Saint Jude, the Model

Saint Jude is a model for the detection of unauthorized or improper root 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 unknown of attacks 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.

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 groups of privileges, 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, 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.

The model states:

- Each privileged process is associated with restriction list
- Each restriction list is based on a global rule-base that describes the valid transitions for an application which is run within a defined context.
- If a privileged process calls *fork*(), *vfork*(), *fork1*(), 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[1].

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 Anomaly Detector IDS system. Researchers at Los Alamos National Laboratory first proposed rule-Based Anomalous IDS in the system “Wisdom and Sense” (W&S). Rule-Based anomaly detectors operate similar to standard statical anomaly detectors, except that instead of usage patterns, explicit rules are used to describe good or bad behavior.

The benefit of Rule-Based Anomaly Detection IDS is that after the rule-base has been populated with the known legal actions within the system, the chance of chance of false positives approaches zero, while maintaining a near-zero rate of false negatives. In a similar token, the amount of training necessary to fully

[1] 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.

populate the rule-base can become burdensome, with re-training necessary when significant changes occur within the system.

Implementations

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

StJude.pl (March, 1999)

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 0.20 (July 30th, 2001)

Saint Jude Linux Kernel Module is the first in-kernel implementation of the Saint Jude Model. Currently Versioned 0.21, Saint Jude LKM supports SMP systems, and offers defensive mechanisms to defend the LKM from direct attacks whilst it protects the system from compromise. Saint Jude LKM is still only tested on the x86 architecture.

As of Version 0.10, Saint Jude has detected remote compromises of protected daemons via the Override directive. This has permitted the LKM to detect and avert attacks in the form of Internet worms (Lion, Ramen) and direct attacks using unpublished exploit code (ssh protocol 1.5)

Protection of the on-system binary images of the Saint Jude implementation, as well as files critical to successful system operation is addressed in 0.21 with the addition of features that block the removal of the immutable flags on ext2 and ext3 file systems. The immutable attribute, when applied to a file or directory, inhibits the operating system from modifying the protected file-system object. This effectively renders the file-system object read-only.

In recent months, Saint Jude has demonstrated a level of protection from kernel-level vulnerabilities, such as the ptrace vulnerability in Linux 2.4 kernels prior to 2.4.12.

The current implementation still struggles with the difficulty in transitioning from the Learning Stage to Production stage that is common with Rule-Based Anomaly detectors. Improvements have been made via the inclusion of a Learning output parser that facilitates the rule-base generation process.

Development continues, focusing on implementation survivability within a hostile environment, and improvements in the usability of the implementations.

StJude_SKM 0.01 (January 30h, 2002)

Saint Jude Solaris Kernel Module is the second in-kernel implementation of the Saint Jude Model. Currently versioned 0.01, it is in the early development stages. Currently the Model is running on the 64bit and 32bit SPARC versions of Solaris 8 and compiled with the Forte 6 C Compiler.

This version of the model contains the Override functions, and is compatible with the learning parser that was developed for the Linux kernel module. This version contains no defensive logic at this time.