PROBE: A Process Behavior-Based Host Intrusion Prevention System

Minjin Kwon, Kyoochang Jeong, and Heejo Lee

Department of Computer Science and Engineering,
Korea University, Seoul 136-713, Korea
{mjkwon,kyoochang,heejo}@korea.ac.kr

Abstract. Attacks using vulnerabilities are considered nowadays a severe threat. Thus, a host needs a device that monitors system activities for malicious behaviors and blocks those activities to protect itself. In this paper, we introduce PROcess BEhavior (PROBE), which monitors processes running on a host to identify abnormal process behaviors. PROBE makes a process tree using only process creation relationship, and then it measures each edge weight to determine whether the invocation of each child process causes an abnormal behavior. PROBE has low processing overhead when compared with existing intrusion detections which use sequences of system calls. In the evaluation on a representative set of critical security vulnerabilities, PROBE shows desirable and practical intrusion prevention capabilities estimating that only 5% false-positive and 5% false-negative. Therefore, PROBE is a heuristic approach that can also detect unknown attacks, and it is not only light-weight but also accurate.

1 Introduction

According to a variety of attacks, security is a substantial issue in today's networks. Malicious users are attempting enormous methods to successfully disrupt a target system. Against them, many technologies such as firewalls, anti-virus programs, and intrusion detection systems (IDSs) have been used to keep networks and hosts safe. However, according to the advent of sophisticated attacks, we nowadays need a new method based on intrusion prevention systems (IPSs) to protect systems. Even though network-based IPS can block malicious traffic, some can pass through it [1]. Thus, host-based IPS plays an important role in the last line of defense.

To defend these attacks, we propose PROcess BEhavior (PROBE) which is a host-based intrusion prevention system that investigates system processes to identify abnormal process behaviors. By using it, intrusions which use remote exploits to infiltrate a system are detected because an attack is behaved out of common behavior. We present a novel characterization for process sequences of a system. This characterization is based on two observations: a process is always executed when a user wants to execute a program, and a process on an operating system runs in sequence. The previous researches [2,3,4] detect anomalous behavior of system programs by inspecting different system call sequences in comparison with normal patterns of short sequences of system calls. The approaches

L. Chen, Y. Mu, and W. Susilo (Eds.): ISPEC 2008, LNCS 4991, pp. 203–217, 2008.

to the solutions are not practical for preventing the vulnerabilities in the real world because of system call monitoring complexity. On the other hand, PROBE is designed to detect major security violations by monitoring just several process behaviors without tracing every system call triggered by running processes.

This paper's main contribution lies in detecting unknown attacks and advancing a practical mechanism for intrusion detection. Our approach detects any novel intrusions different from normal procedures without prior knowledge of the attack mechanism. We represent processes' relationship as a tree, and then perceive the execution of an abnormal process just by examining a process's parent process and its child process. Thus, PROBE has low process monitoring overhead and it is appropriate for adoption to protect a host from intrusions.

2 Related Work

Intrusion Detection Systems (IDSs) are designed to detect unwanted attack or manipulation of a computer system. However, they cannot stop traffic, only identify an attack as it occurs. On the contrary, Intrusion Prevention Systems (IPSs) not only detect an attack, but also block the attack. Thus, it is considered as a combination of both an IDS which has the power of detecting attacks and a firewall which has the power of filtering attacks. Researchers have followed two main directions in the investigation of techniques to identify attacks: anomaly detection based IDS and misuse detection based IDS.

Anomaly detection based IDS generates an alert when a behavior deviates far from the predefined normal behavior. There were studies of intrusion detection for anomaly detection[2,3,4] using short sequences of system calls of running processes. While this technique can detect unknown attacks, it unfortunately generates false-positive[1] problem. Thus, most IDS products in the market today use misuse detection instead of anomaly detection[1].

Misuse detection based IDS contains signature database which has typical patterns of exploits used by attackers. If the attack signature matches any of some predefined set of signatures, IDS raises an alarm. Autograph[5] is a Internet worm signature monitoring system. However, its main drawback is that one must know the signature of attack to detect intrusion, so it is difficult to detect unknown, novel attack. Polygraph[6] is a study about the automatic generation of signatures that match polymorphic worms and the detection of them. To evade the problem of signature database maintenance, the techniques which detect malware on the basis of its behaviors are being studied.

Behavior-based Spyware Detection[7] is one of the typical behavior-based malware detection techniques. While this technique can detect obfuscation transformations which can easily be evaded by signature-based techniques, it has high false-positive and false-negative[2] as compared with signature-based detection.

[1] False-positive represents a legitimate behavior that is incorrectly identified as a malicious behavior.

[2] False-negative denotes an abnormal behavior that is incorrectly identified as a legitimate behavior.

These detection techniques of IDSs focus on how an attack works, so it detects attacks after a system is already infected by malwares. To stop malicious behavior before it causes any harm, IPSs, in contrast, focus on what an attack does—its behavior[1]. Thus, we propose a dynamic detection system, PROBE, that uses normal and abnormal characteristics of process connection. PROBE does not require signatures for attacks as with misuse detection, nor monitoring every system calls as previous system call trace studies. Thus, we instead concentrate on the behavior of the system processes.

3 The PROBE Mechanism

Since it is difficult to find out all vulnerabilities of operating systems and to patch them, we need a technique that can control abnormal access to a system by discovering characteristics of process creation. Before we get down to details of a technique that is based on the characterization of process behavior, we should examine the execution procedure of normal boot processes closely to use a control technique which can regulate an abnormal access within boot processes. The majority of the actual intrusions do not follow normal system operations no matter which exploit was used [8]. Thus, we will investigate from boot processes running during the operating system startup, then explain a principle and control procedures of abnormal processes.

3.1 Windows NT Boot Process

A process is a container for a set of resources used when executing an instance of a program. As with other operating systems, Microsoft Windows system goes through an elaborate boot up process. The boot process has a series of sequential steps since the computer is powered on. By understanding the details of the boot processes, we can diagnose problems that can arise during a boot. Our objective is to make our host system safe against any intrusions since the host starts up. The tracing abnormal behavior of system processes can achieve an execution of safe system booting and correct booting procedure. Thus, we show the execution steps of boot support files and the information on what some of the system files are for. We can represent process execution sequences of an operating system as a tree design. Figure 1 illustrates process execution sequences of Microsoft Windows NT system. The sequence of Windows boot processes is all arranged beforehand until application programs are executed. Application processes will be corresponded to a leaf node which is a node that has zero child nodes. When considering the process sequence tree, we can see that the path from a root node to a leaf node has a regular pattern. If an unauthorized user accesses to a system using a system's vulnerabilities, the executed process sequence by the attacker has a different characteristic from the normal process execution sequence. Thus, we propose a model which can detect an abnormal execution of a process by analyzing process execution relationship from a root node to a leaf node using a process tree.

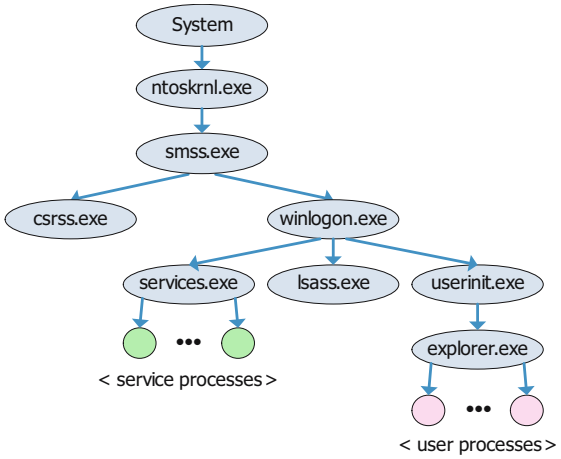


Fig. 1. The process tree of Microsoft Windows NT system

3.2 Design

The boot processes are executed in sequence during the booting steps of an operating system. However, it is possible that an adversary is able to exploit a system using specific bugs or vulnerabilities such as overflows during a boot. Since an attacker executes at least one process to invoke an abnormal behavior to overwhelm a system, a check of whether a process is executed by the operating system came from a normal process or a non-related process helps prevent attacks such as an overflowed buffer exploit. Thus, we design a host-based intrusion prevention system beginning with boot processes to execute the system securely. Our system, PROBE, takes a closer and deeper look at the activity of processes run on the host, calculates three weights on each edge based on relationship between a parent process and a child process, and determines acceptance or rejection of a process using the three weights. Therefore, PROBE protects desktops or servers by keeping operating systems securely from intrusions.

Process Information. To build a process tree and obtain the characteristics of each path, we need some information related to processes. PROBE utilizes the information which is provided by the operating system related to the processes. A number of tools for viewing processes and process information are available[9]: the tools included within Windows itself, Windows resource kits, and etc. The most widely used tool to examine process activity is Task Manager. We arranged these information into the six categories in Fig. 2.

3.3 Three-Phase PROBE Mechanism

PROBE inspects and detects abnormal behaviors by looking at processes within a host. To facilitate the understanding the relationship of processes, we present a process tree that shows parent and child connection between processes. For

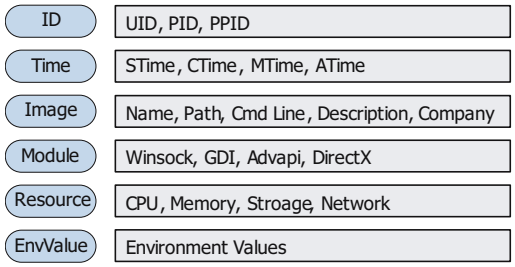


Fig. 2. Process Information which presents a unique process characteristic that can be gained from Windows

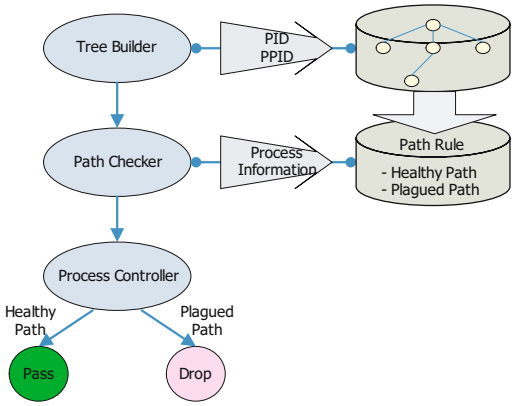


Fig. 3. The architecture of PBOBE that consists of three phases: Tree Builder, Path Checker, and Process Controller

doing this, PROBE works according to following three phases: *Tree Builder, Path Checker,* and *Process Controller*. We schematized the architecture of PROBE in Fig. 3.

A process behavior-based host intrusion prevention system starts from creating a tree structure. Tree Builder creates a process tree using PID and PPID of a process. After the process tree is created, Path Checker analyzes each tree path based on Healthy Path Rule and Plagued Path Rule. First, if the path is not determined as a Leaf-Node path which is a Plagued Path Rule, the process is a healthy path. Otherwise it is a suspicious process, so it needs additional steps. If there is a path which is decided as a healthy path by Healthy Path Rule among plagued paths, it becomes also a healthy path. Finally, Process Controller manages the process according to the prejudged decision. Healthy paths are executed and plagued paths are dropped in this phase. This procedure is illustrated in Fig. 4.

Tree Builder. Tree Builder monitors processes of a system, then creates a process tree from running processes on an operating system. A node in a tree is a

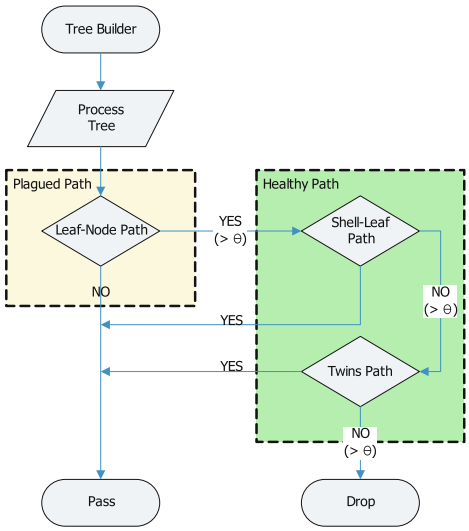


Fig. 4. The general algorithm of PBOBE. The θ threshold value represents a discriminator to distinguish a normal process and an abnormal process. Thus, the value of θ should be feasible for discriminating between a usual process and a unusual process. We set the value as 0.7 in our simulation.

running process and it has references to other nodes. To construct a tree, we use a process identifier which is a number used by an operating system kernel to uniquely identify one specific process. Using two process identifiers, one for a process's PID (Process ID) and the other for a process's PPID (Parent Process ID), it can create a tree based on a relationship between a parent process and a child process. The tree is created by first running system process with both child pointers null. Thus, the root node of a tree is the node with no parents. After the following process begins according to process steps required to boot a system, the additional node is created and inserted into the root node as child node to build a larger tree. Thus, a process which has a PID of a newly launched process creates a node. After checking its PPID, the process becomes a child node of its parent process node. The procedure of Tree Builder is described in Fig. 5.

Path Checker. Path Checker analyzes each directed edge of a tree which was made at Tree Builder phase and detects if something abnormal occurs according to Healthy Path Rule and Plagued Path Rule. For this, we use important information from operating system about system objects—attributes, modification time, etc. This Process Information is later used for checking the processes whether they are under the rule of Healthy Path or Plagued Path. Path Checker uses this Process Information to attempt to determine the intent of a process by catching the relationship with a parent process and a child process. PROBE can detect abnormal process execution just only examining local tree information of

```
Input: Process List
Output: Process Tree T = (V, E) where V is the set of vertices and E is the set of edges.

Tree Builder Algorithm
1  V ← ϕ;
2  E ← ϕ;
3  WHILE EnumProcess()
4        PID ← GetProcessID(); // PID of current process
5        PPID ← GetParentProcessID(); // PID of current process's parent
6        IF (V ∩ Node(PID) == ϕ
7              V ← V ∪ Node(PID);
8              E ← E ∪ Edge(PPID , PID);
9        END IF
10 END WHILE
END of Algorithm
```

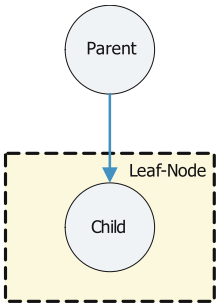
Fig. 5. The first phase of PROBE: Tree Builder

an edge between a parent node and a child node, which was connected by the creation procedure of a process, not all tree information.

- **Plagued Path:** Abnormal path. It defines the edge in the process tree which is constructed through Tree Builder phase when the process is executed by a suspicious behavior. We assigned Leaf-Node Path into the plagued path which has a probability of something abnormal behavior happening.
- **Healthy Path:** Normal path. It defines the edge in the process tree which is constructed through Tree Builder phase when the process is executed by an ordinary behavior. We assigned Shell-Leaf Path, the parent process serves as a shell process, and Twins Path, two related processes triggered from one program execution, into the healthy path.

Leaf-Node Path. Leaf-Node Path checks if the child process is an application process node. Each downward path from a root node to a leaf node is unique. In principle, processes from application programs are dangled at the bottommost level of the tree. Typically, most of application processes do not create any child processes except a process that carries out a function as a shell program. Leaf node application processes are only executed by a particular parent process such as a shell program. Figure 6 describes an algorithm that measures weight of Leaf-node path. Thus, if it is perceived that both a parent process and a child process are processes derived from any application programs, there might be a possibility that something abnormal such as execution of exploits using buffer overflow is happening. That is, it is considered as a doubtful activity that application process creates another application process.

Shell-Leaf Path. Shell-Leaf Path checks if the parent node which executes a child process node is a shell program process. A shell denotes not only the system-defined shell, Explorer.exe, but also a program which serves as a shell process. The way to determine normal behavior would be to monitor process relationships in execution. Figure 7 shows these relationships. To identify a shell process, sibling processes created from a same parent process are influenced to weight a value of Shell-Leaf Path.



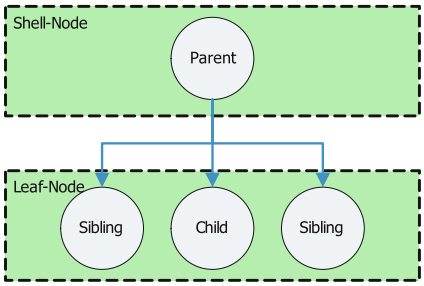
Input: Parent-Child Process Information
Output: Weight of Leaf-Node Path

Leaf-Node Path Algorithm

```
1  P ← Parent Process;
2  C ← Child Process;
3  W ← 0;
4  IF P.Winsock
5        W ← W + Weight(Winsock);
6  END IF
7  IF C.Winsock
8        W ← W + Weight(Winsock);
9  END IF
10 IF NOT C.GDI
11       W ← W + Weight(GDI);
12 END IF
13 IF C.Advapi
14       W ← W + Weight(Advapi);
15 END IF
16 IF W > 1
17       W ← 1;
18 END IF
19 RETURN W
```

END of Algorithm

Fig. 6. Leaf-Node Path, which is a plagued path, is considered to be a security risk if the behavior of process is related to any of intrusion's behaviors we defined as follows: 1) the parent process uses network services, 2) the child process uses network services, 3) the child process is not associated with Windows graphics, or 4) the child process utilizes advanced Windows API's.



Input: Parent-Child-Sibling Process Information
Output: Weight of Shell-Leaf Path

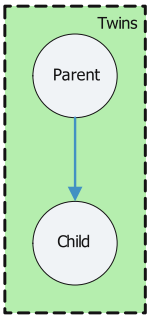
Shell-Leaf Path Algorithm

```
1  P ← Parent Process;
2  C ← Child Process;
3  S ← Sibling Process;
4  W ← 0;
5  IF P.Storage
6        W ← W + Weight(Storage);
7  END IF
8  IF P.GDI
9        W ← W + Weight(GDI);
10 END IF
11 W ← W - Leaf-Node(C);
12 WHILE every S
13       W ← W + (Leaf-Node(S) / NUM(S));
14 END WHILE
15 W ← 1 - W;
16 IF W > 1
17       W ← 1;
18 ELSE IF W < 0
19       W ← 0;
20 END IF
21 RETURN W
```

END of Algorithm

Fig. 7. Shell-Leaf Path which is a healthy path to check the parent process fills the role of characteristics of Windows shell

Twins Path. Twins Path checks a characteristic of closeness between two nodes. We named it Twins Path that a path between a parent process and a child



Input: Parent-Child Process Information
Output: Weight of Twins Path

Twins Path Algorithm
1 P ← Parent Process;
2 C ← Child Process;
3 W ← 0;
4 IF (C.CTime - P.CTime) <= δ
5 W ← W + Weight(CTime);
6 END IF
7 IF P.Path == C.Path
8 W ← W + Weight(Path);
9 END IF
10 IF P.Company == C.Company
11 W ← W + Weight(Company);
12 END IF
13 W ← 1 - W;
14 IF W < 0
15 W ← 0;
16 END IF
17 RETURN W
END of Algorithm

Fig. 8. Twins Path which is a healthy path to measure how much the parent process and the child process are related. The δ threshold value denotes time interval to measure the creation time difference of a parent process and a child process within a certain span.

process has a similar characteristic. When there is a similarity between processes, an application process can execute another application process. Thus, a process not owned by the requesting process will be blocked by this phase. This is a signal that an abnormal behavior is happening. To measure the similarity between a parent process and a child process, we use the Process Information such as process image creation time and company to reflect weight of similarity into Twins Path as shown in Fig. 8.

Process Controller. Process Controller regulates process execution according to whether the process path is healthy path or plagued path. When the process behavior of a system deviates far from the Path Rule, alerts are generated and the process cannot be executed.

4 Evaluation

Today attacks are often taking advantage of multiple vulnerabilities. In order to measure the effectiveness of PROBE, we need to develop a means to produce an accurate model of today's Internet security vulnerabilities. It has been tested using SANS Top-20 lists[10] released in recent two years, November 2006 and November 2005. SANS announces critical vulnerabilities, the most often exploited Internet security flaws, that led to worms like Blaster, Slammer, and Code Red every year since 2000. Among 40 vulnerabilities released in 2006 and 2005, there are 28 vulnerabilities (13 in 2006 and 15 in 2005) which are applicable to Windows system. Out of 28, we selected 16 remotely exploitable vulnerabilities

Table 1. The 16 remotely exploitable vulnerabilities in Windows system among critical Internet security vulnerabilities (SANS Top-20 lists) released by SANS Institute in 2006 and 2005

No.	Vulnerability	Release	Description
1	Internet Explorer	Nov. 2006 (v.7) W1 Nov. 2005 (v.6) W2	Vulnerabilities in ActiveX controls installed by software. Remote code execution
2	Windows Libraries	Nov. 2006 (v.7) W2 Nov. 2005 (v.6) W3	Vulnerabilities in Windows libraries. A remote attacker executes arbitrary code
3	Microsoft Office	Nov. 2006 (v.7) W3 Nov. 2005 (v.6) W4	Vulnerabilities in Microsoft Office applications (Outlook, Word, PowerPoint, Excel, Visio, Etc.)
4	Windows Services	Nov. 2006 (v.7) W4 Nov. 2005 (v.6) W1	Vulnerabilities in a wide variety of services. networking methods and technologies
5	Web Applications	Nov. 2006 (v.7) C1 Nov. 2005 (v.6) C3	PHP remote file include, SQL injection, cross-site scripting (XSS), cross-site request forgeries
6	Database Software	Nov. 2006 (v.7) C2 Nov. 2005 (v.6) C4	Use of default configurations, buffer overflows, SQL injection, use of weak passwords
7	P2P File Sharing Applications	Nov. 2006 (v.7) C3 Nov. 2005 (v.6) C5	Modifying legitimate files with malware, seeding malware files into shared directories
8	Media Players	Nov. 2006 (v.7) C5 Nov. 2005 (v.6) C7	A malicious webpage or a media file to compromise a user's system without interaction

on Windows system except for vulnerabilities overlapped or applied to a specific application. We assumed that 16 vulnerabilities to be arisen in our system. Those are shown in Table 1.

We simulated PROBE using vulnerability information on Microsoft Windows XP Professional SP2. Figure 9 presents a system process tree that shows processes used in our simulation. We need weight of Process Information such as Winsock, GDI, Advapi, Storage, CTime, Path, and Company to calculate weight of each edge between a parent process and a child process. We use these weights to calculate Leaf-Node Path, Shell-Leaf Path, and Twins Path in PROBE's 3 phases. In each phase, the weights of Process Information have a same value within its phase. By using these weights of Process Information, each process calculates three edge weights. The first one is Leaf-Node value. If its value is close to 1, the process has a high possibility that it is an abnormal process. The second value is Shell-Leaf. If a process relationship is Shell-Leaf, the value is close to 0. It means that an abnormal behavior of a process has 1. The third value is Twins. It checks how much the two processes are similar. If they are totally different, it has a value close to 1. Thus, if all three weight values exceed a predefined threshold, the process is regarded as an abnormal process. We can see the results of PROBE simulation in Fig. 10. All attacks were detected except for "attack 4" and it shows false-negative. This happens because the process used vulnerabilities in "services.exe". PROBE does not detect an intrusion in case of an attack which uses a vulnerability of a known shell. Also, there is a false-positive case. Because yahoo widget is not known as a process which roles as a shell, it causes a false alarm.

Therefore, PROBE detects most of intrusions which utilize remotely exploitable vulnerabilities except for attacks via normal processes. Thus, PROBE shows desirable host-based intrusion prevention capabilities. By investigating

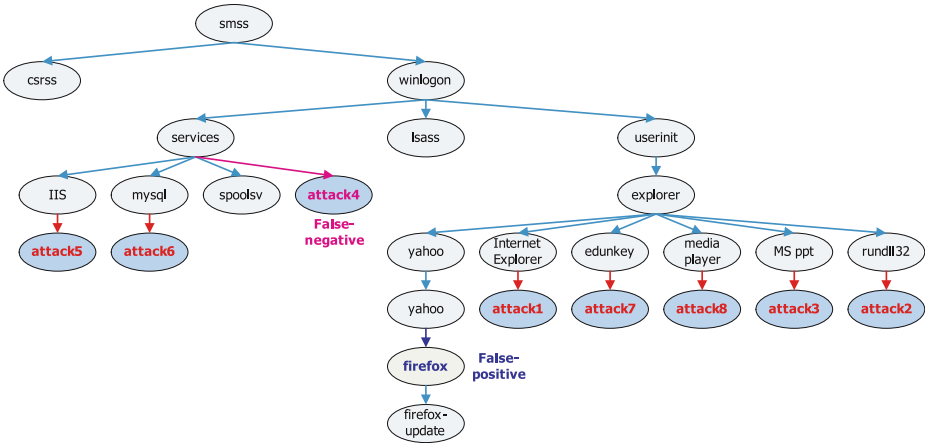


Fig. 9. A system process tree that shows process relationships used in our simulation

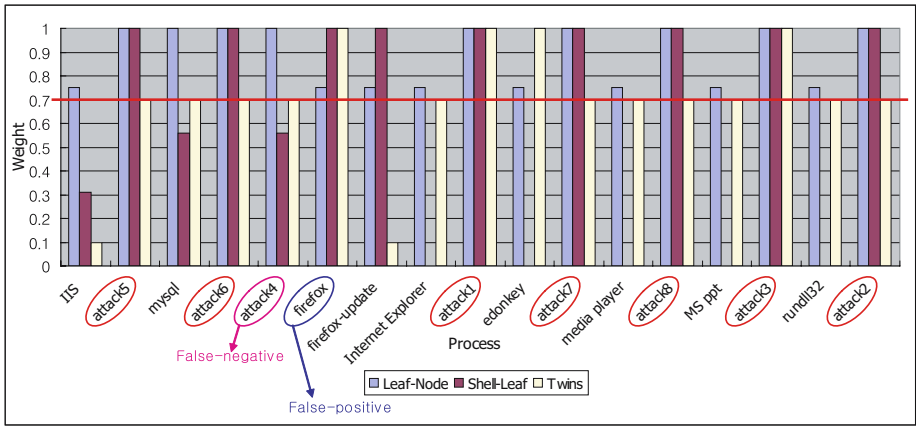


Fig. 10. The result of our simulation. Among 8 attack cases, 7 attacks are detected. There happened one false-negative case and one false-positive case.

more Healthy Path Rule to strengthen PROBE, we will be able to reduce false-positive and false-negative rate, and achieve a much better result than now.

5 Discussion

5.1 Benefits of PROBE

When an unauthorized user wants to have access to a system, the user utilizes bugs or design flaws in the system. The flaw or weakness in the system makes an opportunity to force it to conduct unintended operations by malicious users

and vandals. Especially, the unknown attacks occur when prevalent signatures of attacks are not able to identify the attacks. It takes a great deal of time until patches to be applied and signatures to be updated. Fighting the unknown attack is one of the greatest challenges facing the security industry today[11]. Thus, we need to adopt an anomaly detection approach to reduce a false-negative effect against unknown attacks.

Additionally, PROBE has very low process monitoring overhead and memory requirements. The existing studies[2,3,4] for anomaly detection need enormous traces of system calls which should be monitored to discriminate between normal and abnormal characteristics. In contrast to the earlier approach based on system call behavior (normal database size of sendmail-1318, lpr-198, and ftpd-1017, which is the unique sequence of system calls for each of the process to be stored in each process database)[4], we only use 7 process information to determine a normal process and an abnormal process. Thus, it is light-weight and practical solution for intrusion prevention. Also, Fig. 11 shows the PROBE's elapsed time of API calls in comparison with existing mechanisms. We can see that PROBE has the highest efficiency in processing by comparing the elapsed time of API calls. Our evaluation of PROBE demonstrates that it exhibits low process monitoring overhead and memory requirements. PROBE rapidly and efficiently detects novel attacks at exceedingly low memory and processing time. When compared with existing intrusion detection methods which use system call sequences, PROBE differs in that we use a much simpler way to detect intrusions. For a program, the theoretical sets of system call sequences will be huge. Complete attack prevention is not realistically attainable due to system complexity. Thus, we rely on examples of normal system process runs rather than normal databases of all unique sequences during traces of system calls. An advantage of our approach is that we do not have to build up the set of normal system call patterns for the database. We simply compare with predefined process creation rules by tracing processes of a system. Therefore, PROBE is an efficient solution for detecting intrusions.

5.2 PROBE Limitations and Future Work

PROBE is a network-based intrusion prevention system that uses system processes to detect attacks which use security vulnerabilities by looking for intrusions performing programs without passing through legal process creation procedure. System damage due to an attack is caused by running programs that execute system processes. Thus, we restrict our attention to running system processes. Intrusions are detected when a process behaves out of its characteristics of normal system processes. The Path Rule between processes is defined to represent the ongoing behavior of the process creation.

It is important that intrusion detection systems are capable of detecting attacks against the Windows NT operating systems because of its growing importance in government and commercial environments[12]. There are lists of Windows NT attacks developed for the 1999 DARPA Intrusion Detection Evaluation[12]. These attacks categorize with groupings of the possible attack

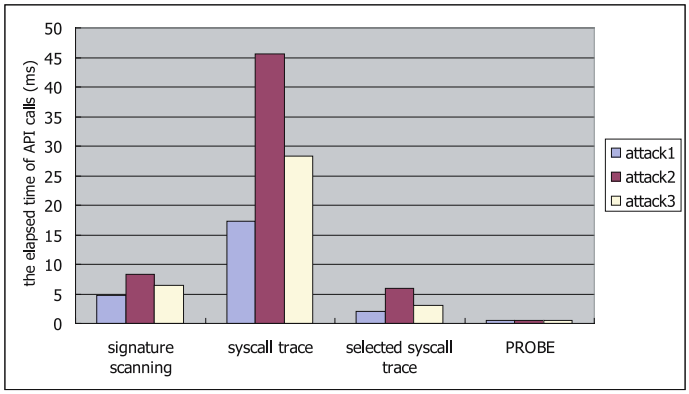


Fig. 11. We measured the elapsed time of API calls (milliseconds). PROBE is compared with existing intrusion detection mechanisms: signature scanning (misuse detection), syscall trace (system call traces), and selected syscall trace (partial system call traces). We tested under 3 attack environments: attack1 (Remote-to-User attack which establishes backdoors), attack2 (Remote-to-User attack which is a trojan horse that allows a remote attacker to control a system), and attack3 (Remote-to-User attack which uses a buffer overrun vulnerability). Under these attacks, we can see that PROBE executes intrusion detection at a low processing time.

types: Denial-of-Service, Remote-to-User, and User-to-Super-user. Table 2 shows a description of each attack category and document the individual Windows NT attacks in each category. These attacks spawn processes which deviate from normal process execution. PROBE can detect intrusions which use system or software bugs or exploits such as vulnerabilities in the victim computer or trojan programs to establish backdoors on the victim system. Most attacks create a new process which deviates from normal process behavior even in case of adding a user to penetrate into the system by exploiting a vulnerability on a system. Thus, it does not pass through normal process creation procedure.

The major problem is that system's behaviors change with time. As a result, the system's behavior can deviate more from the Path Rule initially determined. To solve the problem of false-negative by the unknown attacks, we followed anomaly detection approach. However, it has a limitation to discriminate between attack and non-attack. To protect against newly discovered attacks, we

Table 2. Windows NT Attacks Developed for the 1999 DARPA Intrusion Detection Evaluation

Attack Category	Attack Name
Denial-of-Service	CrashIIS, DoSNuke
Remote-to-User	Netbus, NetCat, PPMacro
User-to-Super-user	CaseSen, NTFSDOS, SecHole, Yaga

need more Path Rule to evolve. Certain intrusions which can only be detectable by examining other aspects of a process's behavior, and so we might need to consider them later. Future work will focus on extending PROBE to find other abuses of privilege and to find an error in configuration of a system. Also, we intend to expand our base of intrusions and gather more data for more processes running in real environments, so we can get more realistic and accurate estimates of false-positive and false-negative.

6 Conclusions

Our system, PROBE is a host-based intrusion prevention system which is installed on a particular host and detect attacks targeted to that host only. For the purpose of protecting a system against host-based attacks, we proposed process behavior-based protection approach. PROBE is designed to detect security vulnerabilities in a host without monitoring every system calls. It only finds out the characteristics of process relationship between a parent process and a child process. Thus, it can detect unknown attacks by judging a behavior of process creation relationship. Also, it is a light-weight solution, and shows practical and accurate result for intrusion prevention. Our approach was evaluated on a test set of SANS Top-20 Internet Security Attack Targets. The results demonstrate that our approach can effectively identify the behavior of abnormal processes. Future work will focus on extending PROBE to reduce false-positive by analyzing process relationships and find out more accurate results. By using PROBE, we expect to secure our systems against unknown system vulnerabilities of new kinds of exploits.

Acknowledgments

This work was supported by Defense Acquisition Program Administration and Agency for Defense Development under the contract UD060048AD and the ITRC program of the Korea Ministry of Information & Communications.

References

1. Sequeira, D.: Intrusion Prevention Systems: Security's Silver Bullet? In: Business Communications Review (March 2003)
2. Forrest, S., Longstaff, T.A.: A Sense of Self for Unix Processes. In: IEEE Symposium on Security and Privacy, pp. 120–128 (1996)
3. Forrest, S., Hofmeyr, S.A., Somayaji, A.: Computer Immunology. Communications of the ACM 40, 88–96 (1997)
4. Hofmeyr, S.A., Forrest, S., Somayaji, A.: Intrusion Detection using Sequences of System Calls. Journal of Computer Security 6, 151–180 (1998)
5. Kim, H.A., Karp, B.: Autograph: Toward Automated, Distributed Worm Signature Detection. In: Proceedings of the 13th Usenix Security Symposium (August 2004)

6. Newsome, J., Karp, B., Song, D.: Polygraph: Automatically Generating Signatures for Polymorphic Worms. In: IEEE Security and Privacy Symposium (May 2005)
7. Kirda, E., Kruegel, C., Banks, G., Vigna, G., Kemmerer, R.A.: Behavior-based Spyware Detection. In: 15th Usenix Security Symposium (August 2006)
8. Cunningham, R.K., Lippmann, R.P., Webster, S.E.: Detecting and Displaying Novel Computer Attacks with Macroscope. IEEE Transactions on Systems, Man and Cybernetics (July 2001)
9. Russinovich, M.E., Solomon, D.A.: Microsoft Windows Internals. 4 edn., Microsoft Press (December 2004)
10. SANS: SANS Top20 Lists (November 2006), `http://www.sans.org/top20/`
11. Henry, P.A.: Day zero threat mitigation, Seminar: Fighting the Unknown Attack (May 2006), `http://www.pisa.org.hk/event/fighting-unknown-attack.htm`
12. Korba, J.: Windows NT Attacks for the Evaluation of Intrusion Detection Systems (June 2000)