed users is 5 days, so after these 5 days it can be expected to have seen a user's full week application usage.

Table 1 shows that the average number faults per session are reasonably low with no more than 2 faults per user and session. This means that on the average a user will only be asked for consent once or twice in a session. The average number of applications in the working set is however significantly higher than the average number of applications executed in one session. This ratio is unnecessarily high which lead us to an improvement of the model described in the next section.

	Sessions after learning	Faults after learning	Average faults per session	Average session size (Given as number of applications)	Average working set size (Given as number of applications)
Executive	61	103	1.69	43	66
Accounting	51	88	1.73	40	61
Sales	30	44	1.47	35	50

Table 1: Basic Statistics of Evaluated Data

We present the working set chart of the user accounting/secretary in figure 2 and add that the values of the other users do not differ significantly. The working set provides a good upper limit and approximation of the average user session. The x-axis of the chart displays the observed sessions and the y-axis describes the number of executed applications. After the 5 learning sessions the working set mostly remains stable. Outliers are only found between session 35 and 42 which indicate system update installations with the characteristic phases of running many applications in a session as well as frequent restarts. As described in the next section our extended algorithm is not prone to working set distortion due to system updates. Still the figure shows that the trimming of unneeded applications from the working set works fairly reasonable, which is indicated by the working set line quickly returning to its normal value.

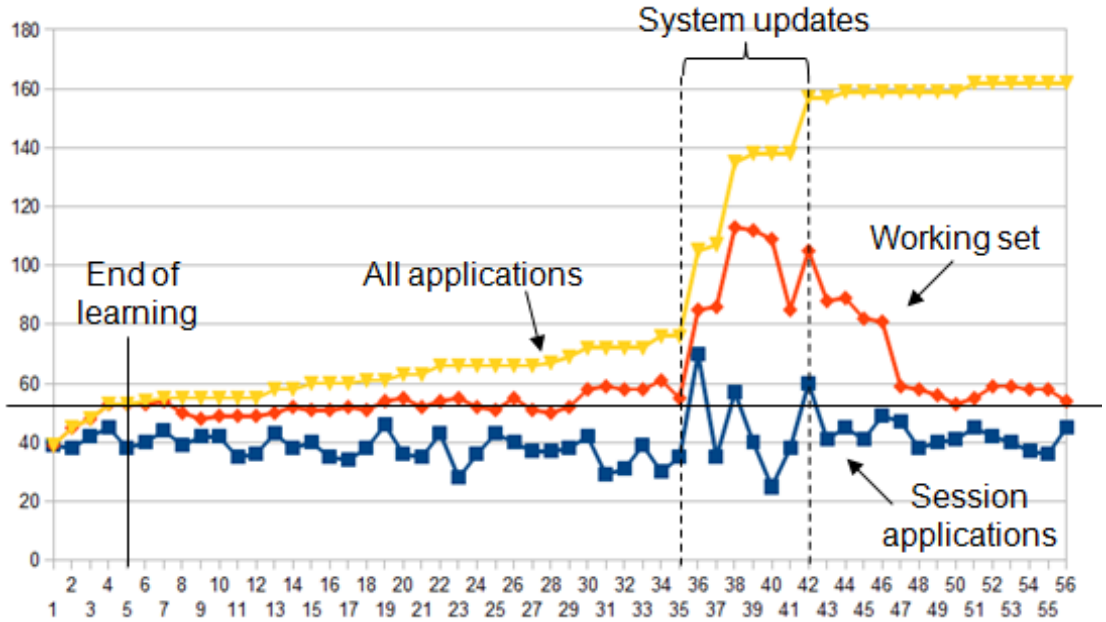

Figure 2: Working Set Charts User Accounting/Secretary

4. Extended Model

Based on the presented empirical analysis we were able to conclude important improvements over the original model which will be presented in this section.

4.1. Key Properties of the Extended Model

4.1.1. Application permission usage

A key aspect of the consolidated model is to know for each application its permission usage. It is proven by mandatory security frameworks like Grsecurity, or TOMOYO that automated policy learning works fairly well and is able to derive accurate per application policies after only a short number of application executions. Our goal is to derive so called *application roles* that resemble the typical program behavior in the current user context, like e.g. loaded libraries, network activity, inter process communication (IPC) or typical file associations. In this way we try to detect not just unknown applications but also unknown application behavior.

4.1.2. Multiple, differentiated working sets

As described in the previous section an important finding of the first empirical analysis is that the average number of applications in the working set is 50 % higher than the mean number of applications executed in a session. Also in the algorithm presented in this paper (see figure 1) every session is started with the current working set as the basis of directly available permissions, which on the average contains 50 % more applications than needed. It could be observed however that the average number of applications executed in one session remains relatively constant. To remedy these deficiencies the final model will provide multiple working sets. Most importantly a *session set* that is empty at the start of a session and only contains the applications executed in the current session. The size of this session set could be calculated automatically as the average number of executed per session applications or be set manually (by an administrator). Extending the size of the session set could also have different consequences ranging from none, over a warning of executing unusually many applications, to a hard boundary which would require the user to close an application first before being allowed to open a new one.

Eventually we imagine the model to have a differentiated and rich set semantic with an active day, week, month and complete user set that allow identifying periodically used applications as well as providing varied challenges, depending on the set a called application currently is contained in.

4.1.3. Calendar function for being independent from actual logon session

Another short coming of the simple model we identified is the dependency on the “physical logon session”. The operating system logon session is not the ideal unit for updating the working set. Logon sessions are often constraint by the operating system, e.g. if a restart is forced because of security relevant updates. Besides from that, users nowadays often use features like hibernation or power save sleep mode which do not end the physical session but possibly keep it open for days or even weeks.

That is why our extended model introduces a *calendar function* for defining the learning unit for updating the user working set. A simple and effective period of considered time is probably one

calendar day which much better reflects what a user regards as a unit of work than the physical session created by the operating system.

4.1.4. Only consider user executed applications

In the first analysis all applications started while a user was logged in were considered. Although the computers were strict single user machines the traces included applications that are not directly related to the user actions, like e.g. processes started by services. This is reflected in the peak of session applications shown in figure 2 between session 35 and 42 which results from multiple system updates. However the application/privilege usage of the user did not change within these sessions but the surge of applications was rather due to a user unrelated platform specific maintenance task. E.g. on Windows operating systems getting only user related applications can be achieved by only considering the applications associated with the interactive logon session (Brown, 2000).

4.1.5. Consider session activities

To counter distorted and overly reduced working sets resulting from the application of the calendar function and an abandoned session the model should consider the foreground activity of running applications. This allows switching off the working set trimming if the session obviously is orphaned. So the model will automatically stop refining the working set if e.g. a user starts a long lasting calculation or backup process and leaves the computer for a longer period of time.

4.2. Security Goals

The principle of least privilege is approximated by the model in two ways. Firstly the working set reflects a user's current privilege requirement in the form of application roles. This is clearly beneficial over the standard situation in which all applications are always available independent of whether they are ever needed by the user or not. Secondly the application roles create a clear definition of the normal activities of an application. After a short learning phase the working set model thus creates a view that much more closely reflects the actual access control needs of a user than the standard operating system mechanisms can provide.

Given this situation the first security goal of the working set model is to identify unknown applications in a user session. With the tight observation of the user application activities it should not be so easy for a malicious application anymore to stealthy install and run itself on the computer. As mentioned we believe that asking the user for consent whether to allow running an application is a good starting point. However we also plan to incorporate a sandbox concept into the model that automatically launches new and untrustworthy applications that are not part of the working set in a secure environment. This would bear the advantage of shielding the trusted applications from the new application. Also it should greatly increase performance in contrast to running all applications in the sandbox. As the trustworthiness of the application matures through the building of an application role, it can eventually be released from the sandbox.

The second major security goal we see in the incorporation of the proposed model is that through the application roles an apparent change of application behaviour caused e.g. by malware infection should be immediately visible

5. Conclusions and Future Work

In this paper we have presented a first empirical analysis of a working set based access control mechanism that should automatically aid users in restricting their permissions to the actual resource usage and help overcome the current situation of ambient authority in operating systems. After the evaluation of first statistical data we are confident that this model is not too obtrusive and that it can help in providing intelligent user sessions that do not contain unneeded permissions.

To formalize the extended model proposed in this paper, we are currently undertaking a more comprehensive study involving multiple users and different types of contemporary operating systems (Windows 7, Vista and XP). In addition to the process creation and termination events we captured in the analysis presented in this paper, more access control related information described in the functioning of the extended model, like e.g. opened file handles (including granted access to files, directories, registry keys, etc.), TCP/IP based network communication, loaded DLLs and some major system calls are collected.

6. References

American National Standards Institute, Inc. (2004). American National Standard for Information Technology - Role Based Access Control (ANSI INCITS 359-2004).

Brown, K. (2000). Programming Windows Security. Addison-Wesley Professional; First Edition. ISBN: 0201604426

Denning, P. J. (1968). The working set model for program behavior. Commun. ACM, 11/5 1968, 323-333.

Harada, T., Horte, T., & Tanaka K.. (2004). Task Oriented Management Obviates Your Onus on Linux. Linux Conference 2004.

Mühlbacher, J.R., & Praher, C.P. (2009). DS RBAC -- Dynamic Sessions in Role Based Access Control. In: J.UCS - Journal of Universal Computer Science 15 2009, 538-554.

Ott, A. (2007). Mandatory Rule Set Based Access Control in Linux. ISBN: 978-3-8322-6423-9

Ristic, I., Shezaf O. (2008). Enough with default allow in web applications. Black Hat USA 2008.

Russinovich M., & Solomon D. A. (2009). Windows Internals, Fifth Edition. ISBN: 0735625301

Saltzer, J. H. (1974). Protection and the control of information sharing in multics. Communications of the ACM 17, 388- 402.

Smaldone, S., Ganapathy, V., & Iftode, L. (2009). Working SetBased Access Control for Network File Systems. Proceedings of the 14th ACM symposium on Access control models and technologies SACMAT '09, 207-216.

Spencer, R., Smalley, S., Loscocco, P., Hibler, M., Lepreau, J., & Andersen, D. (1998). The Flask Security Architecture: System Support for Diverse Security Policies. Proceedings of The Eighth USENIX Security Symposium 1998, 123-139.

Watson, R. N. M., Anderson, J., Laurie, B., & Kennaway, K. (2010). Introducing Capsicum: practical capabilities for UNIX. ;login: Magazine December 35/6.