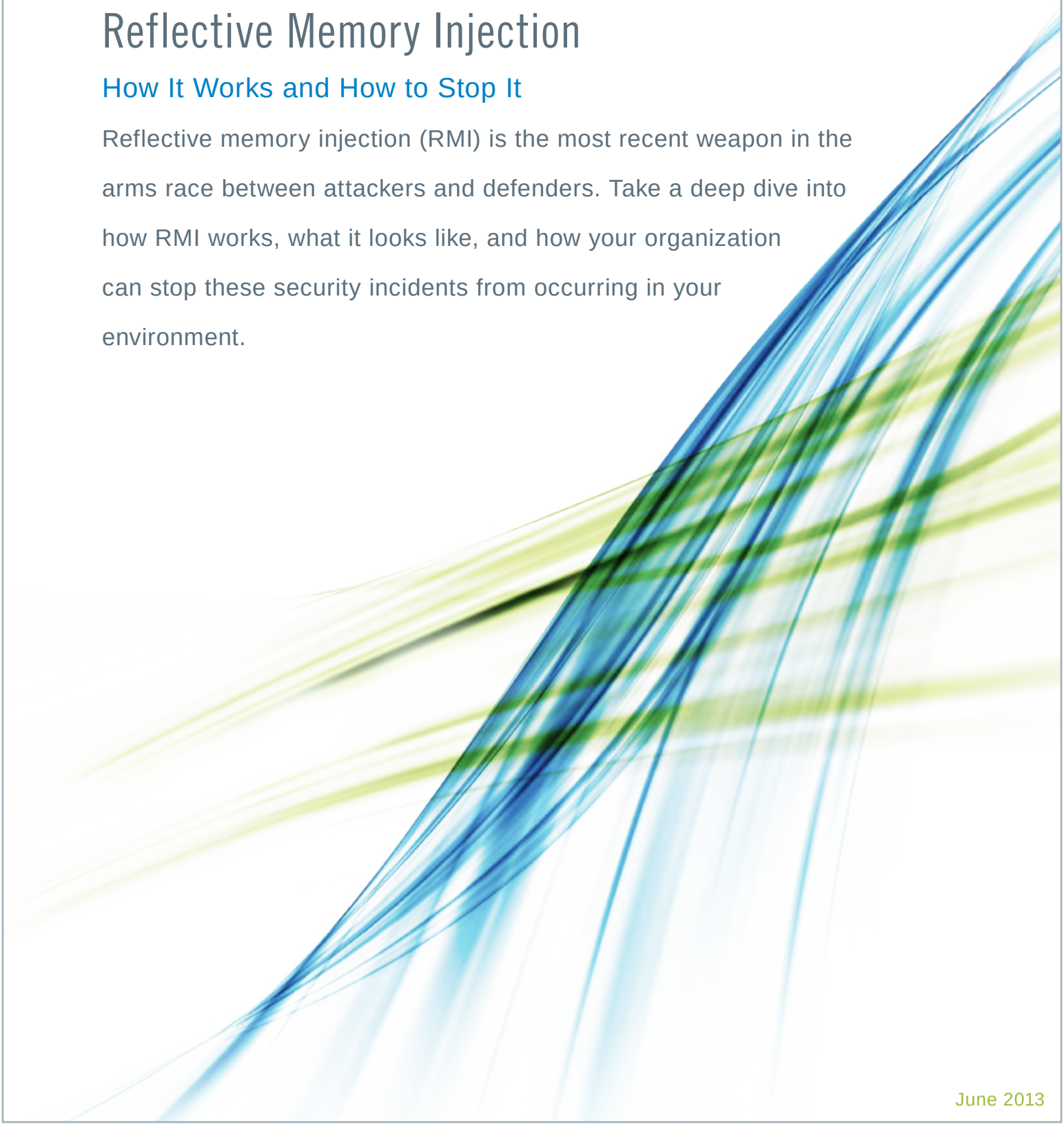
Reflective Memory Injection

How It Works and How to Stop It

Reflective memory injection (RMI) is the most recent weapon in the arms race between attackers and defenders. Take a deep dive into how RMI works, what it looks like, and how your organization can stop these security incidents from occurring in your environment.



June 2013

WP-EN-06-19-13

Overview

Reflective memory injection (RMI) is the most recent weapon in the arms race between attackers and defenders. Back in the 1990s, most security incidents centered around active penetration attempts, launched directly at the networks and servers of target organizations. Such attacks relied on simple security misconfigurations and unhardened systems that were left at their unsecured default settings.

As awareness grew, organizations began hardening their most exposed systems and implementing better network boundaries. But security patching was slow, and many systems simply went unpatched for years. So attackers started looking for software bugs that they could exploit, such as buffer overflows that allow malformed input to trick programs into running arbitrary code.

In time, software vendors (arguably lead by Microsoft) beefed up the default security settings of their products and began releasing patches more regularly, along with automated patch-delivery systems. These changes made life more difficult for attackers, but a significant window of opportunity still existed. Attackers began to exploit zero-day vulnerabilities, before vendors developed patches and customers deployed them to target systems.

There's no defense against a zero-day vulnerability until a patch is available and you follow through and deploy it. But most zero-day vulnerabilities are used as the initial vector through which a malicious executable can be downloaded to the victim system and configured to run persistently. If you can stop that EXE file from executing, you can prevent the attack from escalating.

At about the same time that zero-day exploits became popular with attackers, application whitelisting matured and became more common. Application whitelisting solutions such as *Lumension*® Application Control prevent new EXEs from running on protected systems unless the EXE passes various trust policies.

Software developers also began using more secure coding techniques and compilers. OS and hardware vendors began developing ways to make buffer overflows and related bugs less prevalent and more difficult to exploit and to restrict the size and flexibility of buffer overflows.

With buffer overflows more difficult than ever to exploit and the possibility that malicious EXEs could be blocked by application whitelisting, at-

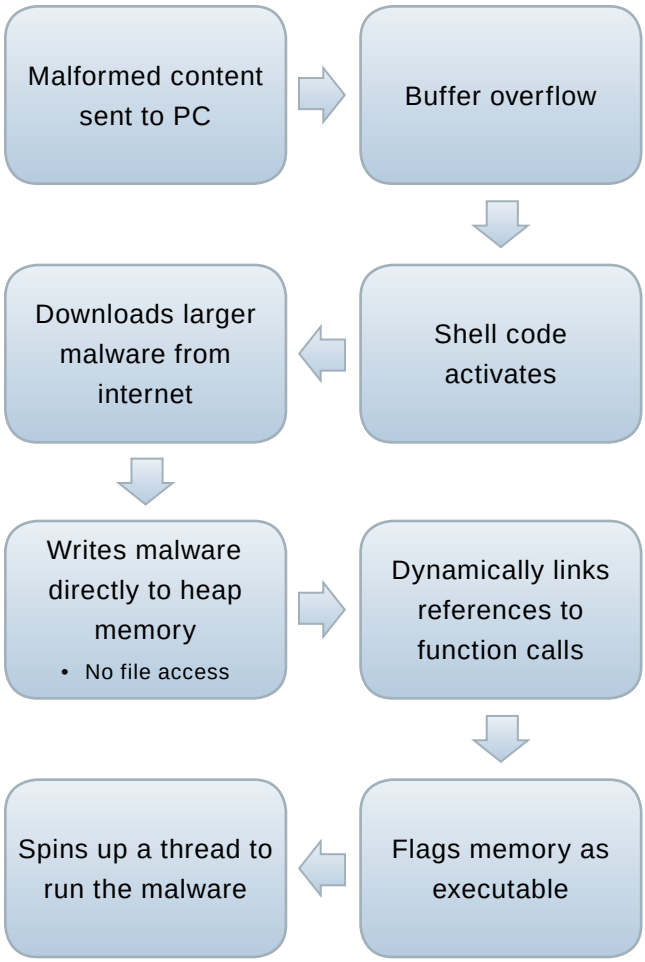
tackers were under pressure to find something new. In 2008, Stephen Fewer published a paper, "Reflective DLL Injection," which detailed how to inject a DLL into a host process without using the standard APIs that triggered security checks by application whitelisting. Fewer provided proof-of-concept code modules that could be used with the Metasploit toolkit.

How Does RMI Work?

Before an attacker can use RMI, the attacker must be able to run a small amount of executable code and must have downloaded or otherwise copied the bytes of malicious DLL into memory. There are many, many ways to accomplish this. We will describe just one.

The attacker places a malicious web page on some site that includes a malformed JPEG image, exploiting a buffer overflow in the Windows image-rendering engine. The attacker succeeds in getting the user to view the web page, at which point the browser downloads and renders the JPEG and renders it, triggering the buffer overflow.

The buffer overflow results in the execution of a small amount of arbitrary CPU instructions (called a shell code). The shell code downloads the bytes of larger malicious DLL into memory. The shell code then calls in to a tiny bootstrap loader function in the DLL. This is where the reflective part of RMI begins.



When you load a DLL into memory, it can't immediately begin to run. It first needs to call certain standard functions, called imported functions, that are provided either by the compiler's standard library or by the OS. The library also exports functions that the OS or other libraries might need to call back into the original library. These references are initially symbolic and must be replaced with the actual memory addresses of those functions which change each time a process is started since executables can be loaded into any location in memory. This process is called dynamic linking and accounts for the term *Dynamic Link Library* (DLL).

Why Doesn't Antivirus or Application Whitelisting Detect RMI?

The shell code keeps the DLL in memory; it does not write the DLL to disk. This distinction is important. Security technologies such as antivirus and application whitelisting can easily "see" a new DLL that appears in the file system and potentially quarantine it or block it from being loaded. Writing the DLL to disk and loading it via the LoadLibrary API is much easier and is referred to as Simple DLL Injection.

For a library to link itself, it must examine its own bytes to find and resolve the symbolic references of its imported and exported functions. This process is analogous to a program looking at itself in a mirror. That is where the term *Reflective Memory Injection* comes from.

A legitimate program simply uses the OS to accomplish this process, by calling the LoadLibrary() API. But attackers want to link up to the malicious library without tipping their hand by touching the disk or calling an obvious API.

After this linking is accomplished, a thread is spun up to run the main code of the malicious DLL. Then the attacker can get down to business, having silently delivered and activated a potentially large payload of malicious functionality, beginning with a tiny buffer overflow and without tripping any alarms. At this point the attacker can proceed with the attack, executing any features of the new DLL under the auspices of whichever user is logged on.

Continued »

What Does an RMI Process Look Like?

Detecting RMI code is complex. Classic methods of detecting DLLs that are injected through normal APIs are obsolete. For instance, since the DLL was not loaded with LoadLibrary(), the malicious code won't show up in the process's metadata.

You must look for evidence of what RMI does, which is to allocate a region of private memory and flag it for not just Read and Write access but also for Execute. The fact that the chunk of memory is in the *data* part of a process's memory but is flagged to allow execution is suspicious; a program's executable code normally resides in the process's "image" (EXE) region of memory and in any legitimately mapped DLL file. The region must be marked as writable so that the shell code can write the code into the region. It must also be flagged as executable so that the shell code can subsequently pass execution to it.

However, just looking for chunks of private memory that are flagged RWX will lead to false positives. Some legitimate programs, such as the .NET Runtime, use self-modifying code.

To eliminate such false positives, you must examine the contents of the suspicious memory chunk. Look for structures and constants that show the chunk to be a DLL in disguise. DLLs have a standard format with several headers and tables. Look for are the MZ and PE headers, which are prefixed by those actual characters, as shown in the following figure. The Process Hacker screen in this figure shows a region of private memory in Microsoft Internet Explorer (IE) that is flagged for Read, Write, and Execute. In addition, the contents of the memory are a positive match for a DLL in Win32 Portable Executable (PE) format.

Although the contents of the memory region is a DLL and is writable, other investigation shows that it does not appear in the list of DLLs that are loaded through the normal API, nor is it mapped to a file on the file system. This is suspicious but not definite. Legitimate cases of programming could yield the same appearance, but these are unusual, such as copy-protection mechanisms in game software. Indeed, code that loads this way is code that wants to hide.

How Lumension Detects RMI

Lumension® Application Control (part of *Lumension*® Endpoint Management and Security Suite) has long been a leader in application whitelisting, which is the most effective way to prevent malicious or otherwise untrusted or unwanted applications from executing.

RMI bypasses the file system and security technologies that interface with the file system. Through the acquisition of CoreTrace, *Lumension*® Application Control includes CoreTrace's Bouncer patent-pending technology. Bouncer can detect and stop RMI attacks by monitoring an endpoint's memory address space and associated processes for distinct evidence of exploitation. The architecture and kernel-level position of *Lumension*® Application Control allow it to extend beyond simple whitelisting to provide memory protection.

In order to prevent exploits such as DLL injections and reflective memory injections, *Lumension*® Application Control extends the whitelisting model into memory, preventing execution of processes that originate from unauthorized programs. *Lumension*® Advanced Memory Protection eliminates one of the biggest endpoint security blind spots that attackers target.

Lumension® Advanced Memory Protection is an integral part of an overall layered strategy in *Lumension*® Endpoint Management and Security Suite for defense against sophisticated attacks such as Advanced Persistent Threats (APTs). *Lumension*® Endpoint Management and Security Suite provides layered protection against memory-based attacks by combining market-leading patch and remediation management and traditional application-whitelisting capabilities.

The defense-in-depth capabilities in *Lumension*® Endpoint Management and Security Suite provide organizations with these capabilities:

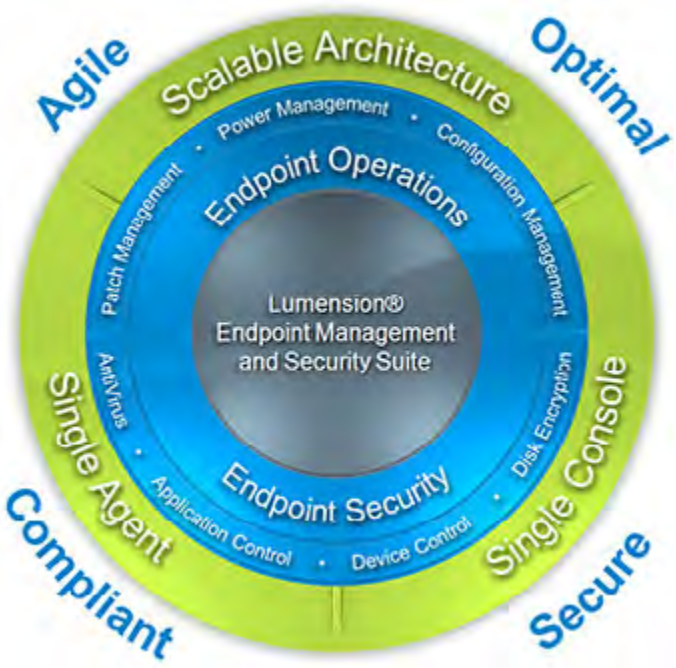
- Remove all known memory-based vulnerabilities and ensure that the attackable surface area is as small as possible.
- Identify and block attempted memory-injection exploits in memory, through Advanced Memory Protection.
- Deny any attempts to install on-disk payloads, through the proven application-whitelisting security model

RMI Protection as Part of a Complete Endpoint Security Solution

Application whitelisting and RMI protection go hand-in-hand to provide comprehensive protection against malware and the APTs that use malware. And preventing malicious code (whether in memory or on disk) is the single most effective way to stop APTs.

But more than one layer of security is required for defense-in-depth. To defend against today's risks, endpoints need multiple security technologies:

- Configuration control
- Patch management
- Antivirus
- Device control
- Vulnerability scanning
- Encryption



Deploying all these technologies can lead to a confusing array of products, agents, and consoles, all competing for endpoint computing resources and attention from IT staff. In contrast, Lumension delivers all these technologies as a single-server, single-agent, single-console platform that provides modularly licensed, best-of-breed capabilities and systems management—and that can grow with you as your needs evolve.

Lumension® Endpoint Management and Security Suite enables you to take control of your endpoints through an agile solution suite that simplifies systems management, expands operational visibility, and delivers more effective IT security, all while reducing complexity and endpoint total cost of ownership (TCO).

Learn more about *Lumension*® Endpoint Management and Security Suite at http://www.lumension.com/endpoint-management-security-suite.aspx watch an on-demand demo at http://www.lumension.com/endpoint-management-security-suite/demo-in-detail.aspx or start a free virtual hosted or on premise trial at http://www.lumension.com/endpoint-management-security-suite/free-trial.aspx.

About Randy Franklin Smith

Randy Franklin Smith is an internationally recognized expert on the security and control of Windows and Active Directory security who specializes in Windows and Active Directory security. Randy publishes www.UltimateWindowsSecurity.com and wrote *The Windows Server 2008 Security Log Revealed* – the only book devoted to the Windows security log. Randy is the creator of LOGbinder software, which makes cryptic application logs understandable and available to log-management and SIEM solutions. As a Certified Information Systems Auditor, Randy performs security reviews for clients ranging from small, privately held firms to Fortune 500 companies, national, and international organizations. Randy is also a Microsoft Security Most Valuable Professional.

Disclaimer

About Lumension Security, Inc.

Lumension Security, Inc., a global leader in endpoint management and security, develops, integrates and markets security software solutions that help businesses protect their vital information and manage critical risk across network and endpoint assets. Lumension enables more than 5,100 customers worldwide to achieve optimal security and IT success by delivering a proven and award-winning solution portfolio that includes Vulnerability Management, Endpoint Protection, Data Protection, Antivirus and Reporting and Compliance offerings. Lumension is known for providing world-class customer support and services 24x7, 365 days a year. Headquartered in Scottsdale, Arizona, Lumension has operations worldwide, including Texas, Florida, Washington D.C., Ireland, Luxembourg, Singapore, the United Kingdom, and Australia. Lumension: IT Secured. Success Optimized.™ More information can be found at www.lumension.com.

Lumension
IT Secured. Success Optimized.™

Global Headquarters
8660 East Hartford Drive, Suite 300
Scottsdale, AZ 85255 USA
phone: +1.480.970.1025
fax: +1.480.970.6323