Fighting Through An Attack

Computer Measurement Laboratory
info@cmlab.biz

There are really two distinct issues related to the notion of fighting through an attack or managing an attack. First there is the recognition phase. This occurs when a process is deemed to be operating outside of its normal or certified operational characteristics. The second phase is the management or containment of the attack in such a manner as to permit the continuation of the assault without endangering the processing environment or permitting intellectual property to be compromised. In response to this problem, we at CML have developed the CML Hardware-enabled Monitoring of Software (HMS) system. The primary objective of this technology is to create the infrastructure for an autonomic software process control system that is implemented in hardware.

We have developed an operational HMS system for the Linux PC environment. This is our Opus HMS system. It is designed to monitor and control the execution of the Linux kernel and any other executing software system running under the Linux kernel. The Opus HMS was designed to monitor the continuous operation of all executing software and insure that each software system was operating within a nominal, or certified, range of behavior. We have subsequently enhanced the Opus HMS system to the Opus μHMS system.

The philosophy behind the design of the Opus μHMS is the concept that program activity may be attributed to individual users of individual programs. This means that we are able to certify the execution of a program for each user of the system. This increases the resolving power of the HMS architecture to the point that the activity of *every user of the system may be monitored and controlled in real time*. This capability will provide the security administrator with the necessary tools to manage an attack in progress and also the capability of identifying the precise nature of the attack by an individual. This concept is particularly useful in controlling the potential of insiders to compromise a system.

There are essentially two types of attacks that can be deployed on a system: those that have been identified and classified from historical records and those that have never been seen before. Current approaches to software defense rely on the knowledge of the attack signature. That is, there are recognizable symptoms of the effect of the attack on the kernel. It is reasonable to expect, however, that an insider will be able to run well under the detection radar of traditional detection systems in that their use of the system will probably not be unconventional for the software. The use of the system will, however, be unconventional for that user.

Sessions

There are two distinctly different types of sessions that the Opus μHMS system must manage. There are users that have authorized access to the host Linux system and there are users that exploit system resources through their interaction with an Internet interface system such as Apache.

There are two ways that an authorized user might enter the system. They may be directly connected to a local Ethernet port (or serial port for that matter). They may also be remotely connected via `rlogin` or its modern replacement, `ssh`. In either event, the user will login to the system and will be known henceforth by the system by his/her unique user ID, UID. We will call these sessions UID sessions. They are characterized by the fact that the user has been authorized by a system authority to have access to certain system facilities. Typically, each UID is then bound to a particular group that, in turn, is known by its group ID, GID.

Each UID will be bound to a particular subset of programs on the system that they can execute either directly by UID or indirectly by GID. Not all users will have access to all executable programs on the system. Similarly, not all users will have access to all files on a system. These permissions are typically set by the local system administrator. It is the role of the Opus µHMS system to enforce and manage these permission sets.

There is another type of session that must be managed by Opus µHMS as well. This is a session that is initiated by a request for service from an IP client coming down the Ethernet wire. This type of Ethernet service will cause the Linux host to initiate an `http` daemon thread to manage this connection. In that the user from the outside, in this case, is not an authorized user, we must fabricate an artificial UID for this session. To this end, we will bind each IP session to a particular httpd thread that will be known by its thread ID, TID. On Linux systems, the TID and the PID will be one and the same. Thus we will use the PID-IP binding to identify distinct sessions that arise out of the ether.

The certificate process works slightly differently for each of these sessions. The Certificate Librarian in the Opus µHMS system will maintain execution certificates for each user on each system. For software that might be initiated on the PID-IP sessions, the only certificate will be the one for the process. We able to make this work in the Opus µHMS system by assigning a distinct UID for `httpd` sessions.

Containment

The basic notion behind containment is to be able to isolate and insulate each process from every other process running on a machine as a first step. When a process begins to operate outside of its certified execution space, the process should be totally isolated from the file system and other processes. There are three basic considerations that need to be addressed in the containment of a potentially threatening process. The process must be isolated from other processes in memory. The processes file accesses must not be permitted to alter the normal run time environment. And, the process must not be able to transmit information to areas outside of the containment region.

As an aside, processes using shared memory are a potential problem here. If the two (or more) processes sharing memory are running under the same UID, they will all be sandboxed. In the near term, if a process using shared memory is sandboxed while sharing memory with another (non-sandboxed) user, then the sandboxed process will be killed.

The first step in this isolation process is to prevent information leakage through memory accesses. In a potentially hostile environment, it is possible for a process to dump the contents of memory for the memory space assigned to the process by the Linux kernel. Given this very simple fact, it is prudent to initialize all memory assigned by the kernel to a process. To this end the `malloc` function in the kernel has been stubbed out and replaced with a call to `calloc`. This will insure that all memory given to any process is zeroed out before the active process receives it. No process will ever receive anything but zero filled memory from the Linux kernel.

The next step in the containment of a potentially divergent process is to remap the files that the program is using to a new root directory. We have used the `chroot()` command to do this. At the point where a user is deemed to be a threat, that user will have his/her root directory changed by the system to a new directory on the system called `/chroot`. Any files that are being accessed by the user will then be remapped to this new root directory. While there are definite limitations to this process, it adequately demonstrates the notion of file isolation for the purposes of the first project phase of this effort.

Finally, we would like to preserve the integrity of the intellectual property, IP, on the system. To this end a process that we have determined to be deviant must not be permitted to move information outside of the containment environment. While we have not implemented this feature during this phase of the project, it is certainly a very important aspect of the containment system.

In the Linux system, all devices are, in the last analysis, files written to by device drivers. To insure the containment of the IP we will need to create a new virtual environment for the process to live in. A very near term solution to this problem would be to simply redirect such program output to the special file /dev/null. Unfortunately, this strategy will not permit the recovery of information should the offending process be deemed to be normal after the fact. Another, strategy would be to hide information in the virtual environment as long as the UID remains sandboxed.

IP sessions may be managed in several different ways. At the point that an IP session is sandboxed, its priority may be lowed to keep the system running when fighting through an attack. This process may also be moved to a separate machine for a more closely monitored sessions. As a case of last resort, the operating system may be empowered to ban the IP address for a fixed period to allow time to analyze the nature of the assault.

Analysis and Observation

Once a process has crossed a critical threshold in its operating telemetry, the Opus μHMS will automatically transfer the user and any active user processes to the sandbox. The objective, at this point, is to understand exactly what the user is doing that is outside the range of certified behavior. There are two possible outcomes in this process. It might well be that a user is legitimately exercising the software in a manner dictated by their assigned duties. In other words, there is nothing intrinsically evil in the user's intent. In which case we would like to understand how the departure has occurred from the certified behavior to understand whether the user's certificate needs to be updated to include the newly discovered functionality.

A much more disturbing outcome, however, is that this particular user is up to no good. They are attempting to exploit potential weaknesses in the software system. In this case, we would like to have the ability to watch the user to determine the locus of the exploit that they seek to invoke. To that end, we employ the sandboxing technique to isolate the user so that their activity may be allowed to progress without loss of control.

Once a user has been sandboxed, we now wish to initiate the forensic activities to monitor the progress of the potential attack. This monitoring process will occur on several levels. First, a user may be attempting to initiate new processes for which he/she is not authorized. This will be represented by new nodes appearing on the process tree. The user may also be using an authorized software system in a non-certified manner. We will see the results of that with new subtrees emerging on the certified call tree representation. The user may also attempt to transmit information captured during the execution of a certificate violation. This information may contain valuable intellectual property that the user is attempting to capture and transmit out of the monitored environment. In which case, this information will be logged into a file in the sandbox environment but will not be transmitted to an external source.

Reintegration of Non-Malicious Processing

There is the distinct possibility that the user's activity, while appearing abnormal, is, in fact, part of their assigned duties and responsibilities. In the current sandboxing approach, once the user is isolated in a sandbox jail, all work that the user performs in the jail will be lost.

A very important aspect of the management of a potential attack is the ability to reintegrate the processing performed by a user that has been inappropriately assigned to the sandbox. This will entail updating all file activity that has occurred during the sandboxing interval together with communication with outside entities.

Controlling the Insider Threat Potential

To create a secure environment for the management of the insider threat potential is it first necessary to understand the nature of the threat. CERT has created a taxonomy of the nature of the insider threat. This taxonomy of nefarious user activity is summarized as follows:

Planted logic bomb while still employed
Created backdoors before termination or after being notified of termination
Disabled anti-virus on desktop & tested virus
Installed remote network administration tool
Downloaded and installed malicious code and tools
Downloading and use of hacker tools such as rootkits and password sniffers and crackers
Use of backdoor accounts
Set up new computer for remote access
Modification of logs to conceal malicious activity

In every on the of the above cases, it is clear that if the individual user, including the system administrator with root privileges, is monitored by the Opus μHMS system running under the aegis of an independent security administrator, it would be very difficult for any user to engage in any of the above activities without the system detecting this novel behavior.

From our perspective the whole notion of the insider threat potential can be minimized by a small set of very simple steps as follows:

- Build measurement infrastructure
- Monitor all user activity
  - Within process activity
  - Between process activity
- Trust no user
- Trust no process
- Monitor everything

The Opus µHMS system is the computational equivalent of a closed circuit television system with cameras covering every possible view. Every activity of every user is now visible to a central monitor and control system. Quite unlike a CCTV system, however, the Opus µHMS system is not passive. It may be empowered to isolate a user to determined the precise nature of the potentially malicious nature of the processes that he/she has initiated.