A Study in the Feasibility of Performing Host-based Anomaly Detection on Windows NT*

Aaron Schwartzbard and Anup K. Ghosh
aschwart@rstcorp.com,anup.ghosh@computer.org

Reliable Software Technologies Corporation
21351 Ridgetop Circle, #400, Dulles, VA 20166
phone: (703) 404-9293, fax: (703) 404-9295
http://www.rstcorp.com

Abstract

Windows NT has become the dominant desktop platform. To date, host-based intrusion detection research has focused on Unix-flavored platforms. As a result, we have a large gap between the platform people use in practice and the platforms on which intrusion detection research is active. In this paper, we examine the feasibility of applying host-based intrusion detection to the Windows NT platform. Specifically, we are interested in applying anomaly detection algorithms to Windows NT processes in order to detect novel attacks against these systems. We describe our previous experiences in program-based anomaly detection on Sun Microsystem's Solaris platform and describe an adaptation of this technique to the Windows NT platform. We describe the relevant issues in performing program-based anomaly detection on the Windows NT platform and the auditing facilities available on the platform for supporting this approach.

1 Introduction

Attacks against computer systems have been rising dramatically in recent years. The Computer Security Institute (CSI) and the Federal Bureau of Investigation (FBI) surveys corporations, financial institutions, and government agencies annually. In the 1999 CSI/FBI survey, they found that the number of organizations reporting attacks through external Internet connections rose from 37% in 1996 to 57% in 1998. Interest in computer security attacks have risen from the domain of obscure electronic bulletin board discussions to the front pages of national newspapers, as the Internet has risen from obscurity to the backbone

*This work is sponsored under the Defense Advanced Research Projects Agency (DARPA) Contract F30602-97-C-0117. THE VIEWS AND CONCLUSIONS CONTAINED IN THIS DOCUMENT ARE THOSE OF THE AUTHORS AND SHOULD NOT BE INTERPRETED AS REPRESENTING THE OFFICIAL POLICIES, EITHER EXPRESSED OR IMPLIED, OF THE DEFENSE ADVANCED RESEARCH PROJECTS AGENCY OR THE U.S. GOVERNMENT.

Appeared in *Proceedings of the 2nd International Workshop on Recent Advances in Intrusion Detection*, West Lafayette, IN, USA. Sept. 7-9, 1999.

of government and corporate communications. In spite of the magnitude of the problem, today's technology is largely reactive. That is, current intrusion detection technology can only react to an intrusion once it is widely spread and known. By the time intrusion detection tools are updated, much damage may have occurred and defense assets may have been compromised.

Current intrusion detection technology is reactive because of its reliance on "attack signatures". That is, intrusion detection systems know when an attack is occurring by looking for signatures of known attacks in the audit data. This approach is a fairly reliable method for detecting known attacks and can provide low rates of false alarms; however, it is not capable of detecting novel attacks against systems, and even variants of known attacks. The key problem that the research community must address is how to detect novel attacks against systems.

Another significant issue is the gap between the platforms that current host-based intrusion detection technologies support and the machines that are actually used in practice. Today, host-based intrusion detection technologies are available for the Unix platform, with Sun Microsystem's Solaris operating system being the most popularly supported platform. However, the market has shifted radically in recent years from Unix-based workstations to Windows 32-bit (Win32) platforms, such as the Windows 95/NT/2000/CE platforms. This shift has resulted in homogeneity in desktop machines to be mostly Win32-based. Unfortunately, the intrusion detection technology has not followed suit, due to in large part the reluctance of the research community to adapt its research to the Win32 platform. While there are commercial intrusion detection products that run on the Windows platform, they are strictly signature based, meaning novel attacks will not be detected. In the next section we describe the salient differences between network and host-based intrusion detection systems.

In this paper, we address the "platform gap" by describing an approach for performing host-based intrusion detection on the Windows NT platform. We are interested in developing an intrusion detection technology that is capable of detecting novel attacks against the Windows NT platform.

Results from a recent U.S. Defense Advanced Research Projects Agency (DARPA) study highlight the strengths and weaknesses of current research approaches to intrusion detection. The DARPA scientific study is the first of its kind to provide independent third party evaluation of intrusion detection tools against such a large corpus of data. The findings from this study indicate that a fundamental paradigm shift in intrusion detection research is necessary to provide reasonable levels of detection against *novel attacks* and even variations of known attacks. Central to this goal is the ability to generalize from previously observed behavior to recognize future unseen, but similar behavior.

One of the largest challenges for today's intrusion detection tools is being able to generalize from previously observed behavior (normal or malicious) to recognize similar future behavior. This problem is acute for attack-signature–based detection approaches, but also plagues anomaly detection tools that must be able to recognize future normal behavior that is not identical to past observed behavior, in order to reduce false positive rates.

2 Differentiating Intrusion Detection Systems

Intrusion detection systems (IDSs) can be differentiated along two axes: (1) host- or network–based, and (2) anomaly or misuse detection. The types of data processed by host- or network–based IDSs is quite different in nature, and as a result, the types of attacks that are detected by either technique will be different in general. Network-based IDSs sniff the network for packets — primarily TCP/IP packets. These packets are then processed by the IDS to detect intrusions using either misuse or anomaly detection algorithms.

Host-based intrusion detection systems analyze data captured on the machine host. Host-based monitors audit people or software processes. The data captured can be user data such as keystrokes, login/logout times, operational profiles, or programs run during a session. Or the data can be program behavior data such as system calls or internal program states of monitored programs. Today, many commercial intrusion detection systems work on network data rather than host-based data. It is often possible to use much fewer monitors for network-based intrusion detection than for host-based intrusion detection.

Even with the state-of-the-art being heavily network-centric, three factors are making network-based intrusion detection systems a less desirable approach: (1) bandwidth in local area networks is growing at a staggering rate, (2) end-to-end encryption, and (3) switched Ethernet systems. The rate at which bandwidth on local area networks is growing, where 100 Mbit networks are going to be commonplace, is far beyond the peak performance rate at which network-based ID systems can process data. End-to-end encryption is being rolled out in next generation Internet protocols that can render network-based intrusion detection systems obsolete. That is, ID systems whose *modus operandi* is to sniff network packets between host A and host B, will be incapable of processing encrypted data. Finally, switched Ethernet systems require at least as many network monitors as a host-based system, eliminating the benefit of fewer network monitors. Further, switched ethernet eliminates the ability of network monitors to recognize attacks across a network, such as a subnet scan. For these reasons, we believe host-based intrusion detection systems will become increasingly important, particularly for the dominant platforms, such as Windows NT.

The second differentiating factor between intrusion detection systems is the type of algorithm used for detection. The two general types of intrusion detection algorithms are misuse detection and anomaly detection. Misuse detection systems are those that fall into the category of "signature-based" systems. That is, they work by comparing data captured online with signatures of known attacks stored in a database. As mentioned earlier, the advantage of this approach is the high rate of certainty in detecting well-known attacks and the low rate of false positives. The major drawback of misuse detection systems, however, is that they cannot detect novel attacks, or sometimes, even slight variants of well-known attacks. As a result, misuse detection systems are completely reactive to computer misuse. All misuse detection systems must be updated often to be able to detect the latest well-known attacks against systems.

The other type of algorithm commonly used for intrusion detection is known as anomaly detection. Anomaly detection techniques directly address the problem of detecting novel attacks against systems. This is possible because anomaly detection techniques do not scan for specific patterns, but instead compare current activities against statistical models of past behavior. Any activity sufficiently deviant from the model will be flagged as anomalous, and

hence considered as a possible attack.

Though anomaly detection approaches are powerful in that they can detect novel attacks, they have their drawbacks as well. For instance, one clear drawback of anomaly detection is its inability to identify the specific type of attack that is occurring. However, probably the most significant disadvantage of anomaly detection approaches is the comparatively high rates of false alarm. Because any significant deviation from the baseline can be flagged as an intrusion, it is likely that non-intrusive behavior that falls outside the normal range will also be labeled as an intrusion — resulting in a false positive. In spite of the potential drawbacks of anomaly detection, having the ability to detect novel attacks makes anomaly detection a requisite if future, unknown, and novel attacks against computer systems are to be detected.

In the following we discuss performing anomaly detection for processes running on the Windows NT platform. The goal of this work is to be able to detect novel attacks against the Windows NT platform by detecting significant deviations from normal program behavior. The approach leverages the base object auditing facilities of the Windows NT platform. Before laying out the approach, we describe our previous work in program-based intrusion detection and its success on the Solaris platform.

3 Program-Based Intrusion Detection

The premise of program-based intrusion detection is that there are certain behaviors that are characteristic (or uncharacteristic) of individual programs. Further, as different programs vary in functionality, so should they vary in exhibited behaviors. For example, the behavior of the program `tar` — which normally is non-interactive and affects files — is markedly different from that of the program `lynx` — which normally is very interactive but does not affect files. Every non-trivial program has some externally visible behavior. That behavior might be manifest as file I/O, requests for more memory, or any other action that requires computer resources.

Just as the behavior of a program exhibits distinguishing characteristics and traits at the user level, the behavior of a program also exhibits distinguishing characteristics and traits at the system level. The interactions between the operating system and a program that performs complex computations will be different from those between the operating system and program that performs extensive disk I/O.

Program-based (alternatively called process-based) anomaly detection requires building a profile for each monitored program. Two possible approaches to monitoring process behavior are: instrumenting programs to capture their internal states or instrumenting the operating system to capture external system calls made by a program. The latter option is more attractive in general because it does not require access to source code for instrumentation. As a result, analyzing external system calls can be applied to commercial off-the-shelf (COTS) software directly. Most modern day operating systems provide built-in instrumentation hooks for capturing a particular process's system calls. On Linux and other variants of Unix, the `strace(1)` program allows one to observe system calls made by a monitored process as well as their return values. On Sun Microsystem's Solaris operating system, the Basic Security Module (BSM) produces an event record for individual processes. BSM recognizes 243 built-in system signals that can be made by a process. Thus, on Unix systems,

there is good built-in support for tracing processes' externally observable behavior.

Most process-based intrusion detection tools are based on anomaly detection. A normal profile for program behavior is built during the training phase of the IDS by capturing the program's system calls during normal usage. During the detection phase, the profile of system calls captured during on-line usage is compared against the normal profile. If a significant deviation from the normal profile is noted, then an intrusion flag is raised.

On Unix systems, program-based anomaly detection has been demonstrated to be a viable option for recognizing malicious behavior as anomalous. In particular, system-level program-based anomaly detection has had great success. System-level program monitoring is performed by monitoring the interactions between a program and the operating system — not between the program and the user. Monitoring the interactions between a program and the operating system provides more information than monitoring interactions between a program and the user since all user interaction must go through the operating system. Thus, in a sense, system-level monitoring can capture user interactions in addition to actions that are normally transparent to the user (such as requesting more memory).

Unix operating systems are based on an imperative paradigm. Programs make requests of the operating system using system calls, and the operating system either performs the requested action and returns some indicator of success, or the operating system cannot perform the requested action, and returns an error code. Actions a program might request the operating system to perform include (but are not limited to) mapping a file to an area of memory (`mmap`), checking the attributes of a file (`stat`), creating a child process (`fork`), and opening a file descriptor (`open`).

Early work in process monitoring was pioneered by Stephanie Forrest's research group out of the University of New Mexico. This group uses the analogy of the human immune system to develop intrusion detection models for computer programs. As in the human immune system, the problem of anomaly detection can be characterized as the problem of distinguishing between self and dangerous non-self [Forrest et al., 1997]. Thus, the intrusion detection system needs to build an adequate profile of self behavior in order to detect dangerous behavior such as attacks. Using `strace(1)` on Linux, the UNM group analyzed short sequences of system calls made by programs to the operating system [Forrest et al., 1997].

More recently, a similar approach was employed by the RST research group in analyzing BSM data provided under the DARPA 1998 Intrusion Detection Evaluation program [Ghosh et al., 1999]. The study compiled normal behavior profiles for approximately 150 programs, a significant increase in scale over previous equality matching intrusion detection studies.

The results from the study showed a high rate of detection, if not a low false positive rate [Ghosh et al., 1999]. Despite the simplicity of the approach and the high levels of detection, there are two main drawbacks to the equality matching approach: (1) large tables of program behavior must be built for each program, and (2) the equality matching approach does not have the ability to generalize. The first problem becomes an issue of storage requirements for program behavior profiles and is also a function of the number of programs that must be monitored. The second problem is the inability of the equality matching algorithm to recognize behavior that is similar, but not identical to past behavior. The problem is that behavior that is normal, yet slightly different from past recorded behavior, will be recorded as anomalous. As a result, the false positive rate could be artificially elevated.

Instead, it is desirable to be able to recognize behaviors that are similar to normal observed behavior, but not necessarily identical to past normal behavior as normal. Likewise, the same can be said for a misuse detection system. Many misuse detection systems are trained to recognize attacks based on exact signatures. As a result, slight variations among a given attack can result in missed detections, leading to a lower detection rate. It is desirable for misuse detection systems to be able to generalize from past observed attacks to recognize future attacks that are similar.

4 Auditing Program Behavior On Windows NT

The Windows NT operating system is a relatively new enterprise platform. It has been gaining acceptance rapidly in recent years. Despite its growing position of wide deployment in a server capacity, and other highly connected domains, the availability of intrusion detection tools for Windows NT is sparse compared to the availability of such tools for Unix platforms.

Of the Windows NT based intrusion detections tool that exist, most are designed to perform network intrusion detection. While network intrusion detection is useful, due to higher bandwidth, end-to-end encryption, and switched networks, network intrusion detection is becoming less and less applicable to modern computing environments.

In order for a Windows NT intrusion detection tool to be viable in the long-run, it must be host based. Host based intrusion detection, such as the program based anomaly detection discussed above, can be a powerful tool for detecting misuse of a system.

One option for Windows NT host based intrusion detection tool development would be to port Unix intrusion detection systems to Windows NT. While such a solution might be feasible, it would certainly be both difficult and sub-optimal. If a person were to attempt to apply system call monitoring to Windows NT, the first issue that arises is that the vast majority of Windows NT system calls are not documented, and not accessible through the Win32 API. NT system calls are accessed by calls through the NTDLL.DLL library. While it is theoretically possible to wrap NTDLL.DLL in order to catch system calls, that would require knowledge concerning the nature of the system calls themselves, which is proprietary.

The second (and perhaps more important) issue that arises when confronting Windows NT intrusion detection is that the model on which Windows NT is based largely differs from that on which Unix is based. Whereas Unix is imperative (using system calls, a program requests that the operating system perform certain actions, and using signals, the operating system requests that a program perform certain actions), Windows NT is object oriented[1]. I/O operations are performed by the operating system giving an object corresponding to a specific resource to a program, and the process operates on that object. For example, a request for more memory is handled by passing a memory object to a process. While system calls (in the sense of Unix) do exist in Windows NT, operations performed on resources are accomplished using Windows NT objects.

Since, in essence, intrusions (or, more generally, security violations) amount to unauthorized accesses of resources, monitoring resource accesses is a reasonable approach to intrusion

[1]While it is true that Windows NT is not object oriented in the strictest sense, it is designed to allow computer resources to be treated and manipulated as objects.

detection. Further, while collecting system calls under Windows NT is a difficult task, the operating system provides the facilities to monitor object accesses.

4.1 Windows NT Base Objects

All resources are treated as objects on the Windows NT platform. However, not all objects are equal. On a Unix platform, some actions are visible to the user (*e.g.*, writing to a file), and some are transparent to the user (*e.g.*, mapping a file to memory). Similarly, on Windows NT, some objects, like files, are visible to users, whereas other objects, like sections (that is, a memory mapped portion of memory), are transparent. Those objects that are normally transparent are known as "Base Objects".

Normally, to audit objects, one must use the "Audit..." dialogue box from the "Policies" menu of the Windows NT User Manager. Simply selecting object auditing from the dialogue box will result in the system auditing only those objects that are normally visible to the user[2]. In order to audit base objects, the following registry key must be created:

Hive: HKEY_LOCAL_MACHINE\SYSTEM
Key: \CurrentControlSet\Control\Lsa
Name: AuditBaseObjects
Type: REG_DWORD
Value: 1

and the computer must be restarted. Once the computer has been restarted, auditing can be started by the normal method, and audit information concerning base objects will be recorded (as long as auditing of objects is enabled). The audit data containing base object information can then be processed for use in intrusion detection algorithms.

The objects that a process accesses under Windows NT, and the order in which the process accesses them, form a signature for a program. Just as `tar` and `lynx` have different characteristic patterns of actions performed under Unix, programs like `WINZIP` and `NETSCAPE` will have different characteristic patterns of object accesses under Windows NT.

5 Observations Concerning Object Monitoring

Object accesses under Windows NT may prove to be a rich source of audit information. Our intent is to use this information to detect program misuse. In order to do that, we must demonstrate that Windows NT object access data possesses two characteristics. The first characteristic that Windows NT object access data must possess is that a process trace must be *sufficiently* similar to other process traces from the same program. The term "sufficiently" may be somewhat ambiguous, but it is necessarily so. The degree to which process traces from one program must be similar depend on the method used to measure similarity. If one were to use a simple table lookup method, the tolerance for additional, or missing, object accesses in a stream of execution (that is, the *tolerance for variation*) might be very low. If,

[2]This is actually a simplification. Once object auditing is enabled, not all objects are necessarily audited. Some objects, such as printer devices and files stored on a hard disk, require the administrator to manually enable auditing for each object. Though this might seem like a daunting task for a volume that contains a large number of files, it is possible to recursively enable auditing from the top level of a directory tree.

on the other hand, one were to use a method that possesses the ability to generalize (such as neural networks), the tolerance for variation could be much higher.

The second characteristic that must be demonstrated to prove the viability of object access monitoring as a means of intrusion detection is that a misused, or subverted, program must differ *significantly* from normal use. Again, the term "significantly" may be somewhat ambiguous. The degree to which subverted programs must differ should be greater than the tolerance for variation of a given intrusion detection technique. In this case, techniques that have a low tolerance for variation have an advantage over those that have a high tolerance.

Ideally, for any given technique of intrusion detection, the monitored data should form a partition, with the boundary (corresponding to the tolerance for variance) between the set of traces of non-malicious executions of a program, and the set of malicious executions of a program. In such a situation, the intrusion detection system can be a perfect discriminater. Unfortunately, rarely is that the case. When performing process monitoring, the partition between malicious processes and non-malicious process is usually imperfect. An imperfect partition can results in both false positives (when the threshold for variance is too low), or false negatives (when the threshold for variance is too high).

Our preliminary investigations show that the first characteristic described above is possessed by Windows NT object access process traces. For a given program, traces from different executions tend to display a large degree of similarity to one another. We have also observed that different programs produce traces that are recognizably distinct. Although that does not prove that the second necessary characteristic is true of the data, it does offer some evidence that the trace of object accesses produced by a program is related to the functionality of the program. Since subversion of a program normally entails exploiting some rarely used or unintended functionality that exists within a program, there is a large possibility that the subversion will manifest itself in the process traces.

6 Process-Based Intrusion Detection On NT

The technique we intend to use to perform process-based intrusion detection on Windows NT is based on object accesses. Because we intend to perform process-based anomaly detection, we plan to adapt some of the techniques and algorithms we developed for process-based anomaly detection on the Solaris platform.

The first difference between process traces on Solaris and process traces on Windows NT is the format. On each platform, audit information is recorded in a format that is composed of a series of records. Each record corresponds to a single *event* (that is, a BSM event on Solaris, or some object access on Windows NT). Additionally, each record contains information such as the time that the event occurred, and parameters and attributes of the event itself (the parameters and attributes are highly dependent on the particular event). On Solaris, prior to applying the anomaly detection algorithm, the audit data must be stripped of much of the information contained in the records. Some of the information is completely discarded. The resulting format contains only process identification numbers and events. Events from different programs are de-interleaved and placed into files corresponding to the programs that created them.

On Windows NT, a similar form of preprocessing must occur. However, as the records of

the Windows NT audit data are in a different format, the preprocessing algorithm will have to be altered in structure (though not in nature). By taking advantage of the preprocessing stage, Windows NT data can be formated in such a way that minimal changes will need to be made to the anomaly detection algorithms as they are used on the Solaris platform.

There is a difference between Windows NT data and Solaris data that is perhaps more significant than differences in format. That is, the data collected on Windows NT concerns the passing of objects between the operating system and processes, whereas the data collected on Solaris concerns much more detail about actions performed with system resources. The effect that the difference in detail level remains to be evaluated emperically. The difference may require further alterations to the anomaly detection algorithms.

7 Conclusion

Despite the wide acceptance of Windows NT in the marketplace, there is paucity of intrusion detection tools aimed at Windows NT. Of those, very few qualify as host-based anomaly detectors. The importance of host-based anomaly detection will continue to grow as networks become less amenable to network-based intrusion detection, and as vulnerabilities, and exploits for those vulnerabilities, continue to be proliferated.

Blindly applying Unix intrusion detection techniques may not be appropriate for the Windows NT platform. Whereas monitoring system calls makes sense on Unix, it might not be optimal on Windows NT. Windows NT host-based intrusion detection should be designed with the Windows NT model in mind. By auditing objects (including base objects), an intrusion detection system can monitor the resources used by a process.

Hopefully, more attention will be given to Windows NT by intrusion detection researchers. Continuing to ignore this platform as it becomes more and more widely deployed is counterproductive to the goals of the computer security community. The future of computer security and intrusion detection must take Windows NT into account.

References

[Forrest et al., 1997] Forrest, S., Hofmeyr, S., and Somayaji, A. (1997). Computer immunology. *Communications of the ACM*, 40(10):88–96.

[Ghosh et al., 1999] Ghosh, A., Schwartzbard, A., and Schatz, M. (1999). Using program behavior profiles for intrusion detection. In *Proceedings of the SANS Intrusion Detection Workshop*. To appear.

Aaron Schwartzbard

Mr. Schwartzbard is a Research Associate at Reliable Software Technologies, where he is currently studying host-based intrusion detection techniques. Having received his degree from the University of Virginia in May of 1998, he has worked at Reliable Software Technologies for just over a year. His research interests include computer security and artificial intelligence.

Anup Ghosh

Dr. Anup K. Ghosh directs security research at RST, which includes projects in e-commerce security, information warfare and robustness testing, intrusion detection, malicious software detection, mobile code security, and information survivability. Dr. Ghosh is a principal investigator on research projects from the Advanced Technology Program (ATP) from the National Institute of Standards and Technology (NIST), the Defense Advanced Research Projects Agency (DARPA), the Air Force Research Laboratory, and other U.S. Dept. of Defense agencies. His work in information security is investigating novel methods for certifying component-based software for security, analyzing program behavior for intrusion detection, assessing the robustness of Windows software to unexpected and possibly malicious events, and detecting and preventing malicious behavior in executable code.