CHAPTER 7

DISCUSSION

Although the detection mechanism is both remarkably efficient and effective, it cannot be perfect in all respects.

The Detection Mechanism (DM) is useful in providing powerful security data for review, with its capability to determine which applications are running malicious software. It is expected that this database of applications will be extremely helpful in performing an audit of firewall rules and monitoring for policy compliance. By setting the anomaly detector, it is possible to quickly watch all the applications and locate vulnerabilities and security violations. Vulnerability assessment and anomaly detection will help the administrators to alert on or block threats. A defense center can aggregate information from all the sensors and this data can be used to secure the network centrally. Using the data gathered from the anomaly detector, the defense center can raise or lower the “relevance” of an event based upon the target’s importance and its susceptibility to the attack. In addition to a variety of alerting methods, administrators can block traffic via inline Intrusion Sensors, or via third-party firewalls, switches, and routers. They can also remedy vulnerabilities via third-party patch and configuration management solutions.

DM is implemented for offline processing of system call traces. It analyses system call traces and determines the anomalies during process executions. The current prototype is implemented in C in windows XP. It is not easily portable from windows.
Currently the training phase is separate and does not incorporate new sequences into normal profile.

Detailed discussions follow, along with comparison with other approaches.

7.1. SELECTION OF TRAINING SAMPLE

Before an anomaly detection system can be effective it must have a model of normal behavior. Normal profile is collected during training. The training data should be very carefully selected so as to contain all different categories of sequences. The next new code path may generate a dozen new sequences or none at all.

Perhaps the simplest way to ensure that Detection Mechanism (DM) has profiles of normal program behavior would be for software developers to distribute default profiles of normal program behavior. These synthetic normal profiles could be easily generated by running some or all of a program's regression test suite. If DM detects anomalous program behavior relative to such a profile when a program is being used properly, then the program's test suite is not comprehensive enough. Over a period, DM will replace many of these profiles with ones that are specialized to the usage patterns of a host. These profiles would generally be smaller than the default synthetic normal profiles and would restrict program behavior to those code paths that are actually used on a given machine.

7.2. EFFECT OF THRESHOLD

For example, a per-profile anomaly threshold could be tightened as a profile stabilized. In sequence-based methods, a change in user behavior has much effect on the correct threshold. But in this method, the effect of threshold is very steady. Zero threshold is a good choice since it considers all which deviates from normal as anomalous.
even if the variation is very small. To get an accurate performance, a threshold of zero can be chosen.

7.3. ADAPTING TO NEW SEQUENCES

Detection Mechanism could also be improved through the addition of a user space daemon to manage process profiles and regulate status responses. Such a daemon could note when new programs are run and use site-specific policies to determine whether it should be allowed or not. It could periodically scan the profiles directory for normal or almost-normal profiles that are likely to generate false positives.

The Mechanism never forgets program behavior even if a given behavior was encountered only once. To mitigate this limitation, the daemon could prune profiles to remove entries that hadn't been recently used. By correlating anomalies with network connections or log entries, a monitoring daemon could also decide whether a few scattered anomalies indicate that the system is under attack. It could then use this information to trigger a customized response.

7.4. PORTABILITY

DM is implemented in windows. With a few changes, the mechanism should also be able to run in Linux kernel. Because most UNIX variants use monolithic kernels and support similar system call interfaces, it should be straightforward to port core analysis routines to such systems.

DM captures the essence of a program's interactions with the outside world by observing its system calls. Hence it cannot be adopted on systems that do not support a traditional system call interface.
7.5. OFFLINE PROCESSING

When frequency component of system calls are used for analysis, the monitored process should terminate, in order to collect the final sequence of samples generated. So only the offline processing of system call traces is adopted.

With slight modifications, the method can be applied to on-line processing. The frequency components at each vulnerable point can be gathered and the same may be used for determination of anomalies.

7.6. LIMITATIONS

Similar to other monitoring approaches using system calls, the capability of the suggested approach is limited by the information contained in the system calls. The parameters passed to the system calls are also not considered. System calls contain information only about the limited events that occur. Only events that use system calls will appear in the system call sequence of a particular process. This restricts the capabilities of the monitoring system.

7.7. IMPROVEMENTS

Instead of capturing the behavior of entire applications, a better approach may be to monitor the execution of application components. Stillerman et al. [Stillerman 99] have shown that sequences of CORBA method invocations can be used to detect security violations; in a similar manner, it should be possible to detect security violations in Windows applications.

The violations that can be detected by the anomaly detector depend on the set of normal traces. The set of trace policies restrict the behavior of programs beyond that enforced by the protection mechanisms built in the operating system. Further research is
needed to identify a methodology for determining whether a given set of normal traces is adequate for a system.

Similar to other intrusion detection approaches, the new approach is not a panacea to the intrusion problem. Attacks that do not produce state changes (e.g., passive wiretapping) or that require massive behavioral analysis can not be detected. This approach also assumes integrity of the system call data. Thus, attacks that involves spoofing, which produce the same system call trace but from a different source, may not be detected.

Last, the new approach is a detection approach, which raises an alarm when an intrusion occurs. This approach can, at best, detect security violations, and it is up to the system administrator or security officer to deal with each detected violation.

7.8. COMPARISON

The proposed solution is a simple, accurate and systematic approach compared to current approaches for monitoring the security of a system.

7.8.1. Misuse Detection

In misuse detection, the goal is to identify actions (or misuse signatures) that represent intrusive activities and to check for occurrences of these actions in the audit trails. Expert system rules, state-transition diagrams, and patterns in Petri networks describe misuse signatures.

The specification-based approach can be thought of as the dual of misuse detection. A misuse signature describes undesired behavior in a system while a trace policy describes the desirable behavior of a subject. In particular, new approach focuses
on the desirable behavior of security-critical programs (e.g., privileged programs) in a system. One way to specify the desirable behavior of a program is to enumerate the operations the program needs to perform in order to accomplish its function.

A misuse detector matches a signature with the whole system trace to identify intrusions while an analyzer in a specification-based execution monitor parses the trace of a subject to determine whether the subject conforms to a trace policy. Although matching of different signatures can be distributed over multiple hosts, each misuse detector requires the whole system trace. In a distributed system with many hosts, the whole system trace would be huge and cannot be processed by a misuse detector in real time. In the suggested approach, an analyzer monitors the execution of a particular subject; only the audit records associated with the subject are needed by the analyzer.

In misuse detection, signatures are mostly driven by previous attacks or known vulnerabilities. Although possible, it is not intuitive to encode a policy as misuse signatures. The new approach is more policy-oriented; a trace policy for a subject is specified based on the functionality of the subject and the system security policy. Therefore, it can succeed in catching attacks that exploit unknown vulnerabilities in programs.

7.8.2. System Call Based Methods

When compared with the widely used fixed-length contiguous subsequence models, the system call representation explored in this dissertation is somewhat simple. It may be argued that much more sophisticated models are available that take into account the identity of the user or perhaps the order in which the calls were made. But the experiments show that a much simpler approach may be adequate in many scenarios. The
results of experiments described in the previous sections show that it is possible to achieve near perfect detection rates and false positive rates using a data representation that discards the relationship between system call and originating process as well as the sequence structure of the calls within the traces.

It is possible to achieve accurate anomaly detection using fixed-length contiguous subsequence representation of input data. In this approach, the detector will find anomalous subsequences right after they are executed depending on user-specified thresholds. A weakness of the subsequence based approach [Somayaji 98] is that small changes in user behavior can result in very different patterns of system calls. In general the rate of novel sequences goes down; yet for all programs, there are discontinuities when usage patterns change. A profile that has "almost settled down" is not "almost stable"; the appearance of even a few novel sequences means that previously unseen code paths are being executed.

The proposed model has advantages that learning is faster, memory requirements are significantly lower and simple Bayesian model discriminates normal sequences and abnormal sequences.

7.9. CONCLUSION

This thesis describes a new approach to security monitoring. The main idea is to detect buffer overrun vulnerabilities by specifying the desirable behavior of security-critical programs in a system and to monitor their executions for behavior inconsistent with the specifications.

The approach aims at building process profiles with system call frequencies and to detect anomalies by measuring deviations from the process profile. Aspects of program
behavior that are security-relevant were identified and a number of model parameters for describing the desirable behavior of programs were developed. The detection model uses novel ideas from probability, and provides a way to enhance accuracy of detection. The method is demonstrated by applying it to a simple vulnerable process and sendmail.

The model is very suitable for detecting well-known attacks and trivially modified attacks under time and space constraints. If the anomaly detector needs to be built in real-time and the built system must be as light as possible to be able to work over limited resources, the new approach will be a perfect fit because the generated detector is simple and powerful to detect well known attacks.

Since no effective mechanism has been determined for dynamic detection and prevention of buffer overflows, obtaining information about the occurrence of buffer overflow itself is very important. The suggested method performs offline analysis of system call traces to detect presence of anomalies due to buffer overflows. Taking into account the security relevant information such as the time of attack, the function which is used for attack, the user who executed the vulnerable program and the attack code used, the system administrator can take precautionary measures to prevent further attacks, once the anomaly is signaled. The anomaly detector presented, is a security mechanism that moves us a step closer to a self- defending system.