Network and Complex Systems
ISSN 2224-610X (Paper) ISSN 2225-0603 (Online)
Vol 2, No.1, 2012


# Sans Signature Buffer Overflow Blocker

G. Prisilla Jayanthi

Holy Mary Institute Of Technology & Science

Bogaram ,Keesara ,RRDist

Mobile :9393316810 Email: prisillajayanthi@yahoo.co.in

Ambika Prasad Mohanty

Senior Consultant, Infotech Enterprises,

Panjagutta, Hyderabad -500082

AmbikaPrasad.Mohanty@infotech-enterprises.com

**Abstract**

The objective of Sans Signature buffer overflow blocker mainly is to intercept communications between a server and client, analyse the contents for the presence of executable code and prevent the code reaching the server. In this project, Sans Signature is a signature free approach, which can identify and block new and unknown buffer overflow attacks. The system can intercept the data coming via various channels before the server receives the packets. Typically, the data exchanged with a standard application is the data related to the transaction. Therefore, the presence of executable code along with data is something unwarranted. This system will analyze the incoming of the data, check is it contains any executable code. If the executable code is found, the packet is dropped. If the data packet is found to be safe, it is allowed to pass through. The payload or data is analyzed at the application layer called Proxy based Sans Signature. The system has been designed to identify certain executable pattern that is considered harmful. It also has a thresh hold limit beyond which, the packet will be considered to be discarded. Given the intelligence of the algorithm, it prevents most of the buffer overflow attacks. The system can handle the packet analysis in a transparent manner, thus making it suitable for deployment at Firewall/Application Gateway level. Hence, it is quite powerful and efficient with very low deployment and maintenance cost.

**Keywords:** buffer overflow attacks, code-injection attacks, Defense-side obfuscation

## 1. Introduction

In computer system, buffer overflow is one of the most serious vulnerabilities. It is the root 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 has had too much of data copied into it. This 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. A buffer overflow attack may corrupt control flow or data without injecting code such as return-to-libc attacks and data-pointer modification.

### 1.1 **Existing System**

**STILL**, a real-time, out-of-the-box, signature-free, remote exploit binary code injection attack blocker to protect web servers. STILL is motivated by an important observation that the request messages to web servers are exclusively data and not binary executable code. Since remote exploits are typically binary executable code, this observation indicates that if we can precisely distinguish (service requesting) messages that contain binary code from those that do not contain any binary code, we can protect web servers as well as other Internet services (which accept data only) from binary code-injection attacks by blocking the messages that contain binary code. An application layer proxy-based STILL is deployed between the web server and the corresponding firewall to protect web servers. STILL (including static taint

analysis and initialization analysis) detect not only unobfuscated exploit code, traditional polymorphic and metamorphic exploit code, but also self-modifying and indirect jump obfuscation code that could easily defeat previous static analysis approaches. Indeed, STILL is robust to almost all anti-signature, anti-static-analysis and anti-emulation obfuscation. STILL is signature free, thus it can block new and unknown remote code injection attacks such as zero-day exploit code.

The new techniques used previously to detect self-modifying and indirect jump exploit code are called static taint analysis and initialization analysis. We observe that self-modifying and indirect jump exploit code first need acquire the absolute address of payload. Accordingly, we first try to find the piece of code which acquires the absolute address of pay-load at runtime from an instruction sequence. The variable which holds the absolute address will be marked tainted. Then, we use the static taint analysis approach to track the tainted values and detect whether tainted data are used in the ways that could indicate the presence of self-modifying and indirect jump exploit code. A tainted variable is propagated to a new tainted variable by data transfer instructions that move data (e.g., push, pop, move) and data operation instructions that perform arithmetic or bit-logic operations on data (e.g., add, sub, xor). For data transfer instructions, the destination operand will be tainted if and only if the source operand is tainted. For data operation instructions, the destination operand will be tainted if and only if either source or destination operand is tainted.

**Address Space Layout Randomization (ASLR)** is a main component of PaX . Address-space randomization, can detect exploitation of all memory errors. Instruction set randomization can detect all code injection attacks. Nevertheless, when these approaches detect an attack, the victim process is typically terminated. “Repeated attacks will require repeated and expensive application restarts, effectively rendering the service unavailable.”

Detection of Data Flow Anomalies There are static or dynamic methods to detect data flow anomalies in the software reliability and testing field. Static methods are not suitable in our case due to its slow speed; dynamic methods are not suitable either due to the need for real execution of a program with some inputs.

## 2. Proposed System

Sans Signature is a *generic* approach which does not require any pre-known patterns. Then, it uses the found patterns and a data flow analysis technique called program slicing to analyze the packet’s payload to see if the packet really contains code .

Besides, they used a special rule to detect polymorphic exploit code which contains a loop. Although they mentioned that the above rules are initial sets and may require updating with time, it is always possible for attackers to bypass those pre-known rules. Moreover, more rules mean more overhead and longer latency in filtering packets. In contrast, Sans Signature exploits a different data flow analysis technique, which is much harder for exploit code to evade.

### 2.1 Performance Evaluation

#### 2.1.1 Proxy –Based Sans Signature

To evaluate the performance impact of Signature free to web servers, we implemented a proxy- based Sans Signature prototype. *Fig.* shows the architecture of Sans Signature.

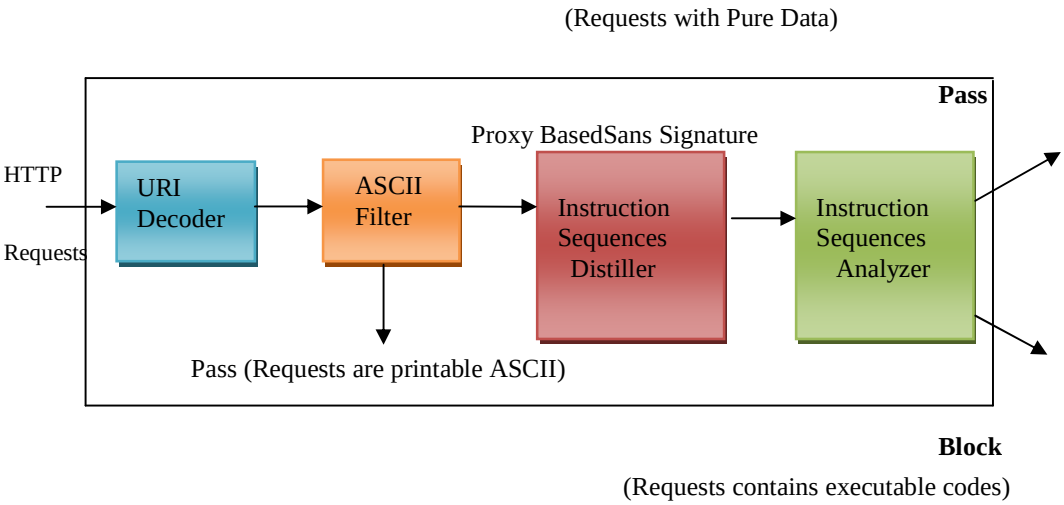



Figure 2.1.1 : The architecture of Sans Signature

**3.Res**

**ult**

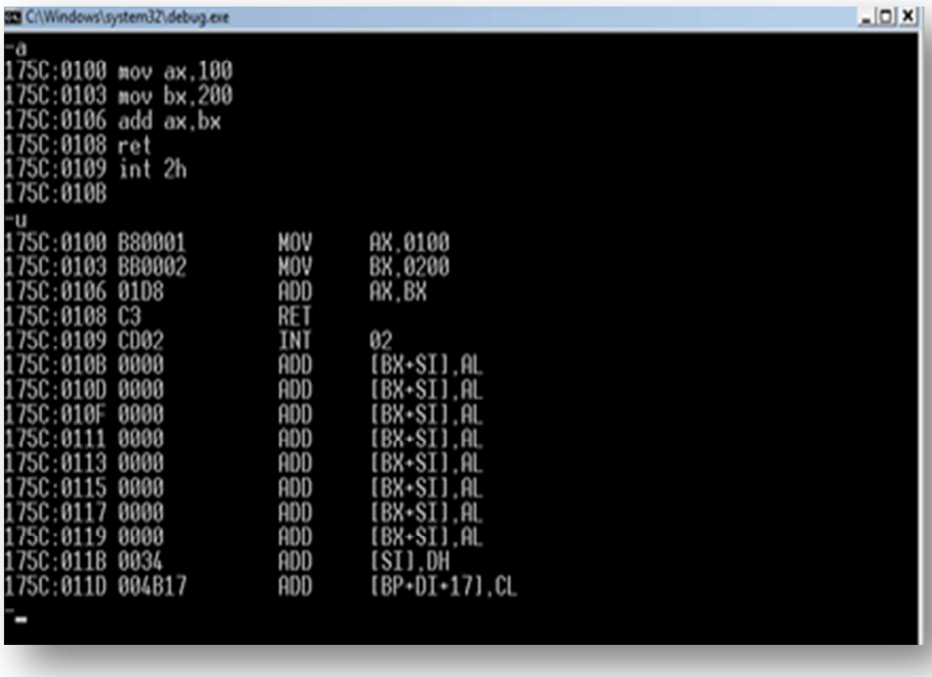



**Figure 1**: To create the opcode values, the machine instructions are entered at debug.exe.
Few machine instructions entered to generate the more opcode values using machine instructions

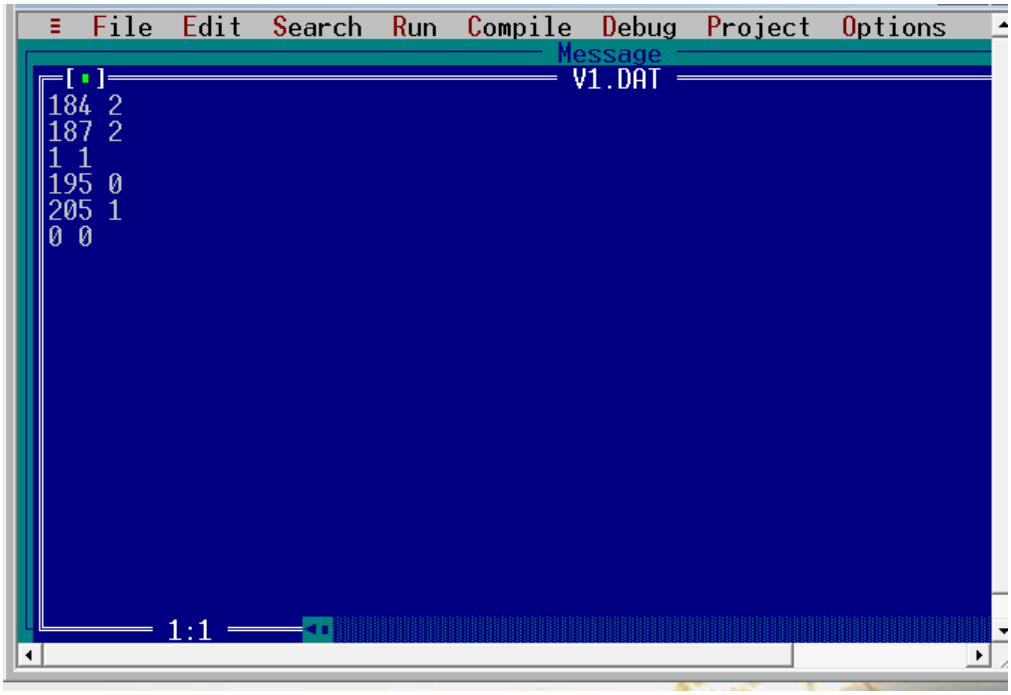
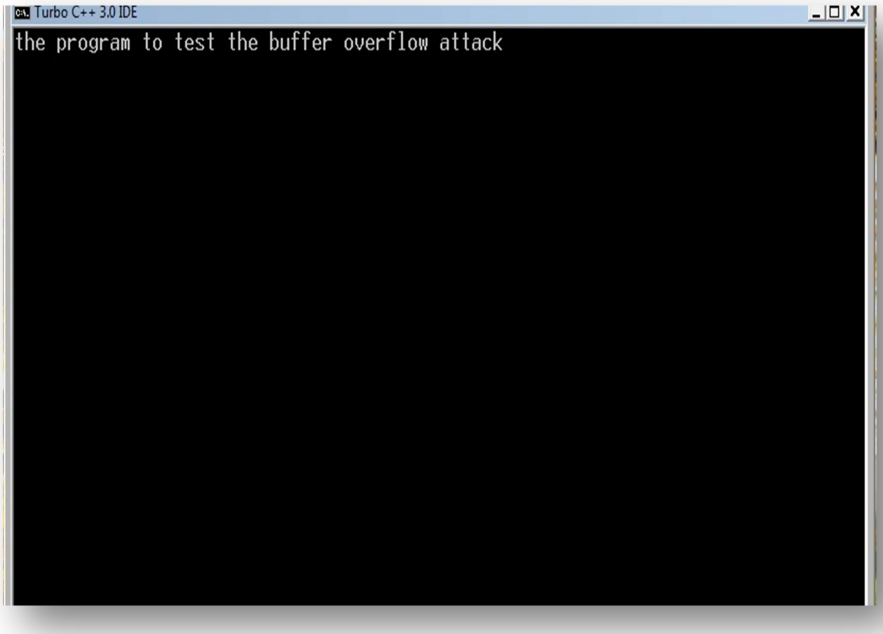



**Figure2 :** The screen shows the opcode values generated and stored in the buffer, and are compared with the input data that enter the system.



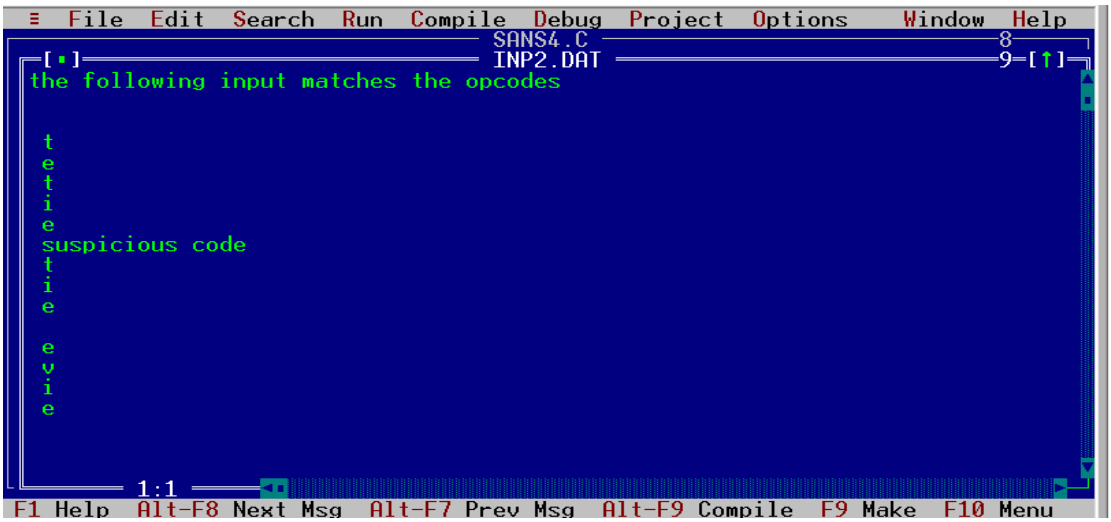




**Figure3:** The screen shows the main page where the input data is entered to check whether any of the data matches with the opcode values

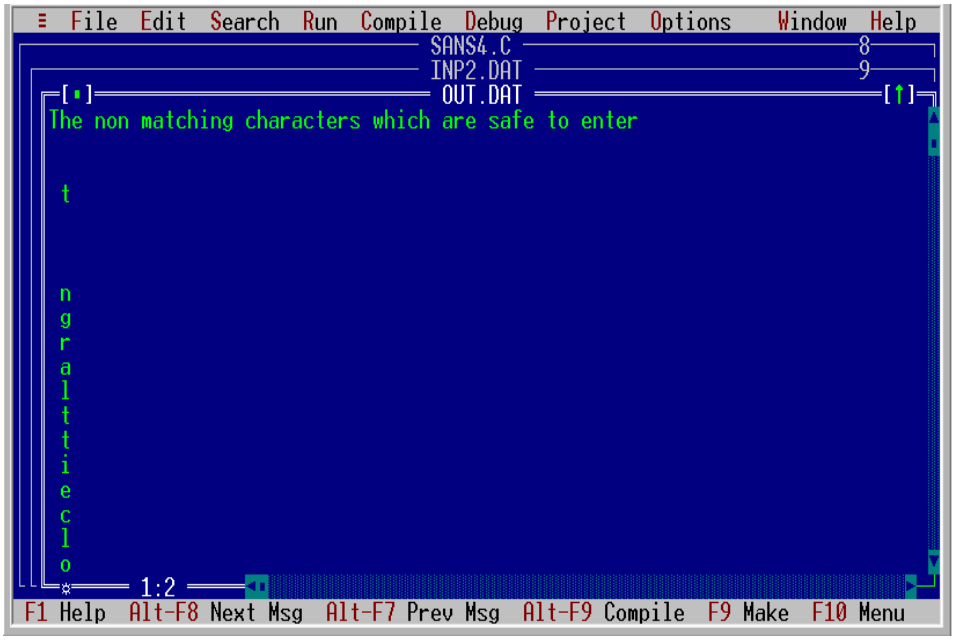



**Figure4**: Screen shows matched characters with the opcodes values which are harmful data (mixed with executable code).

### 3.1 Performance Analysis

The proposed paper is implemented in C Language on a Pentium-III PC with 20 GB hard-disk and 256 MB RAM with apache web server. The propose paper's concepts shows efficient results and has been efficiently tested on different messages.

## 4. Conclusions

The proposed Sans Signature, an online signature-free out-of -the- box blocker that can filter code-injection buffer overflow attack message, one of the most serious cyber security threats. Sans Signature does not require any signatures, thus it can block new unknown attacks. Sans Signature is immunized from most attack-side code obfuscation methods and good for economical Internet wide deployment with little maintenance cost, negligible throughput degradation and low performance overhead and also enhances the complex patterns for instruction.

## 5.References

Xinran Wang, Chi-Chun Pan, Peng Liu, and Sencun Zhu,**"**Sigfree: A Signature –Free Buffer Overflow Attack Blocker" IEEE transactions on dependable and secure computing, vol 7, no. 1, January – March 2010.

Zhaohui Liang, Bin Liang, Luping Li, Wei Chen, Qingqing Kang, Yingqin Gu , "Against Code Injection with System Call Randomisation" 2009 International Conference on Networks Security , wireless communications and trusted Computing.

John J. Donovan , "Systems Programming ", Tata McGraw- Hill publishing company limited.

Kenneth J. Ayala , "The 8086 Microprocessor Programming and Interfacing the PC" , an International Thompson Publishing company.

Yu-Chang Liu, Glenn A. Gibson., "Micro Computer System : The 8086/8088 family Architecture, Programming and Design" .

Brain W. Kernighan, Dennis M. Ritchie,"The C programming Language" , Tata McGraw- Hill publishing.

Benjamin Schwarz Saumya Debray Gregory Andrews ,"Disassembly of Executable Code Revisited* " ,Department of Computer Science University of Arizona *Tucson, AZ 85721*