Check Point
SOFTWARE TECHNOLOGIES LTD.

We Secure the Internet.

Malicious Code Protector

A New Approach for Detecting and Blocking Buffer Overflow Attacks

In This Document

INTERNAL WEB PERIMETER
Intelligent Security

Check Point SOFTWARE TECHNOLOGIES LTD.
We Secure the Internet.

Introduction

Attacks using buffer overflow vulnerabilities have become one of the greatest security concerns on the Internet, with buffer overflows sited in over 50% of the most recent CERT advisories. In fact, the first 6 advisories of 2003 are directly related to buffer overflow vulnerabilities. The shear power of these attacks and the large number of hosts that are vulnerable make it a popular attack vector. Some of the largest attacks in the past several years can be linked to buffer overflow vulnerabilities:

Malicious Attack	Year	Economic Damage[1]	Buffer Overflow Vulnerability
Sasser	2004	$3.5 Billion	Microsoft Local Security Authority Subsystem Service (MS04-011)
SQL Slammer	2003	$1 Billion	Microsoft SQL Server and MSDE 2000 (MS02-061)
MS Blaster	2003	$750 Million	Microsoft RPC- Remote Procedure Call (MS03-039)
Code Red	2001	$2.6 Billion	Microsoft IIS Server: Internet Server Application Program Interface (MS01-033)

Buffer overflow attacks, sometimes referred to as malicious code attacks or stack attacks, are popular for two fundamental reasons: they ultimately let an attacker do whatever they want to do on a target host; and most application programs are vulnerable to this attack. Hackers can use this exploit to modify files, launch applications, or even inject executable code of their own. A worm is a good example of the later case, using the buffer overflow as an entry point for the worm. This translates into an attack that can spread across the Internet in hours, seen in these two examples: 250,000 hosts were hit with the Code Red worm within 9 hours[2]; and even more dramatically the Witty worm was able to hit 110 hosts in the first 10 seconds of the attack[3]. Given the scope of the problem, buffer overflow attacks will continue to be an issue for the foreseeable future.

While attack countermeasures have been developed to identify a buffer overflow attack after it has emerged, current solutions are unable to identify new attacks or variations on the attack. This is a critical problem given the spread at which these attacks can spread. To solve this problem Check Point developed Malicious Code Protector, a patent-pending technology that provides a new approach for detecting and blocking buffer overflow attacks. Unlike previous detection solutions that rely on a defined attack signature, Malicious Code Protector uses a fundamentally different detection method with the capability of detecting previously unknown attacks and variations on existing attacks.

[1] Computer Economics, “The Impact of Malicious Code,” June 2004

[2] CERT/CC, ‘CERT® Advisory CA-2001-23 Continued Threat of the “Code Red” Worm’, January 17, 2002, URL: http://www.cert.org/advisories/CA-2001-23.html.

[3] Colleen Shannon and David Moore, “The Spread of the Witty Worm”, Cooperative Association for Internet Data Analysis, URL: http://www.caida.org/analysis/security/witty/

INTERNAL WEB PERIMETER
Intelligent Security

Buffer Overflow Attacks

Buffer overflow attacks target the way host machines handle input data and memory space. When an application is running on a host machine it allocates a certain portion of memory (the buffer) for input data to be placed. The problem arises, because while the buffer used by the application is a fixed size, the application itself may not restrict the amount of data that can be input into the buffer. For example, a programmer may expect data to be less than 26 bytes and will allocate the appropriate amount of memory. However, a user may input 27 bytes of data. The result is the application writes more data than is allocated in the buffer (the overflow) and corrupts the memory.

Why is this important? This innocent programming oversight can provide an attacker with the ability to break out of the context of the running application and execute an attacker's malicious code inserted with the attack. To take advantage of this ability (vulnerability) to overflow a buffer, an attacker writes an attack such that it is both larger than the buffer of a particular application and whose payload contains some executable code in machine assembly language (see glossary). The attacker also cleverly includes memory pointers, or addresses in memory that redirect the flow of a running application. The result is an application that takes input data, overruns its buffer, encounters a memory pointer that points back to the malicious executable code contained in the payload of the attack, and then, via the host, begins to run the malicious code.

There are two qualities that make this attack attractive to hackers. First, the attack can apply to any application that takes input data. From a network perspective, this can include Web servers (HTTP), DNS servers, FTP servers, networked Microsoft applications, etc. Second, given that as part of the attack the host begins to run the attackers malicious code, an attacker can create malicious code to do whatever it wants. This includes such things as executing remote shell commands (command line), opening backdoors for further attacks, running a worm, or installing a trojan horse.

For example, a buffer overflow vulnerability found in Microsoft IIS Servers allowed an attacker to write malicious code that ultimately gave full control over these servers. This is now commonly known as the Code Red attack. While the first Code Red merely used the target host to attack www.whitehouse.gov. Code Red II was more serious, because it installed a trojan horse on the hosts. Using the same vulnerability and malicious code, two very serious and different attacks were created.

Current Defenses Against Buffer Overflow Attacks

In an ideal world applications would not accept more data than is actually allocated in the buffer and no host system would allow an application to write past a buffer in its memory. However, there are thousands of applications in the world that don't validate input data and millions of hosts that run these applications. The sheer size of the problem makes it impractical to go back to each and every application, as well as each place within an application that accepts user input, and implement validation checking. To solve this dilemma, organizations are turning to network-based solutions as a one-stop solution to prevent the injection of buffer overflow attacks into an organization's network.

IDS/IPS Solutions

Intrusion Detection Systems (IDS) are designed to detect application-level attacks by examining data for known patterns (signatures) or anomalies that may indicate the existence of an attack in a stream of data.

IDS are traditionally used to fine-tune a security policy and defenses after a security incident has occurred, but do not offer proactive protection. This is critical given that an attack can be contained in a single packet. The SQL Slammer worm, for example, was a small 276 byte worm that did its damage in a single packet. The advantage of an Intrusion Detection System is the low probability of false alarms since the search criteria of signatures can be tightly defined. The

disadvantages are also obvious, new attacks are frequently missed – and only take a slight modification to the attack to make the Intrusion Detection System ineffective. The Code Red II variant, for example, differed by only 13 bytes from the original and was able to infect hosts 3 times faster than the original[4].

While some Intrusion Detection Systems claim to look for machine code, they do not have the capability to examine machine assembly code or the Virtual Server analysis used by MCP to provide accurate, speedy attack detection. In reality, IDS systems do not look at machine assembly instructions but rather look for a specific string of binary numbers that may infer that assembly instructions exist. This inference of machine assembly code rather than actual assembly code analysis is a fundamental reason why IDS systems cannot detect new attacks and have low detection rates for variations of existing attacks.

Intrusion Protection Systems (IPS) actively block perceived threats to a network, based on common signatures. However, similar to Intrusion Detection Systems, Intrusion Protection Systems are heavily signature-based and as such, fail to proactively thwart previously unknown threats.

A New Approach: Malicious Code Protector

Malicious Code Protector is a patent-pending technology from Check Point designed to detect malicious code attacks targeting applications with buffer overflow vulnerabilities. It is designed to recognize attacks without requiring a signature of a known attack or variations of an attack. The following section describes the goals of Malicious Code Protector, assumptions that help it accurately detect attacks, and how it detects actual malicious code.

Design Goals

Malicious Code Protector was designed with the following core goals:

1. **Ability to detect unknown attacks.** The goal of Malicious Code Protector is to detect an actual attack rather than a signature of an attack. By definition, attack signatures cannot anticipate or be “pre-created” for a new attack. However, the coding of executables in network-based buffer overflow attacks all share common, unique characteristics that can be identified and characterized more efficiently with an actual examination of the executable code itself.
2. **Protocol independence.** Malicious Code Protector was designed to integrate with the Check Point Application Intelligence framework, which supports dozens of protocols. As recent attacks have shown, even long standing protocols such as MS-RPC and SQL can be exploited with new attacks.
3. **Fast.** The goal was to design a solution that can work at wire speed. This means it should run on standard Pentium hardware supporting traffic rates of at least 1 Gigabit per second, to accommodate the needs of most enterprises.
4. **Multiple platform compatibility.** The Malicious Code Protector was designed to support multiple server platforms. While the current solution currently protects Windows and Intel platforms (>90% servers on the Internet), it can be extended to support additional platforms such as Sun Microsystems SPARC based systems.

[4] Baca, Jeremy, “Windows Remote Buffer Overflow Vulnerability and the Code Red Worm,” SANS Institute, 2001.

Design Assumptions

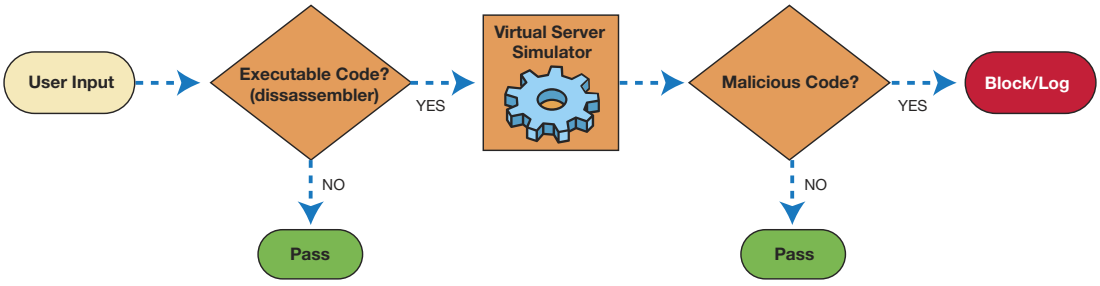
As discussed earlier, buffer overflows attack the memory system of a target host using specially crafted malicious code. All buffer overflow attacks send executable code in the network payload of an attack. Buffer overflow attacks must contain code that will be run on the target computer; otherwise they can not perform any hostile actions.

Based on the above, the following assumptions were made:

1. All network-based malicious overflow attacks must contain executable code in machine language.
2. Network traffic does not usually contain executable machine code. In the rare cases where a legitimate executable code is transferred over the network (e.g. download of an .exe file), it can be easily identified as such. Typically, EXE files are sent from servers to clients, while attacks are launched from clients to servers.
3. It is possible to write an algorithm to detect machine code in network traffic with high accuracy, low false positives rates and high performance.

Based on the assumptions above, Check Point created an algorithm that meets the design goals of the Malicious Code Protector. Since Malicious Code Protector can detect machine code in network traffic, and we know that each attack must have machine code from our assumptions, the algorithm can detect actual attacks regardless of the specific buffer overflow vulnerabilities an attack is exploiting.

The Algorithm



Looking for Executable Code

The heart of the Malicious Code Protector is a disassembler engine that can examine network traffic and detect executable code (i.e., disassemble binary data into machine assembly language). This ability to detect executable code is related to the assumption that executable code is normally not allowed to traverse a network, with the exception of a few well known cases, such as an FTP transfer of an executable (*.exe) file. Malicious Code Protector monitors data streams and looks for a sequence of data that the disassembler engine can translate into machine assembly language. This indicates the possible existence of executable code passing through a network.

However, this alone is not sufficient when trying to determine whether a certain data stream contains executable code, let alone code of malicious nature. There are instances where non-executable data can generate an assembly-looking output. For example, a .gif file can in some cases produce machine assembly instructions even though it is not an application. Therefore, the Malicious Code Protector must be able to distinguish between the random “noise” of assembly-like data and a real executable in network traffic.

Virtual Server: Determining Malicious Intent

To determine whether a suspected block of data is actually executable code, Malicious Code Protector examines the sequence of assembly instructions (i.e., Op Codes) to look for a logical relationship between the instructions and the expected use of these instructions. Given the greater context of the assembly instructions, and the existence of larger "meta-instructions", Malicious Code Protector determines the likelihood that a given set of assembly instructions is in fact executable code.

Yet, the mere existence of executable code does not necessarily count for its malicious nature. The final Malicious Code Protector step is to examine the set of assembly instructions to look for particular meta-instruction sets. As mentioned earlier in the description of buffer overflow attacks, hackers use certain programming techniques within their malicious code to get the target to actually run the malicious code. They typically contain a similar set of instructions to work around uncertainties pertaining to the size and location of a buffer. Malicious Code Protector looks for such instructions to identify a malicious application. It then feeds the instructions into a Virtual Server to analyze its reaction to the instructions. This virtualized environment is a fundamental reason Malicious Code Protector is able to detect malicious code. Finally, the MCP analyzes the results of all of these steps to determine whether a certain data stream is in fact a malicious code.

A Technical Evolution

Disassembly technology has been around for quite a while, but has not been used for network-based attack detection due to accuracy and performance challenges. However, these limitations are overcome by integrating disassembly technology with Check Point Stateful Inspection and Application Intelligence technologies. This combination increases accuracy through a much deeper understanding of the application flow. In addition, performance is greatly increased by applying the disassembly only where it is possible to inject a malicious attack into network traffic.

Accuracy

Accuracy is a major issue in machine learning methodologies, with the inevitable tradeoff between detection rates and false positive rates; the problem lies within the ability to achieve both at the same time.

Using ONLY a Disassembler to identify malicious code could yield high detection rates, but would also cause high false positive rates. To prevent this, Malicious Code Protector uses a second layer of disassembly which turns machine assembly code into logical programming elements such as loops and branches. Since the logical constructs are higher in this approach, the likelihood of random constructs spontaneously appearing is lower, resulting in much lower false positive rates. By achieving a higher level of understanding of what the executable code is attempting to do, malicious intents can be better identified.

To improve the accuracy detection rates, Malicious Code Protector looks for logical constructs used by hackers in buffer overflows. Since all buffer overflows are limited in size and share many limitations inherent in these network-based attacks, it is possible to identify repeating constructs, such as decryption loops, which are indicative of malicious code.

After running several sets of detectors, Malicious Code Protector uses machine learning techniques to combine the weights achieved by each flow of the algorithm. The combination of weights increases accuracy by ruling out exceptions and looking at common context.

Performance

Disassembly of bytes into machine assembly language is a computer-intensive operation. Interactive Disassembly, which can follow code branches, requires even more computing resources. To overcome these performance issues and operate at wire speed, Malicious Code Protector implements several performance enhancing techniques. Performance considerations include:

1. Performance is greatly increased by applying the algorithm only in places where malicious code can be inserted. For example, in the context of Web communication between a client and a Web server, there are only a few places where a server will be looking for user input (URLs, forms, etc.). These are the only places where a server will act on the input and is where hackers place malicious code. Understanding the context of application communication with Application Intelligence, Malicious Code Protector can run faster and with a much better accuracy rate.
2. Malicious Code Protector is stream based, and can work on a single byte at a time. This allows it to be used in high performance environments where packets are forwarded rather than stored.
3. Malicious Code Protector implements specific optimization techniques, including caching, hashing and information sharing in order to avoid repetitive operation calculations. This is especially useful with servers, since requests tend to repeat themselves (e.g. server name).

Advantages Over Existing Solutions

When compared to network based systems that use signatures of attacks (IDS/IPS), Malicious Code Protector delivers two fundamental advantages:

1. Pre-Emptive. Without having to rely on signatures created from analysis of a known attack, Malicious Code Protector eliminates the time lag between when an attack emerges and when a signature is defined and published in the market. Through its analysis of the machine assembly language in a Virtual Server environment, Malicious Code Protector can detect new attacks that seek to exploit buffer overflow vulnerabilities. This provides attack protection from “Day 0”.
2. Accurate. By looking at actual machine assembly language and its behavior in a Virtual Server environment, Malicious Code Protector can detect variations on attacks that typically follow an outbreak of malicious code (e.g., Code Red II followed 7 days behind the original attack). The binary “signature” of an attack variant can differ enough from the existing signature that it will no longer match the variant, even if the attack itself is still exploiting the same buffer overflow vulnerability. Malicious Code Protector overcomes this deficiency by examining the actions of the malicious code rather than a binary string that could represent it.

Malicious Code Protector and Check Point Products

As discussed in the design goals section of this document, Malicious Code Protector was designed to be protocol independent. It can be run against any type of network traffic to look for malicious code designed to exploit buffer overflow vulnerabilities in target hosts. The first release of Malicious Code Protector is with Web Intelligence and is used to inspect Web (HTTP) traffic. Integrated with Web Intelligence, Malicious Code protector is used to inspect Web traffic that has the potential to carry user input to a Web server (URLs, forms within Web pages, etc.).

Web Intelligence is integrated in the following products:

- VPN-1 NG with Application Intelligence, R55W or higher: Web Intelligence protects Web applications behind a VPN-1 gateway.
- Connectra: Web Intelligence protects Web servers being accessed through the Connectra gateway, as well as protecting Connectra itself.

[5] http://www.caida.org/analysis/security/code-red/

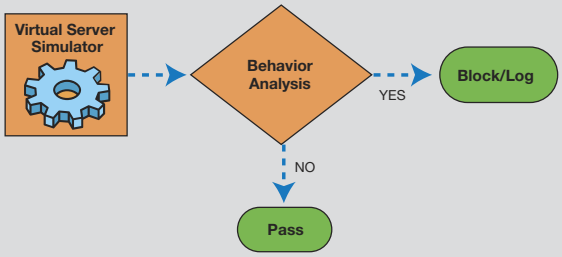
Step	Description	View of Datastream
1	This is an example of a data stream containing an embedded malicious code attack (malicious code is in blue)	...ff 73 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 6f 6f 6f 6f 6f 6f 6f 6f 6f 6f 6f 6f 42 42 42 42 8b 89 e8 77 eb 15 5b 53 68 aa 01 78 58 ff d0 31 c9 b1 11 58 e2 fd 31 c0 48 c3 e8 e6 ff ff ff 73 74 61 72 74 20 63 61 6c 63 2e 65 78 65 00 0d 0a 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 63 6c 6f 73 65 0d 0a 0d 0a 0d 0a...
2	With an understanding of the HTTP Web protocol, Web Intelligence recognizes this stream as an HTTP GET with user input data, a likely place a hacker would put a malicious code attack.	GET / HTTP/1.1 Host: ooooooooooooBBBBïë_w_ [Shj¬ xX _1__ X__1_H__µ start calc.exe Connection: close
3	Determining that this is a possible point of attack, the disassembler engine of Malicious Code Protector examines the datastream to look for the existence of machine assembly code. In any datastream some data will randomly translate as assembly code. However, in this case a string of assembly code indicates the existence of executable code.	JMP+26, CALL-26, POP EBX, PUSH EBX, PUSH 0x7801AAAD, POP EAX, CALL EAX
4	In the last step, Malicious Code Protector runs the executable code in a Virtual Server environment. Based on an analysis of the behavior of the executable code, it can be determined that an executable is attempting to exploit a buffer overflow vulnerability and is malicious.	

Table 1: Step-by-step description of how Web Intelligence uses Malicious Code Protector to inspect Web traffic:

Conclusion

Attacks exploiting buffer overflow vulnerabilities in applications have developed into considerable security issues, causing billions of dollars in damage. Given the thousands of applications that contain these vulnerabilities, and with more vulnerabilities being discovered every day, the problem is likely to remain an issue for years to come. Today, a window of vulnerability exists between when an attack emerges and when an attack signature can be written and distributed. Given that worms containing malicious code can infect thousands of hosts a minute, this critical delay in protection is costly. Malicious Code Protector is a new approach to protecting against buffer overflow attacks. With its ability to examine actual executable code and run it through a Virtual Server environment, Malicious Code Protector delivers the “Day 0” protection that is critically needed to minimize the damage to emerging attacks on the Internet.

Check Point
SOFTWARE TECHNOLOGIES LTD.
We Secure the Internet.

Glossary

Machine Assembly Language: The actual computer language executed by a CPU. In the context of high-level programming languages, such as C++, machine assembly language results from taking a program written in the high-level programming language and running it through a compiler for a given processor.

Disassembler: A tool that takes binary data and translates it into machine assembly language based on a specific processor (i.e., Intel, SPARC, etc.). The results are the actual operational instructions a processor would use execute the program (e.g., XOR, NOOP, etc.)

About Check Point Software Technologies

Check Point Software Technologies is the worldwide leader in securing the Internet. It is the confirmed market leader of both the worldwide VPN and firewall markets. Through its Next Generation product line, the company delivers a broad range of intelligent perimeter, internal and Web security solutions that protect business communications and resources for corporate networks and applications, remote employees, branch offices and partner extranets. The company's Zone Labs (www.zonelabs.com) division is one of the most trusted brands in Internet security, creating award-winning endpoint security solutions that protect millions of PCs from hackers, spyware and data theft. Extending the power of the Check Point solution is its Open Platform for Security (OPSEC), the industry's framework and alliance for integration and interoperability with "best-of-breed" solutions from over 350 leading companies. Check Point solutions are sold, integrated and serviced by a network of more than 2,300 Check Point partners in 92 countries.

CHECK POINT OFFICES:

Worldwide Headquarters:
3A Jabotinsky Street, 24th Floor
Ramat Gan 52520, Israel
Tel: 972-3-753 4555
Fax: 972-3-575 9256
e-mail: info@CheckPoint.com

U.S. Headquarters:
800 Bridge Parkway
Redwood City, CA 94065
Tel: 800-429-4391 ; 650-628-2000
Fax: 650-654-4233
URL: **http://www.checkpoint.com**

September 2, 2004 PN: 000000

INTERNAL WEB PERIMETER
Intelligent Security