Solitary Confinement: Using Artificial Cells to Protect Computer Systems

Jeff Gilchrist

Elytra Enterprises Inc.

Ottawa, Canada

jeff@gilchrist.ca

Abstract

A security framework called Solitary Confinement (SC), using virtual machines to divide a computer system into small disposable units is described. A mixture of traditional computer security with ideas inspired by biology and the immune system are used for the design of the framework, which reduces the amount of damage malicious and buggy software can inflict on a computer system. By constructing artificial cells using virtual machines and self-protecting components, resources on the system can be restricted and malware can be prevented from spreading, limiting the damage it can cause. Since the cells are disposable, any that become infected can be shut down in a controlled manner as not to interfere with or damage other artificial cells or the system as a whole. The paper presents the rationale and design of the security framework.

KEY WORDS

Security Framework, Malicious Software, Virtual Machine, Immune System, Artificial Cell, Solitary Confinement

1 Introduction

One aspect of computer security that has become increasingly dangerous is malicious software. With so many computers being connected to the Internet, viruses and worms can quickly spread around the world in a matter of seconds, infecting vulnerable machines before humans even have time to react [14, 13]. Trojans and Spyware are bundled with intriguing and popular software offered on websites and P2P networks. Many times, users are not even aware that they have been infected. Attackers can use buffer overflows to turn benign software into a malicious form [16]. This paper describes Solitary Confinement, a security framework for reducing the amount of damage that malicious and buggy software can carry out on a computer system. It uses some traditional security methods and borrows some ideas from biology and immunology to achieve this goal.

The human body is divided into many compartments which provide robust security against intruders. These small independent compartments called cells are the basic building blocks of the human "system". Each cell controls what may enter and exit its membranes, keeping the internal organelles protected [18, 23]. Individual cells are disposable, so the death of one cell does not affect the entire person. Humans live in an environment where we are constantly being attacked by intruders such as viruses, bacteria, and other organisms, yet the majority of us can survive these attacks for many decades. People don't need to download monthly security patches for themselves since our bodies have adapted to live in such harsh environments. We can look at biology to give us ideas on improving the survivability of our computer systems [22].

This strategy of having many independent compartments can be implemented on computers by arranging the system as a bunch of small disposable cells. Dangerous things that happen inside the cell are prevented from spreading. One way to create artificial cells in a computer system is by using virtual machines with self-protecting components. Each cell is an autonomous virtual machine that contains, monitors, and controls one or more processes inside. The cells not only isolate the processes but will also monitor them to detect and react to malware.

The remainder of the paper is organized as follows. Section 2 provides a brief introduction to virtual machines, cells, and apoptosis. Section 3 describes the Solitary Confinement framework and section 4 concludes the paper.

2 Background

2.1 Virtual Machines

A virtual machine is software that exports and controls or emulates the hardware of a computer system. This allows multiple operating systems to run on a single machine at the same time. Systems like VMware [25] allow a user running Linux to launch Windows XP while continuing to run Linux at the same time. Heavy weight virtual machines like

VMware that emulate the entire hardware including BIOS incur a performance penalty, in some cases running 25% to 50% slower than the native system. Other lighter weight virtual machines such as Xen [4] require small modifications to the OS and only suffer a loss in performance of a few percent.

The operating system and processes running inside a virtual machine think they alone are running on the hardware. They have no way of directly accessing the memory or processes of other virtual machines on the system and in most cases do not even know they exist. Additional lightweight virtual machines such as the exokernel [5] and recursive virtual machines [7] allow a separate high performance virtual machine to be created for each process. Since all processes running inside a virtual machine must go through it to access hardware and other resources, the virtual machine has ultimate control over each process.

2.2 Cells

Virtual machines in the Solitary Confinement framework are similar to eukaryotic (animal) cells in biology. They act as a cellular membrane, regulating what data and communications may pass through the membrane. Like cells they are semi-permeable, allowing some things to pass and not others [18]. But unlike cells, virtual machines also have the ability to become completely impermeable if necessary.

One important security feature of cells is apoptosis, the programmed death of a cell. Apoptosis is used to destroy cells that may be a threat to the organism such as cells infected with a virus, cells with DNA damage, and some cancerous cells [11]. The main benefit of apoptosis is that cells can be disposed of without causing harm or stress to other cells in the area. Cells that die by other means could allow their contents to leak out into the surrounding area, inducing an immune response that might be disruptive for other cells and tissue. The orderly destruction of a cell with apoptosis minimizes such disruption [24]. Apoptosis can be used with artificial cells to remove those that have been infected with malware while keeping the rest of the computer system and other artificial cells running undisrupted.

3 Framework Design

The Solitary Confinement (SC) framework will be described at a high level since different security algorithms and strategies can be used depending on the level of protection required. Similar to the immune system, multiple defense strategies are used and newly developed strategies or algorithms can be added and old ones removed to change or increase the coverage of protection over time.

SC uses several principles of the immune system to aid with computer security. Each virtual machine (artificial cell) is autonomous and does not require coordination from a central location to determine if it has been infected or react. Multiple layers of security mechanisms and monitors can be used within an artificial cell. Individual virtual machines in the system can be eliminated without adversely affecting the entire system [22]. Most computer security systems try to prevent an attack from occurring and do not consider what should happen in the event a machine is compromised. Biology on the other hand, shows us that if components of an organism are independent and do not heavily rely on others, they can be disposed of and a new component generated with no harm to the organism as a whole.

With current computer systems, the OS has to perform a lot of work to try and keep processes separated such as with permissions for execute, read, write, keeping memory separated, and the use of jails like chroot. Even all these complex mechanisms are not enough to stop malicious code from wreaking havoc on a system and possibly crashing the whole machine if a process can elevate its privileges through some vulnerability. Security best practices [19] recommend that every common network service such as a firewall, mail, and database each be installed and run on their own server, which can be expensive. The SC framework goes one step further to create an artificial cell for every process running on a computer system. Since each cell is a virtual machine, the processes run inside their own operating system and are isolated from those running in other cells. These small disposable units each contain their own self-protecting components that can induce apoptosis and recreate a new cell if infected or damaged without taking down the whole system with it. If malicious software started executing inside a cell, it would be in a contained environment, prevented from spreading. The framework allows multiple network services to run in separate protected environments on the same machine thus conforming to best practices and reducing costs at the same time.

When a process is launched on a system protected with SC, a new virtual machine is created, a specified amount of memory is reserved, and the process is started inside. This allows the SC cell to monitor everything the process is doing and act as a cellular membrane, giving SC full control over what resources the process can access in case any problems are detected. While memory limitations can be placed on processes in operating systems, the artificial cell provides a cleaner division guaranteeing that the process will not use more than the specified amount of memory. Like biology, the SC artificial cells are designed to keep what is inside contained. SC does not completely prevent malicious software from running, but aims to keep it in a controlled environment if it does get executed, see Figure 1. In order for a process to read or write data from a hard drive, the cell must allow the operation to take place. Opening a network connection, sending information to a video display, read-

ing input from a mouse or keyboard, and sending audio to speakers are some examples of the resources that SC controls. A computer system's memory is restricted by SC as well so processes can only access the portion reserved for them. Since the processes run inside virtual machines, software does not need to be modified in order to be used with the SC framework.
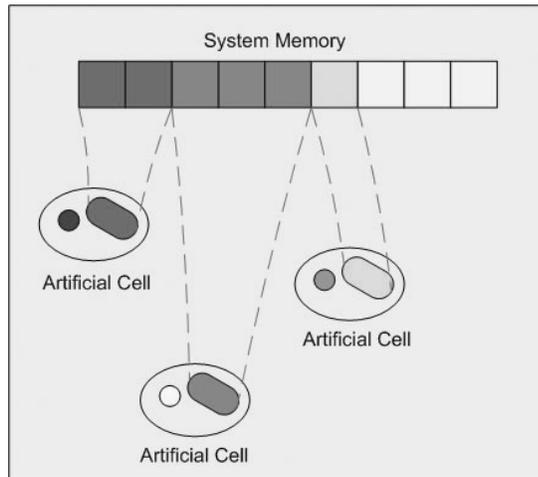
Figure 1. A Solitary Confinement system with artificial cells

Besides using virtual machines to compartmentalize and restrict access, SC is different in that the cells also use self-protecting components. Each cell in the framework can use multiple layers of defense from existing technology for detecting and defending itself from malicious and buggy processes. Cells that require network access could have built-in intrusion detection systems (IDS). Fingerprinting could be used to check the code on disk for any modifications before allowing a process to launch. A common way to fingerprint files is by using cryptographically strong hashes [2, 15] on the file itself such as a system like Tripwire [10]. The hashes for each file are generated and stored in a protected database. Verifying fingerprints of files on disk will protect against certain attacks but it does not prevent an attack on a process once it has launched and is running. Another layer of protection in SC could be checking code in memory. Computer systems with processors that allow executable data are very susceptible to buffer overflow attacks, specifically stack smashing [16]. One protection method that could be used with SC to prevent or detect stack smashing attacks is placing a random canary word in the stack beside the return address just as StackGuard does [3]. Buffer overflows can also occur on the heap, so fault containment wrappers [6] can be used to perform bounds checking before the function is called.

An additional layer of protection in SC might be look-

ing for anomalies in the behaviour of processes. Once SC is trained for normal behaviour, a cell could detect if that behaviour changes. If SC detected anomalous behaviour at some point, the virtual machine could slow down the execution of instructions to delay the process [21]. At the same time it could temporarily restrict access to some resources to see if the anomalous behaviour subsides. If the anomalous behaviour continued for more than a configured amount of time, SC would induce apoptosis to remove the virtual machine from the computer system in an attempt to prevent any further damage and warn the user. One way to profile behaviour is by looking at the system calls that a process makes such as analyzing sequences of system calls to distinguish between self (normal behaviour) and non-self (anomalous behaviour) [8, 21]. Another method also analyses the arguments being passed in an attempt to improve detection of anomalous behaviour [1]. A third method uses interception to capture the full semantics of the system call with automatically generated policies to detect if a process is trying to execute abnormal code [17].

A further layer of protection in Solitary Confinement could be looking for danger signals with the added benefit of being able to detect non-malicious bugs and other problems in software that cause distress. One theory on how the immune system works is by reacting to danger signals raised by damaged or stressed cells and tissue [12]. With SC, the danger signals are detected inside the artificial cells and the virtual machines will react accordingly. If something bad happens in a cell causing a danger signal, SC could increase access restrictions or remove that virtual machine by apoptosis and try to generate a new one. Several computer signals could be used indicate possible danger such as segmentation faults, low memory, and low disk space. Other danger signals could easily be added to the SC framework as required.

The increased protection from malicious software with SC does come with costs. Since each process is run inside a virtual machine, extra memory will be required for the virtual machine and OS running inside. Performance will also be decreased but with virtual machines like Xen [4], the penalty is only a few percent. Xen was designed so that one hundred virtual machines could easily run on a single server. The self-protecting components in each cell will also require memory and CPU time. Both Intel and AMD are adding hardware support to their processors for virtualization to increase the performance and aid in the management of virtual machines [9]. Although the artificial cells are autonomous, configuring each virtual machine with the proper access for a specific process could be complicated. The feasibility for configuring and launching a number of virtual machines on a system has already been demonstrated with Virtual Appliances [20]. A configuration language called CVL was created for such a purpose. The software devel-

opers or system administrators could create configuration profiles for the software that will be run on a given computer system. These profiles would contain information on what kind of network and file access the process will need inside a cell along with details of any other processes it may need to communicate with on the system. The profiles themselves would have to be protected against tampering. Even with the increased costs, the SC framework is feasible on today's computer hardware.

4 Conclusion

Solitary Confinement is a security framework that uses a mixture of traditional computer security with ideas inspired by biology and the immune system to reduce the amount of damage malicious and buggy software can inflict on a computer system. SC uses virtual machines to monitor and compartmentalize processes on a computer system. It restricts access to resources on the system like a cellular membrane, allowing only certain types of data and communications to pass through the virtual machine to the outside world. Each SC cell is autonomous, disposable, and has multiple layers of defense. An artificial cell will detect and prevent attacks automatically with little or no intervention from the user. New layers of defense and methods of protection can be easily added to the framework to increase and change SC's coverage over time.

5 Acknowledgements

Special thanks to Dr. Anil Somayaji and Dave B. Sharp for their comments and suggestions.

References

[1] C. K. amd D. Mutz, F. Valeur, and G. Vigna. On the detection of anomalous system call arguments.

[2] N. consortium. *Portfolio of recommended cryptographic primitives*, 2003.

[3] C. Cowan and C. P. et al. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks. In *Proc. 7th USENIX Security Conference*, pages 63–78, San Antonio, Texas, jan 1998.

[4] B. Dragovic and K. F. et al. Xen and the art of virtualization. In *In Proceedings of the ACM Symposium on Operating Systems Principles*, 2003.

[5] D. Engler. *The Exokernel operating system architecture*. PhD thesis, Massachusetts Institute of Technology, 1999.

[6] C. Fetzer and Z. Xiao. Detecting heap smashing attacks through fault containment wrappers. In *Proceedings of the 20th IEEE Symposium on Reliable Distributed Systems*, 2001.

[7] B. Ford and M. H. et al. Microkernels meet recursive virtual machines. In *Operating Systems Design and Implementation*, pages 137–151, 1996.

[8] S. Forrest and S. A. H. et al. A sense of self for Unix processes. In *Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy*, pages 120–128. IEEE Computer Society Press, 1996.

[9] Intel. *Virtualization (http://www.intel.com/technology/computing/vptech/)*, 2005.

[10] G. H. Kim and E. H. Spafford. The design and implementation of tripwire: A file system integrity checker. In *ACM Conference on Computer and Communications Security*, pages 18–29, 1994.

[11] J. W. Kimball. *Apoptosis (http://biology-pages.info/A/Apoptosis.html)*, 2004.

[12] P. Matzinger. The danger model: A renewed sense of self. *Science*, 296:301–305, 2002.

[13] D. Moore and V. P. et al. *The Spread of the Sapphire/Slammer Worm*, 2003.

[14] D. Moore, C. Shannon, and J. Brown. Code-red: a case study on the spread and victims of an internet worm. In *Proceedings of the Internet Measurement Workshop (IMW)*, 2002.

[15] NIST. Processing standards publication 180-2 (sha-1/2).

[16] A. One. Smashing the stack for fun and profit. *Phrack*, 7(49):14, 1996.

[17] N. Provos. Improving host security with system call policies. In *Technical Report 02-3, CITI*, 2002.

[18] N. Purchon. *Animal Cells (http://www.purchon.com/biology/animal.htm)*, 2000.

[19] RedHat. Red hat enterprise linux 4: Security guide. In *http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/security-guide/*, 2005.

[20] C. Sapuntzakis and D. B. et al. Virtual appliances for deploying and maintaining software. In *Proceedings of the 17th Large Installation Systems Administration Conference*, pages 181–194, 2003.

[21] A. Somayaji and S. Forrest. Automated response using System-Call delays. In *Proceedings of the 9th USENIX Security Symposium*, pages 185–198, 2000.

[22] A. Somayaji, S. Hofmeyr, and S. Forrest. Principles of a computer immune system. In *Meeting on New Security Paradigms, 23-26 Sept. 1997, Langdale, UK*, pages 75–82. New York, NY, USA : ACM, 1998, 1997.

[23] J. A. Sullivan. *CELLS alive! (http://www.cellsalive.com/cells/3dcell.htm)*, 1994.

[24] C. F. Tschudin. Apoptosis - the programmed death of distributed services. In *Secure Internet Programming*, pages 253–260, 1999.

[25] vmware. *The vmware official home page (http://www.vmware.com/)*, 2005.