SigFree: A Signature-free Buffer Overflow Attack Blocker

1.0 Aim of the project:

We proposed SigFree, a real-time, signature free, out of- the-box blocker that can filter code-injection buffer overflow attack messages, one of the most serious cyber security threats, to various Internet services. SigFree does not require any signatures, thus it can block new, unknown attacks

1.1 Description of the project in short:

SigFree can filter out code-injection buffer overflow attack messages targeting at various Internet services such as web service. Motivated by the observation that buffer overflow attacks typically contain executables whereas legitimate client requests never contain executables in most Internet services, SigFree blocks attacks by detecting the presence of code.

SigFree first blindly dissembles and extracts instruction sequences from a request. It then applies a novel technique called code abstraction, which uses data flow anomaly to prune useless instructions in an instruction sequence. Finally it compares the number of useful instructions to a threshold to determine if this instruction sequence contains code. SigFree is signature free, thus it can block new and unknown buffer overflow attacks. SigFree is also immunized from most attack-side code obfuscation methods

2.0 Algorithms

2.1) Distilling Instruction Sequences:-

To distill an instruction sequence, we first assign an address (starting from zero) to every byte of a request, where address is an identifier for each location in the request. Then, we disassemble the request from a certain address until the end of the request is reached or an illegal instruction opcode is encountered. There are two traditional disassembly algorithms: linear sweep and recursive traversal. The linear sweep algorithm begins disassembly at a certain address and proceeds by decoding each encountered instruction. The recursive traversal algorithm also begins disassembly at a certain address, but it follows the control flow of instructions.

2.2) Check if the number of useful instructions in an execution path exceeds a threshold:-

If the number of useful instructions in an execution path exceeds a threshold, we will conclude the instruction sequence is a segment of a program. Algorithm 2 shows our algorithm to check if the number of useful instructions in an execution path exceeds a threshold. The algorithm involves a search over an EISG in which the nodes are visited in a specific order derived from a depth first search. The algorithm assumes that an EISG G and the entry instruction of the instruction sequence are given, and a push down stack is available for storage. During the search process, the visited node (instruction) is abstractly executed to update the states of variables, find data flow anomaly, and prune useless instructions in an execution path.

Main Modules:-

1. Prevention/Detection of Buffer Overflows:-

Throughout the history of cyber security, buffer overflow is one of the most serious vulnerabilities in computer systems. Buffer overflow vulnerability is a root cause for most of the cyber attacks such as server breaking-in, worms, zombies, and botnets. Buffer overflow attacks are the most popular choice in these attacks, as they provide substantial control over a victim.

Class 1A:-

Finding bugs in source code. Buffer overflows are fundamentally due to programming bugs. Accordingly, various bug-finding tools have been developed. The bug-finding techniques used in these tools, which in general belong to static analysis, include but not limited to model checking and bugs-as deviant- behavior.

Compiler extensions:-

If the source code is available, a developer can add buffer overflow detection automatically to a program by using a modified compiler."

Class 1C: OS modifications:-

Modifying some aspects of the operating system may prevent buffer overflows such as Pax , LibSafe and e-NeXsh .

Class 1C techniques need to modify the OS. In contrast, SigFree does not need any modification of the OS.

Class 1D: Hardware modifications:-

A main idea of hardware modification is to store all return addresses on the processor. In this way, no input can change any return address.

Class 1E: Defense-side obfuscation:-

Address Space Layout Randomization (ASLR) is a main component of PaX . Bhatkar and Sekar proposed a comprehensive address space randomization scheme. Address space randomization, in its general form, can detect exploitation of all memory errors.

Class 1F: Capturing code running symptoms of buffer overflow attacks:-

Fundamentally, buffer overflows area code running symptom. If such unique symptoms can be precisely captured, all buffer overflows can be detected.

2. Worm Detection and Signature Generation

The implementation of their approach is resilient to a number of code transformation techniques. Although their techniques also handle binary code, they perform offline analysis. In contrast, SigFree is an online attack blocker. As such, their techniques and SigFree are complementary to each other with different purposes. Moreover, unlike SigFree, their techniques may not be suitable to block the code contained in every attack packet, because some buffer overflow code is so simple that very little control flow information can be exploited

3. SigFree Attack Model

An attacker exploits a buffer overflow vulnerability of a web server by sending a crafted request, which contains a malicious payload. Figure 3 shows the format of a HTTP request. There are several HTTP request methods among which GET and POST are most often used by attackers. Although HTTP 1.1 does not allow GET to have a request body, some web servers such as Microsoft IIS still dutifully read the request-body according to the request-header's instructions (the CodeRed worm exploited this very problem). The position of a malicious payload is determined by the exploited vulnerability. A malicious payload may be embedded in the Request-URI field as a query parameter. However, as the maximum length of Request-URI is limited, the size of a malicious payload, hence the behavior of such a buffer overflow attack, is constrained. It is more common that a buffer overflow attack payload is embedded in Request-Body of a POST method request. Technically, a malicious payload may also be embedded in Request-Header, although this kind of attacks has not been observed yet. In this work, we assume an attacker can use any request method and embed the malicious code in any field.

4. URI decoder

The specification for URLs limits the allowed characters in a Request-URI to only a subset of the ASCII character set. This means that the query parameters of a request-URI beyond this subset should be encoded. Because a malicious payload may be embedded in the request-URI as a request parameter, the first step of SigFree is to decode the request-URI.

5. ASCII Filter.

Malicious executable code is normally binary strings. In order to guarantee the throughput and response time of the protected web system, if the query parameters of the request-URI and request-body of a request are both printable ASCII ranging from 20-7E in hex, SigFree allows the request to pass we will discuss a special type of executable codes called alphanumeric shell codes that actually use printable ASCII) .

6. Instruction sequences distiller (ISD).

This module distills all possible instruction sequences from the query parameters of Request-URI and Request-Body (if the request has one). Instruction sequences analyzer (ISA). Using all the instruction sequences distilled from the instruction sequences distiller as the inputs, this module analyzes these instruction sequences to determine whether one of them is (a fragment of) a program.

3.1 Hardware and Software requirements

a) S/W Specification

- Operating System : -windows XP/Window Vista/Windows 7.
- Development End(Programming Languages):- .net
- Database Server : -SQL Server 2008.

3.2 Design and Implementation Constraints

SigFree cannot fully handle the branch-function-based obfuscation. Second, SigFree cannot fully handle self-modifying code. Self-modifying code is a piece of code that dynamically modifies itself at runtime and could make SigFree mistakenly exclude all its instruction sequences

3.3 Assumptions and Dependencies

In this description, we focus on buffer overflow attacks whose payloads contain executable code in machine language, and we assume normal requests do not contain executable machine code. A normal request may contain any data, parameters, or even a SQL statement. Note that although SQL statements are executable in the application level, they cannot be executed directly by a CPU. As such, SQL statements are not viewed as executable in our model. Application level attacks such as data manipulation and SQL injection are out of the scope.

Though SigFree is a generic technique which can be applied to any instruction set, for concreteness we assume the web server runs the Intel IA32 instruction set, the most popular instruction set running inside a web server today.