HARDENING WINDOWS SYSTEMS

George Davida
P.O. Box 784
Department of Electrical Engineering and Computer Science
University of Wisconsin - Milwaukee
Milwaukee, WI 53201

Jon Peccarelli
Department of Electrical Engineering and Computer Science
University of Wisconsin - Milwaukee
Milwaukee, WI 53201

Keywords: Operating system security, intrusion detection, vulnerability assessment, restricted tokens.

Abstract: Microsoft Windows systems have been the favourite target of viruses, worms and spyware in recent years. Through a kernel level process, we can secure Windows™processes by running them in a form of a sandbox. These sandboxes are configured to only allow a subset of rights to be granted to the process, therefore limiting the ability of a rogue process or a compromised process to inflict damage on the rest of the system.

1 INTRODUCTION

Microsoft Windows™has been a target of viruses, worms, spyware and other malicious code due to a history of insecure systems. In recent years, a concerted effort has been made to secure Windows™. The software firewall has been enabled by default. Hotfixes have been made available for any security related issues once they are discovered. Microsoft has released an anti-spyware product to combat the spyware issue. And they have published whitepapers on hardening the operating system through security, account and local policies. Yet the operating system is still venerable to malicious processes that run in the user context. Many problems with security breaches, viruses, worms and spyware can be traced to systems that allow too many rights to processes.

Therefore, the focus is to control the bounds of a process. In other words, put the process in a sandbox. However, these sandboxes would work differently than the typical sandbox. This sandbox could then be modified by a master kernel level process to grant specific rights approved by the user, administrator or some other external, trusted source. Processes that were not predefined would run in a default sandbox with very limited rights, thus limiting the risk of running, either knowingly or unknowingly, untrusted processes. This would limit the ability of a virus in an e-mail to be able to infect, or install, itself on the host computer, since the process would be unknown and therefore run in a limited sandbox. For example, spyware would be blocked from installing on the host system since it would not have rights to write to the disk and registry.

A benefit of this approach is that system performance remains nearly unaffected by this method of protection, since there is not a constantly running process watching the other processes on the system, unlike the typical sandbox. When a process is created, its security context is adjusted. The process may take a few more clock cycles to start, but this should be fairly invisible to user in terms of performance degradation. This is in stark contrast to anti-virus software which is constantly running and scanning every process and file that is accessed on the system, and to other past attempts that are constantly intercepting all system calls.

2 SECURING PROCESSES

Security in Windows™(NT,2000,XP) is handled by security identifiers (SIDs), tokens and access checks at many different levels. Files, devices, volumes, and registry keys are all examples of items that can be protected. This list also includes processes. Access tokens are utilized by Windows™to handle system security. The security reference monitor (SRM) is responsible for the definition and use of these access tokens. (Russinovich and Solomon, 2005)

User-level processes typically inherit the rights of

Davida G. and Peccarelli J. (2005).
HARDENING WINDOWS SYSTEMS.
In *Proceedings of the Second International Conference on e-Business and Telecommunication Networks*, pages 368-371
DOI: 10.5220/0001416703680371

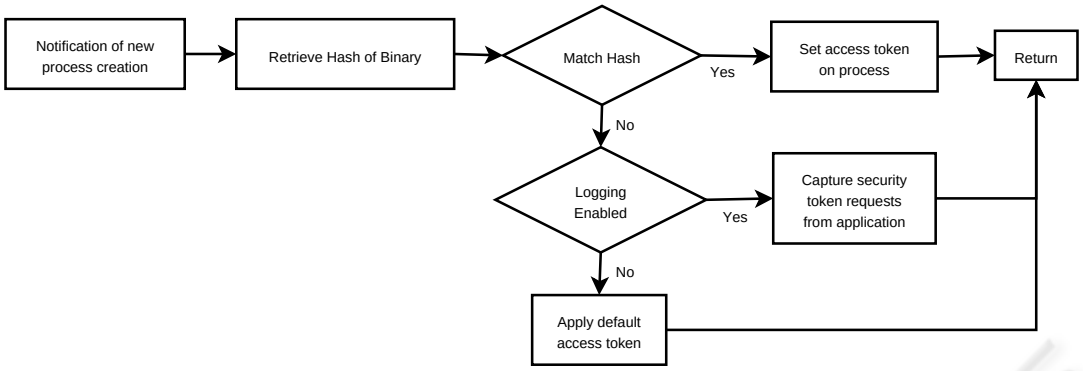

Figure 1: Filter Kernel Driver

the main user-mode process (userinit.exe). The main user-level process is originally created during the logon process for the user. (Russinovich and Solomon, 2005) Since this user-level process has the same rights as the logged on user, any program launched under this user will have these rights. There is an inherit problem with this blanket approach. Processes that the user may not want to launch, or may be unaware of, can be run under the users security context, namely viruses and spyware. It would be nice to have a way to control the access token each process inherited, and even grant limited security rights to processes not known by the user and/or system. By utilizing restricted tokens in Windows™, we can create processes with limited privileges and limited access control lists.

2.1 Related Work

Many techniques have been developed to secure processes above the standard operating system security.

In a client-server architecture, execution management was controlled by system administrators using an execution management utility (EMU). In this scheme, each user has a unique execution control list (ECL) which is essentially a database of programs the user is allowed to execute. Identification of programs is accomplished by the MD5 hash of the executable binary and the executable name. Through the use of a kernel driver, all new process creation is controlled. (Schmid et al., 2001)

Microsoft added a technology called group policy to its Active Directory environments. An administrator can control which executables are able to run on a computer through a software restriction policy. (Trent, 2004) Programs are either allowed to run or not, with no change to security of the running process. This technology also requires a computer to be part of an Active Directory environment, therefore home computers are not able to use a software restriction policy.

The idea of a sandbox is to control and monitor a user-level process, through a set of instructions from a configuration or policy file. A sandbox runs as a process wrapper, intercepting all system calls from the ”wrapped” process. The sandbox either allows or denies the system call based on a configuration file. This can be used to limit access to the entire filesystem, for example, by presenting a subdirectory in the filesystem as the root of the filesystem to the ”wrapped” process. Janus was developed to monitor a user-level process and deny harmful system calls. (Goldberg et al., 1996) Janus is an example of a sandbox.

Kernel hypervisors are a layer of software that provides a set of ”virtual” system calls. These can be used to provide fine-grained security control, for example. (Mitchem et al., 1997)

2.2 Methodology

The EMU and Microsoft approaches are an all-or-nothing scheme, i.e. a user is either allowed or denied the ability to run an application. Once an application is allowed to run, no additional security is provided. The sandbox and hypervisors are processes that run and constantly monitor, and possibly deny, system calls and requests. Our approach is to modify the security token of the process before execution and leverage all of the granularity and security of the operating system.

The following list is the basic steps for creating a process via the CreateProcess function. (Russinovich and Solomon, 2005)

1. Open the image file to be executed.
2. Create the executive process object.
3. Create the initial thread.

4. Notify the subsystem of the new process so that it can set up for the new process and thread.
5. Start execution of the initial thread
6. In the context of the new process and thread, complete the initialization of the address space and begin execution of the program.

The item of interest in this list is step two, creating the executive process object. During this step, a copy of the parent's access token is made, which becomes the new process's access token. Since this token grants access under the user's security context, it is the key to hardening the system.

The first step to controlling the security context of processes is to take control of the process creation process. This is accomplished through a kernel driver. The idea is to capture the process creation process, locate the proper security context for the process and execute the process with the appropriate access token.

For this whole process to work correctly and securely, the security credentials need to be stored in a manner that prevents, and in the worst case detects, tampering. The security context for each process are stored in a Execution Control List database. This information is stored in a format that is encrypted and digitally signed for security and authentication purposes. The kernel driver is the only process that is allowed to write to thi database.

An important point to note is that this concept, by itself, cannot successfully secure a system. When combined with proper techniques already available to secure a Windows™system, this concept can yield a much more secure system. This technique also requires the use of an access-based filesystem, like NTFS, to work properly. If write access cannot be restricted to the ECL database, the kernel driver will be able to identify a compromised system through an invalid digital signature.

This entire methodology is related to the idea of running each and every process on the computer under a unique user id. Each process would have a security context that would be inherited from a unique user id. However, there are some concerns using this methodology. First, how does one process access shared information from another process, since they are running under different user contexts? And secondly, how does one manage such a large number of user accounts, especially in the example of multiple users running on one system (i.e. Terminal Services)?

2.3 Process Identification

For this technique to function properly, a method needs to be used to accurately identify processes. This method needs to be able to determine if a malicious process is trying to run as a known validated process. For example, assume notepad.exe has previously been identified, validated and has a ECL entry in the database. Now a malicious process names itself notepad.exe. This process should not be allowed to run under the original notepad.exe security context. Therefore, a method of process identification needs to developed to ensure that the proper security context is assigned to the intended process only.

A digital signature or hash will be created from the process's executable image. This signature will then be stored in the database to identify the process. Before a process is allowed to execute, it's signature will be compared to those in the database. If a match is found, the access token associated with that signature will be used. Otherwise, the process will be handled as an unknown process as described in Section 2.5. If an executable file is updated through a service pack, patch or upgrade, the digital signature will need to be updated as well, otherwise it could be denied the ability to execute properly.

2.4 Extracting an ECL

Two approaches can be utilized to create an Execution Control List for a process. The first approach is to start the process without any rights and add rights as needed. This approach is very secure, however it requires an in-depth knowledge of the process in question. Since most Windows™programs are closed source, this is usually not possible by the end-user, unless a trial and error approach is attempted. The developers would have the knowledge required to create an ECL for each process as described in Section 2.6 .

The second approach is to run the process with a monitor in place. The kernel driver can log all security token requests of the monitored process. Once this has been completed, the ECL information can be incorporated into the ECL database. From this point, the user can just use this ECL as is or make modifications to the ECL. This allows the user to limit the rights of the process without generating this list from scratch.

2.5 Default Execution Control List

What happens when a process is launched that doesn't have an Execution Control List associated with it? If the process were allowed to run without an ECL, this would be a breach of the hardening of the system. Therefore, an approach needs to be developed to eliminate this potential threat.

When a process is created that does not have an ECL associated with it, the process is created with a default access token. This token can be defined with predetermined access rights. These predetermined access rights could deny all rights to the system - thereby eliminating the ability of any unknown

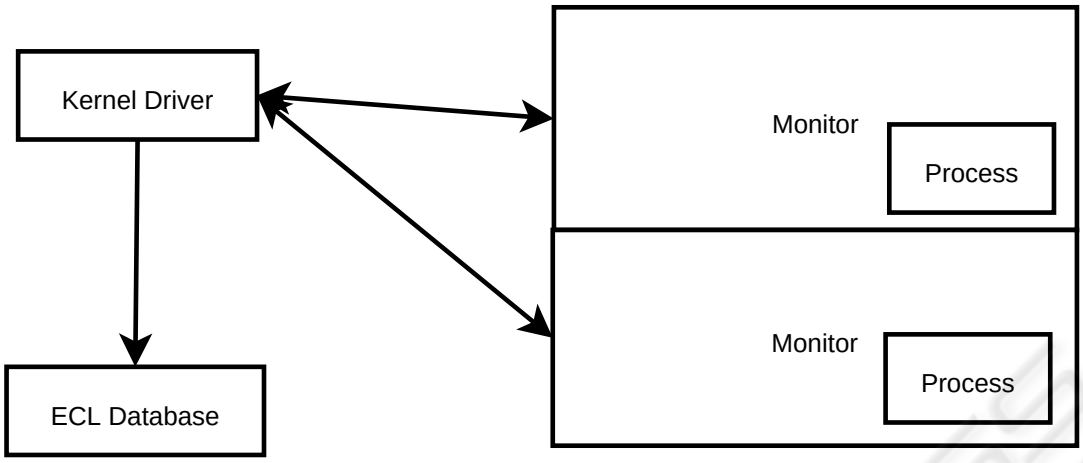

Figure 2: ECL Extraction

process from executing on the system. The kernel driver can also be configured to automatically log the token access requests and start building an ECL for the new process. This would allow the user to incorporate this information in the system-wide ECL database. Multiple default Execution Control Lists could be defined and utilized depending on the process's parent.

2.6 Future Use

Execution Control Lists could be delivered with an application installation. The developers would be responsible for identifying the appropriate rights required for each process in the application to work properly. Currently, many Windows™developers assume the application will have full access to the entire system, which is a large security problem. This would slow, or maybe even stop, that practice.

For tighter security, incorporation into kernel would be the next logical step. Therefore, a callback would not have to be registered to catch process creation. The process would be created with the proper security token directly from the operating system, instead of catching the process creation and modifying the initial process access token. The most logical place for this incorporation would be the security reference monitor (SRM).

A central execution management server could be developed to control an Execution Control List for a corporation, thereby limiting the exposure of individual corporate computers to spyware, viruses and other malicious executables.

REFERENCES

Goldberg, I., Wagner, D., Thomas, R., and Brewer, E. A. (1996). A secure environment for untrusted helper applications. In *Proceedings of the 6th Usenix Security Symposium*, San Jose, CA, USA.

Mitchem, T., Lu, R., and O'Brien, R. (1997). Using kernel hypervisors to secure applications. *Proceedings of the 13th Annual Computer Security Applications Conference (ACSAC'97)*.

Russinovich, M. and Solomon, D. (2005). *Microsoft Windows Internals*. Microsoft Press, Redmond, 4th edition.

Schmid, M., Hill, F., and Ghosh, A. K. (2001). Preventing the execution of unauthorised win32 applications. In *Proceedings of the 2001 DARPA Information Survivability Conference and Exposition*, Anaheim, CA.

Trent, R. (2004). Using software restriction policies to block spyware-adware.