Saint Jude, The Model

Saint Jude is a model for the detection of unauthorized or improper privilege transitions. This document describes this model, how it works, its limitations and how the limitations may be compensated for through the utilization of other protective measures.

The application of this model, and other works similar to it may be used as trigger mechanisms in the development of intrusion resilient systems. It is the hope to produce a first generation trigger for such intrusion resilient systems that is capable of detecting attacks against the integrity of well defined systems prior to the realization of the attack objective, and to be able to do so for attacks with unknown signatures. The implementations of this model that are presented later contain a simple payload, aiming to neutralize the treat the compromised process may pose to its host system.

For simplicity, in discussions of the model we look at only two possible privilege states: privileged (or root) and unprivileged[1]. Through this document the terms "root" and "privilege" are used interchangeably. It should be noted that the Saint Jude model might be extended with little modification to protect multiple states of privilege, not just a single privileged state.

Improper Transitions

A transition is the exchange of one executable image in memory for another via a well defined method, such as execve(). An improper transition is any transition that occurs as the result of tampering with the intended flow of execution of an application. This tampering may come in the form of buffer overflow exploits, feeding of data to exploit unchecked or handled conditions, race conditions, or other techniques used to cause an application to execute other applications that the original application was not designed to execute, or in fashions not intended.

Model Definition

Below is the definition of the Saint Jude model.

Given:

- An Application is a binary image residing on secondary storage that may be executed either directly by the operating system, or by an interpreting process.
- A process is a running instance of an application.
- Each process within a system is either in a privileged or unprivileged state.
- Both privileged and unprivileged processes execute applications via the execve() syscall.
- Unprivileged process may acquire privilege by executing a setuid application.
- Privileged processes may loose privilege by calling exit(), setuid(), or setreuid() (linux). For setuid and setreuid, the arguments passed to the functions must be non-zero.

[1] With two states, we are only concerned with two transition states, for a greater number of states our rule-base would grow as the number of possible transitions would grow at a rate of n^2-n.

The model states:

- Each privileged process is associated with restriction list
- Each restriction list is based on a global rule-base.
- If a privileged process calls fork(), vfork(), or clone() the resultant child process will inherit the restriction list from its parent process.
- The "Best Match" for any single application in the rule-base is defined as the rule that:

  1. Matches the filename of the application executed.
  2. Matches the most command line arguments passed to execve() along with the application.
  3. Is a subset of the prior restriction list, if it exists.

  Note: There will be no prior restriction list in the case the transition from an unprivileged state to a privileged state.

- Upon loosing privilege, the privilege list for a privileged process is destroyed, and the process is no longer monitored.
- Upon acquiring privilege, the newly privileged process has a privilege list generated for it, and is monitored until it looses its privilege.
- Upon an execution request, the request is checked against the process's restriction list.

  If a matching rule is found, the execution is permitted; otherwise, the execution is aborted and the process is flagged as violating the model.

  If the execution is permitted a new restriction list is generated based upon the best match rule-base for the new application.

The Rule-Base

The rule base is generated on a system-by-system basis during a learning phase. This learning phase is conducted prior to deployment of the system and in a secure environment.

The rule base consists of a collection of rules, associated with a key. The key is the application name (i.e., /bin/bash) and any command line arguments that may be passed to it.

Ex, The following are 3 different keys because of different arguments.

```
/bin/bash -c
/bin/bash -s
/bin/bash
```

When searching for a matching key we should look first for the most specific and proceed to the most general by pruning the command line from the right to left until a match is found[2].

The rule base will always contain two special rules -- the default rule, and the unrestricted rule. Both of these rules have the same key, an empty string (i.e. "").

The default rule has an empty list, meaning any process associated with it may not execute any other application without first loosing its privilege.

The unrestricted rule exits to permit administration on systems employing the model. By executing defined gateway applications (i.e. /bin/su) a privileged process may be generated that would allow administrative individuals to maintain the system. Upon running the defined gateway application, the resulting process (and all of its subsequent processes until it looses privilege or encounters an OVERRIDE) are associated with the unrestricted rule.

Because of the "opt out" of the model granted to the gateway application(s), very few (if more then one) should ever be defined. Further, special care must be taken to ensure that the binary image of the gateway application is protected. (Read only media, or deployment of file-system protection such as LIDS.)

If an application is executed by a process associated with the unrestricted rule, and the OVERRIDE is the first rule associated with he executed application, the unrestricted rule will be replaced with the remainder of the rule associated with the executed application. This permits for the execution of daemon processes by individuals entrusted with maintenance of the system without allowing the daemon processes to inherit the unrestricted rule.

Learning

During the learning phase the model is run with a minimal rule-set consisting of the Default rule, the unrestricted rule, and whatever gateway rules there will be. Everything will occur just like normal, with the exception that when a violation of is identified it is recorded and the execution is permitted to occur.

Any violations discovered during this phase should be considered to be normal for the systems operation, but unknown to the model. The output from the learning phase is fed into the model in the form of an updated rule base.

During the learning phase all activities that the system will see should be simulated.

Model Classification

The Saint Jude model, when employed within the context of a reference monitor located internal or external to the kernel is a Rule-Based IDS system. Rule-Based IDS was first proposed by researchers at Los Alamos National Laboratory and was deployed as the Wisdom and Sense system.

The benefit of Rule-Based IDS is that after the rule-base has been populated with the known legal actions within the system, there is no chance of false positives. All positive results are of a significant nature, either being a legitimate intrusion or an unknown behavior of the monitored system.

[2] The argv[0] will always have to be replaced with the filename as it is passed to the execve() call. This ensures we always have the fully qualified path name of the file.

Limitations

The Model is limited by the amount of work which will be required to verify and validate the rule-base after any modification has been made to a running system/configuration, such as the introduction of new applications that may be spawned by a privileged process. Indicators thus far have been that the learning curve on systems running most traditional Unix services has been very short. Workstations, such as red hat 6.2, have a higher demand on the learning phase due to numerous scripts that run as root to control functions such as dialup network connections, pcmcia services, audio mixers and the like.

Another limiting factor, and one of concern, is the potential risk of a directed attack by an attacker aware of the presence of the Model's implementation. It is conceivably possible for a skillful attacker to construct a buffer-overflow attack in such a way that they do cause a transition, but rather introduce 'shell code' that would replace the gateway applications, modify system configuration files, circumvent or disable the model's implementation. For this reason it is recommended to protect the file systems that contain the base operating system, the model implementation, and any gateway applications.

Modifications can may be made internal to implementations to reduce success rate of attacking the implementation itself, but it is necessary for other autonomous security measures exist to further harden the system, protect the model, and further restrict the possible successful vectors of attack.

Of particular interest in the Linux scope is the work of the PaX group at http://pageexec.virtualave.net, their modifications to the Linux kernel further modify the underlying operating system in such a way that the vector of attack seen will be within the protective bounds of the StJude model, and eliminating the possibility described above.

Implementations

Currently there are two public implementations of the Saint Jude model.

StJude.pl

Saint Jude was first implemented on Solaris. It reads audit data coming out of the BSM. This allowed for the development of the model without mucking around in kernel space.

The currently released version is from March 1999 and is versioned 0.4.

There are issues with this release that have been addressed in the Linux version. Problems arise with applications such as the Solaris printing system, where a system() call is made when an error occurs printing a job. The system call spawns /bin/sh -c "/bin/write". The StJude.pl does not take into account command line arguments, and thusly cannot distinguish "/bin/sh -c" from "/bin/sh".

Further, Solaris 8, prior to service release 1, will not be able to support argv because of a bug in the praudit utility. Audit tokens were added to the audit system that was not accounted for in the praudit utility. The result is erroneous returns for the argv and envp tokens. (BugID 4339611).

Unless there is expressed interest in applying the argv modification to the StJude.pl, no further revisions will be made, and development work on Solaris will focus on an in-kernel implementation.

StJude_LKM 0.04

Saint Jude Linux Kernel Module is the first in-kernel implementation. Currently Versioned 0.03, it now supports SMP systems via spin-locks. Although it should run on non-Intel systems, there has been no testing on non-Intel platforms due to lack of resources for testing.

This version first introduced the override feature mentioned above. The override was implemented as a separate list of processes that should be stripped of their permit-all status. This was done for simplicity in implementation.

Also, the concern about probing of syscalls was concerned by slightly randomizing and adding user
Input on placement of the syscall for execeve. Around the syscall were placed booby-trap syscalls to pick up probing attempts. At this time there is another solution for this, that is a bit more elegant, in the works. It is hoped to ship it with 0.05 or 0.06.

Currently there are no known issues with this version, but development continues to produce resources to automate the generation of the rule base from the output of the learning phase, and develop a means to shutdown all processes associated with the same session as the offending process when in production mode. SMP support seems to be stable, at least in a Duel-CPU configuration. In a production environment that, the 0.04 dev code was in operation for 25 days without incident.