# Execution Control Lists: An Approach to Defending Against New and Unknown Malicious Software

Anup K. Ghosh & Matt Schmid
Cigital
21351 Ridgetop Circle, #400, Dulles, VA 20166
phone: (703) 404-9293, fax: (703) 404-9295
email: {aghosh,mschmid}@cigital.com
www.cigital.com

**Keywords:** execution control lists, malicious software, viruses.

## 1 The Problem of Malicious Software

A study released by InformationWeek Research and PricewaterhouseCoopers in July, 2000 estimates that the cost of malicious software, or malware for short, will exceed US $1.5 trillion worldwide this year.[1] The impact on US businesses with more than 1000 employees is estimated to be $266 billion or approximately 2.7% of the Gross Domestic Product (GDP). Most of the costs were estimated to be in lost productivity as a result of downtime and in lost sales opportunities. The study estimates that approximately 40,000 person years of productivity will be lost worldwide this year alone.

Today's approaches to dealing with malware are not effective. The most common approach to dealing with viruses is to use a signature-based virus detection tool. Unfortunately, this approach will not detect the next variation of the same attack. Because these viruses spread so quickly, the reactive approach to virus detection is not effective in stopping a Melissa-style virus. Other approaches such as filtering content at the firewall have proved to be largely ineffective in the past and will be rendered obsolete in the future by end-to-end encryption.

In the following, we describe a novel approach to addressing malicious software by controlling its ability to execute. We omit implementation details for brevity. The approach has been developed into a working software prototype.

## 2 Controlling Malicious Software Execution

One of the key ideas in containing malicious software is to prevent its execution and proliferation. Malicious software finds many ways to spread itself. Traditionally, viruses were passed from BBS systems and public shareware to individual machines. Typically, these viruses infected boot sectors on hard drives and floppies. Macro viruses changed the nature of propagation by spreading via shared documents. With pervasive Internet access, viruses now find ways of spreading via chat rooms, newsgroups, Web pages, shared software, distributed multimedia shared files, email, and even novel forms of software distribution such as Napster and Gnutella. Even so, malware still spreads in old-fashioned ways such as shared physical media and network drives. Due to the multiplicity of ways through which users receive software, today's system administrators have little control over the programs that their users download and run on their machines.

To address the problem of malicious software execution and proliferation, we developed an execution management utility that effectively locks down a Windows 32-bit (Win32) platform to a set of approved (or trusted) programs and prohibits execution of all other programs. Our approach centers on the notion

[1] Global survey of 4,900 information technology professionals across 30 nations conducted by InformationWeek Research and fielded by PricewaterhouseCoopers LLP, released July 10, 2000.

of execution control lists (ECLs), which are implemented by a loadable Windows NT kernel module that is capable of selectively intercepting process creation requests. This approach addresses the specific threat of unknown binary executables compiled for the Win32 platform. It does not address threats from script-based attacks, Java applets, or other code technologies that run within interpreters. Furthermore, our approach does not address threats from trusted code, such as Trojan horses that run within a trusted computing base. However, unlike current virus protection technology, our approach is powerful enough to prevent future and unknown malicious executables from executing.

The first approach that comes to mind when providing protection against malicious code is sandboxing applications. A number of researchers have explored the use of application sandboxes as a technique for restricting the actions of untrusted applications. The basic idea behind a sandbox is to specify the resources that an application can access and the type of access that the application is granted. If the application tries to access a restricted resource, the sandbox interferes and rejects the access attempt. Researchers at the University of California at Berkeley developed the Janus prototype that sandboxes applications running on a Solaris operating system [1]. The sandbox operates by intercepting system calls, and deciding to allow or deny them based on the parameters to the call. The decision-making logic is contained in a configuration file that is specified for each executable that is sandboxed. Now, a number of commercial sandboxing products exist.

Our goal, however, is different from sandboxing. We are not concerned with controlling the behavior of programs, but rather, about controlling which programs are allowed to execute. Thus our approach addresses the software execution and proliferation problem discussed in the preceding section. Another approach to malicious software that has been taken is to periodically check file integrity, ala Tripwire. The idea is that one can detect intrusions after the fact by frequently checking the integrity of critical system files or key target programs. However, this approach will not necessarily detect the presence of new software.

## 2.1 The Execution Management Utility

The principal idea behind the execution management utility (EMU) is to restrict a user to executing only an approved and known set of applications. This set typically includes all the operating system software necessary to run application software, operating system services, and a set of desktop applications necessary for the user to perform his or her duties. In addition to the base set of software that many users or groups of users will have in common, individual users may have different software needs. For example, an administrative assistant may need access to a different set of applications than an accountant requires. The EMU provides a client-server architecture that enables an administrator to have complete control over the applications available to any user working on an NT workstation with the EMU client installed.

Why place the execution management utility under the control of system administrators? Bruce Schneier points out that under the best of conditions, users cannot make good security decisions, let alone under the conditions today in which malicious code finds various ways to install and execute on users' machines [2]. Thus, rather than deploying a solution that counts on the user to make security decisions, the EMU gives that responsibility to an appropriate authority, such as a security officer or system administrator who is responsible for the integrity of the organization's computing infrastructure.

By giving administrators the ability to restrict the applications that users can access, the EMU can protect an organization against a wide range of threats. Running email attachments or downloading executables from the Internet is a common way for a user to damage a system. The threats posed by such executables range from the intentionally malicious virus or Trojan horse to the well-meaning but poorly behaving shareware with which we are all familiar. The damage caused by such executables is made more difficult to identify and repair because the system administrator often does not know what applications an employee may have run.

## 2.2 The Execution Control List

Central to execution management is the Execution Control List or ECL. Each user on the system has his or her own ECL that specifies which programs the user is allowed to execute. The combination of the executable name along with an MD5 hash of its binary executable uniquely identifies the application. Identifying a program through its hash ensures that a user cannot bypass the EMU simply by changing the name of an unknown executable to the name of an executable that is already part of the ECL. Using hashes for identification has the additional benefit of preventing modified applications from executing. Many viruses propagate by silently attaching themselves to an application and then executing whenever the application is run. If a virus alters a program in any way, then its MD5 hash will be changed. Therefore, the EMU ensures that modified executables will not run, effectively protecting the system from any damage these altered applications can do, as well as protecting against unknown malware.

### 2.3 Discussion

Although executables make up one major category of malicious programs, macro viruses and other interpreted forms of mobile code (e.g., Java applets, VBscripts, VBA macros, and Javascripts) also pose great dangers to a system. Because interpreted languages execute through another application (the interpreter), the Execution Management Utility cannot discriminate among various interpreted scripts. Thus, we do not assert protection against malicious scripts.

The only protection the EMU does provide with respect to malicious scripts is that the EMU does enable an administrator to decide which interpreters a user should be able to access. The EMU can prevent the interpreter itself from running, thereby preventing all scripts depending on that interpreter from executing. Unfortunately this is often not a practical solution in today's workplace. For example, disabling the Windows Scripting Host is effective in stopping VBScript email attachments (such as the LoveLetter virus) from running. However, it also disables a key enterprise computing technology, thus stopping forward progress.[2] While we do not address the mobile code security problem, our approach will work hand-in-hand with other third party mobile code protection technologies.

In its current incarnation, the EMU only validates executables that the operating system runs, not dynamically loaded libraries (DLLs). An executable often relies on both third party DLLs and system DLLs for a great deal of its processing. If a user were able to replace a DLL used by a trusted application with a compatible rogue DLL, then the code contained in that rogue DLL would be allowed to execute. Furthermore, application extensions (commonly known as plug-ins) may arrive as DLLs, not as separate executables.

In an effort to address this issue, we are currently experimenting with ways of validating DLLs as they are loaded into memory. Our kernel wrapping technique can be used to intercept the loading of DLLs and to perform run-time checks of these files before they are mapped into memory. DLL validation introduces some new questions and complications that we are currently working out. For example, should a DLL ECL be associated with an application or with a user? Future implementations of the Execution Management Utility should address these issues.

## 3 Conclusions

The execution management utility provides security-conscious administrators with a valuable defense against the introduction of new and potentially malicious code. It ensures that no user can execute a program (knowingly or unknowingly) without the explicit consent of the administrator. This greatly reduces the threat posed by viruses, Trojan horses, and even malicious insiders. In addition to security considerations, the EMU also provides control over the distribution of illegally-obtained applications and the use of entertainment programs on corporate resources.

There are numerous EMU features that make it manageable at the enterprise level, including centrally-managed execution control lists and client-server communication. The kernel based wrapping approach ensures the non-bypassability of the EMU and results in negligible performance overhead. Security features incorporated in the EMU ensure that even a malicious adversary cannot circumvent the execution management utility's execution control lists.

It is important to note that our approach is not a substitute for security protection technologies such as malicious code scanners and sandboxing techniques. Rather, our approach can work together with other techniques. For instance, if a user receives an unknown executable it will be denied execution by the EMU client. The user may then request an administrator permission to run the unknown executable. A system administrator can scan the code against a virus scanner or malicious code classifier. Furthermore,

[2] We address the scripting problem in a separate paper submitted to this workshop.

the administrator can choose to run the unknown executable within a sandboxed environment in order to determine if it is trustworthy. If the administrator decides the application can be trusted, then she may subsequently add the executable to the user's ECL to allow future execution. The important protection our approach provides is a first line of defense against users running unknown and possibly malicious executables.

Although malicious software inspires our approach, our solution also addresses other problem domains. The execution management utility can assist corporations by enforcing policies regarding the use of unauthorized, unlicensed or pirated software. Games and entertainment software, or even non-standard utilities (*e.g.*, Napster, Gnutella, monitoring utilities like SpectorSoft, eBlaster, and advertising-enhanced browsers such AllAdvantage.com), can be banned or carefully controlled.

Next, we list some tangible benefits of the EMU for IT management of the enterprise. The EMU will non-bypassably:

- stop unknown and potentially malicious software from being executed,
- warn administrators or audit when unknown and possibly malicious code is attempting to execute,
- allow corporations to be in compliance with their software license agreements,
- prevent unauthorized software such as personal software, games, and other forms of entertainment, from being run, and
- identify when a permitted software program has been corrupted or infected with unknown viruses.

In summary, we are providing a kernel-level protection mechanism to protect systems against compromise by future and unknown software.

In today's Internet-enabled work environment, it is essential that security administrators have full control over the programs that are executed by their user community. The EMU's ability to control which programs are allowed to execute is a significant step toward maintaining a secure networked environment.

## Acknowledgment

We acknowledge the contributions of Frank Hill and J.T. Bloch in developing the EMU prototype.

## 4 About the Authors

Dr. Anup K. Ghosh is Director of Security Research at Cigital. He directs research in the areas of malicious software, mobile code security, intrusion detection, and e-commerce security. He is author of *E-Commerce Security: Weak Links, Best Defenses* (Wiley, 1998).

Mr. Matthew Schmid is a Research Associate with Cigital and leads the malicious software research project described here.

## References

[1] I. Goldberg, D. Wagner, R. Thomas, and E.A. Brewer. A secure environment for untrusted helper applications: Confining the wiley hacker. In *Proceedings of the 1996 Usenix Security Symposium*. USENIX, July 22-25 1996.

[2] B. Schneier. The trojan horse race. *Communications of the ACM*, 42(9), September 1999.