SIGFREE - ATTACK BLOCKER

N.K.Sethu[1], M. Jancyrani Malli [2],
[1,2]Assistant Professor, Dr. Sivanthi Aditanar College of Engineering, Tiruchendur

ABSTRACT

\Buffer overflow is one of the most serious vulnerabilities in computer systems. Buffer overflow vulnerability is a main cause for most of the cyber attacks such as server breaking-in, worms, zombies, and botnets. A buffer overflow occurs during program execution when a fixed-size buffer had too much data copied into it and it which causes the data to overwrite into adjacent memory locations, and depending on what is stored there, the behavior of the program itself might be affected. Buffer overflow attacks do not always carry binary code, it also attack such as stack smashing probably count in the real world. To overcome all above problem SigFree, an online buffer overflow attack blocker to protect internet services is proposed. Sigfree uses a novel technique called Code abstraction to avoid buffer overflow attacks. Code abstraction first uses data flow anomaly to prune useless instructions in an instruction sequence, then compares the number of useful instructions to a threshold to determine if this instruction sequence contains code. Sigfree can also be implemented as a proxy to protect web servers.

I. Introduction:

Buffer overflow vulnerability is a root cause for most of the cyber attacks such as server breakingin, worms, zombies, and botnets. A buffer overflow occurs during program execution when a fixed-size buffer has had too much data copied into it. Code detection algorithm is used to detect the buffer overflow attacks but it is done using signatures. O(N) algorithm is used, where N is the byte length of the message but in this some data bytes may be mistakenly decoded as instructions. Worm signatures can be generated and used to block buffer overflow attack packets. Compiler extension, Defense-side obfuscation, Os and Hardware modification, defenses may cause substantial changes to existing server OSes, application software, and hardware, thus they are not transparent. Capturing Code running symptoms of buffer overflow attacks generally cause process to be terminated.

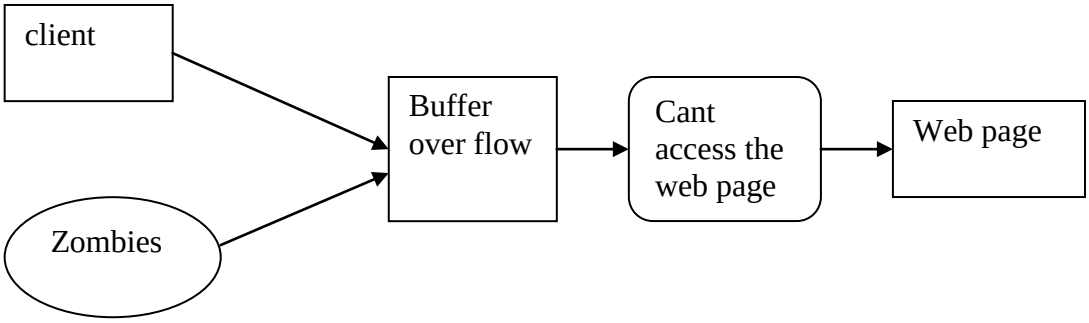
Finding bugs in source code defenses need source code, but source code is unavailable to many legacy applications. Defense-side obfuscation defenses can be very secure, but they either suffer from significant runtime overhead or need special auditing or diagnosis facilities, which are not commonly available in commercial services. As a result, this defense has limited transparency and potential for economical deployment. In Existing system the buffer overflow defenses are Compiler extension, Defense-side obfuscation, Os modification, Hardware modification, Capturing code running symptoms and Finding bugs in source code. In Existing system, worm signatures can be generated and used to block buffer overflow attack packets but it is also limited under the above four requirement and introduce maintenance overhead. O(N) algorithm is used, where N is the byte length of the message but in this some data bytes may be mistakenly decoded as instructions.

Existing System are limited in meeting four requirements:

Compiler extension, Defense-side obfuscation, Os and Hardware modification, defenses may cause substantial changes to existing server OSes, application software, and hardware, thus they are not transparent. Capturing Code running symptoms of buffer overflow attacks generally cause process to be terminated.

Finding bugs in source code defenses need source code, but source code is unavailable to many legacy applications. Defense-side obfuscation defenses can be very secure, but they either suffer from significant runtime overhead or need special auditing or diagnosis facilities, which are not commonly available in commercial services. As a result, this defense has limited transparency and potential for economical deployment.

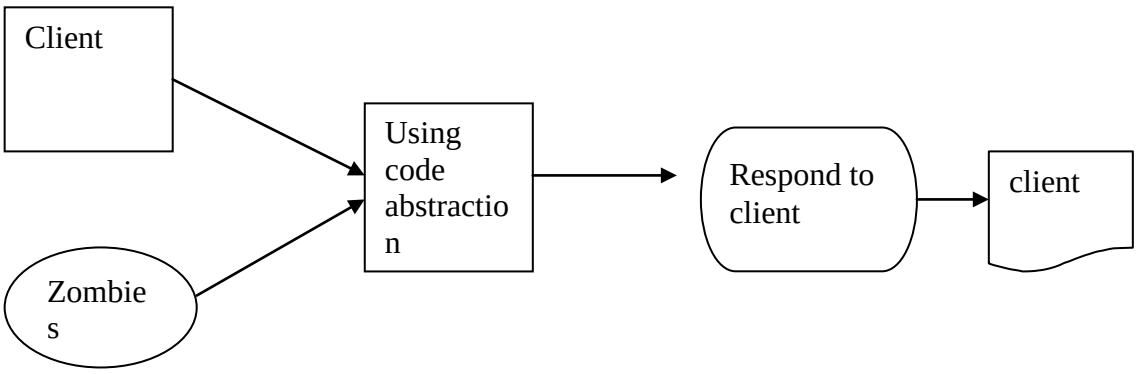
DIAGRAMATIC REPRESENTATION OF EXISTING SYSTEM



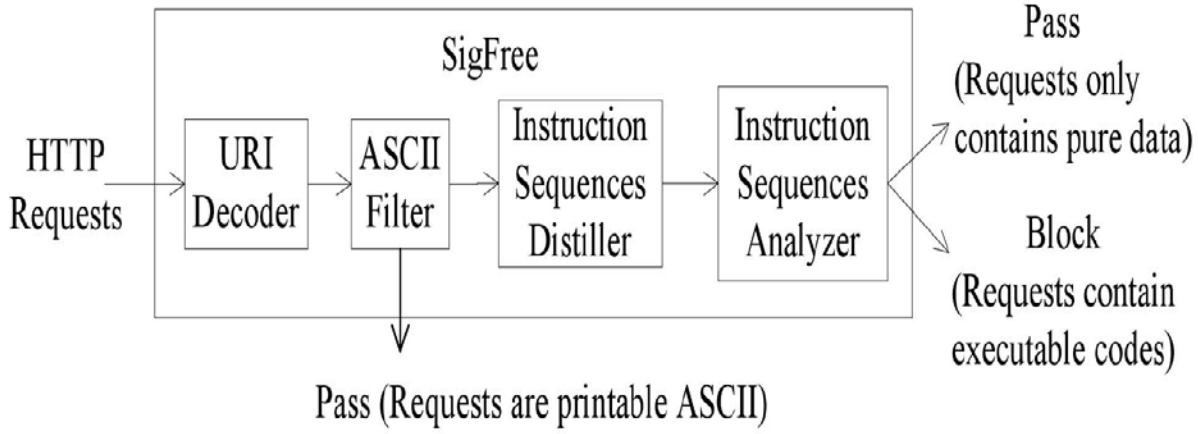
II. PROPOSED SYSTEM

Sigfree uses a novel technique called Code-abstraction to avoid buffer overflow attacks and none of the data bytes will be decoded. Sigfree easily Prevent and detect the buffer overflows. Machine code analysis has three main security purpose: Malware detection, To analyze and identify the code contained in buffer overflow attacks and to analyze obfuscated buffer. Worm Detection and Signature Generation occur easily Because buffer overflow is a key target of worms when they propagate from one host to another. Instruction sequence distiller distills instruction sequences from requests, followed by several pruning techniques to reduce the processing overhead of instruction sequence analyzer. Distilled instructions sequence maybe a sequence of random instructions or a fragment of a program in machine language. So Instruction sequence analyzer uses two methods to differentiate these instructions. Former is Exploits the operating system characteristics of a program and the later is Exploits the data flow characteristics of a program. Proposed system meets those four requirements as follows SigFree is signature free, thus it can block new and unknown buffer overflow attacks. Without relying on string matching, SigFree is immunized from most attack-side obfuscation methods. SigFree uses generic code-data separation criteria instead of limited rules. This feature separates SigFree an independent work that tries to detect code embedded packets. Transparency. SigFree is an out-of-the-box solution that requires no server side changes. SigFree is an economical deployment with very low maintenance cost, which can be well justified by the aforementioned features

DIAGRAMATIC REPRESENTATION OF PROPOSED SYSTEM



ARCHITECTURE DIAGRAM



III.EXPERIMENT

It comprises of following modules

- Design the environment
- Buffer overflow analysis
- Intrusion identification
- Block the intruder packets

Design the environment:

In this the client server network is being developed. The user will give request to the server. It should response the user request. The transport layer is entirely designed with buffer transaction. It is having the limited number of size to certain extent so that the process can be made effectively to reach the desired output. It is used to transfer from the server to the client globally or even locally. We also design intruder data set and hacker programs into the network. The database is to be trained so that they can any combination of output can be produced and prove the efficiency.

Buffer overflows analysis:

In this module two types of algorithm are implemented. They are Linear sweep and Recursive Traversal

These algorithms are used for forwarding the packets and recovering the packets. Here we are classifying the packets depending upon the packets types. We analysis the buffer overflow and traffic in the selected path in the network.

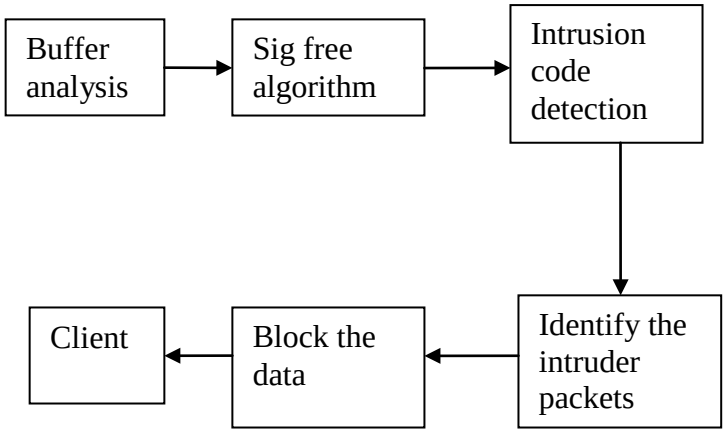
Intrusion identification:

In this module we will implement Sig free method for identify the instruction packets. We will combine the test data with trained data in the decision tree. We implement the code detection algorithm in Sig free method. It is used for identify the intrusion packets in the network.

Block the intruder packets:

We will implement an online signature-free out-of the- box blocker that can filter code-injection buffer overflow attack messages, one of the most serious cyber security threats. We will block the injection packets in the network based on out of the box blocker. The out of the box blocker is based on the Sig free method.

DATA FLOW DIAGRAM:



REFERENCES:

1. "Detecting and Prevention of Stack Buffer Overflow Attacks", B.A. Kuperman, C.E. Brodley, H. Ozdoganoglu, T.N. Vijaykumar,and A. Jalote, Comm. ACM, vol. 48, no. 11, 2005
2. "Beyond Stack Smashing: Recent Advances in Exploiting Buffer Overruns," IEEE Security and Privacy, vol. 2,no. 4, 2004.
3. "Countering Code-Injection Attacks with Instruction-Set Randomization," G. Kc, A. Keromytis, and V. Prevelakis, Proc. 10th ACM Conf. Computer and Comm. Security (CCS '03), Oct. 2003.
4. "Randomized Instruction Set Emulation to Disrupt Binary Code Injection Attacks," E. Barrantes, D. Ackley, T. Palmer, D. Stefanovic, and D. Zovi,Proc. 10th ACM Conf. Computer and Comm. Security (CCS '03), Oct. 2003.
5. "Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software," J. Newsome and D. Song Proc. 12th Ann. Network and Distributed System Security Symp. (NDSS), 2005.