American Journal of Engineering Research (AJER)
e-ISSN : 2320-0847 p-ISSN : 2320-0936
Volume-02, Issue-12, pp-472-483
www.ajer.org

Research Paper Open Access

Design And Development Of Suitable Software Engineering Techniques To Detect And Manage Buffer-Overflows In Computer Systems

Mohd Ayaz Uddin[1], Mirza Younus Ali Baig[2], Prof.Dr.G.Manoj Someswar[3]

1. Working as Assistant Professor in the Department of Information Technology, Nawab Shah Alam Khan College of Engineering & Technology (Affiliated to JNTUH), New Malakpet, Hyderabad-500024, A.P., India.

2. Working as Assistant Professor in the Department of Information Technology, Nawab Shah Alam Khan College of Engineering & Technology (Affiliated to JNTUH), New Malakpet, Hyderabad-500024, A.P., India.

3. Working as Professor, HOD & DEAN (Research) in the Department of Computer Science & Engineering, Nawab Shah Alam Khan College of Engineering & Technology (Affiliated to JNTUH), New Malakpet, Hyderabad-500024, A.P., India.

Abstract: - 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 host. "A buffer overflow occurs during program execution when a fixed-size buffer has had too much 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." (Note that the buffer could be in stack or heap.) Although taking a broader viewpoint, buffer overflow attacks do not always carry code in their attacking requests (or packets) 1, code-injection buffer overflow attacks such as stack smashing count for probably most of the buffer overflow attacks that have happened in the real world. Although tons of research has been done to tackle buffer overflow attacks, existing defenses are still quite limited in meeting four highly-desired requirements:

(R1) simplicity in maintenance
(R2) transparency to existing (legacy) server OS, application software, and hardware
(R3) resiliency to obfuscation
(R4) economical Internet wide deployment.

Keywords: - *Code-injection Buffer Overflow attack, C Range Error Detector, Libsafe, Libverify, Safe Pointer, Data Execution Prevention, Solar Designer, Stack guard*

I. INTRODUCTION

As a result, although several very secure solutions have been proposed, they are not pervasively deployed, and a considerable number of buffer overflow attacks continue to succeed on a daily basis. To see how existing defenses are limited in meeting these four requirements, let us break down the existing buffer overflow defenses into six classes which are as follows:

(1A) Finding bugs in source code.
(1B) Compiler extensions.
(1C) OS modifications.
(1D) Hardware modifications.
(1E) Defense-side obfuscation.

(1F) Capturing code running symptoms of buffer overflow attacks. (Note that the above list does not include binary code analysis based defenses which we will address shortly.)[6] We may briefly summarize the limitations of these defenses in terms of the four requirements as follows. (a) Class 1B, 1C, 1D, and 1E defenses may cause substantial changes to existing (legacy) server OSes, application software, and hardwares, thus they are not transparent. Moreover, Class 1E defenses generally cause processes to be terminated. As a result, many businesses do not view these changes and the process termination overhead as economical deployment. (b)

Class 1F defenses can be very secure, but they either suffer from significant run time overhead or need special auditing or diagnosis facilities which are not commonly available in commercial services.[7]The idea of SigFree is motivated by an important observation that "the nature of communication to and from network services is predominantly or exclusively data and not executable code." In particular, as summarized in , (a) on Windows platforms, most web servers (port 80) accept data only; remote access services (ports 111, 137, 138, 139) accept data only; Microsoft SQL Servers (port 1434) accept data only; workstation services (ports 139 and 445) accept data only. (b) On Linux platforms, most Apache web servers (port 80) accept data only; BIND (port 53) accepts data only; SNMP (port 161) accepts data only; most Mail Transport (port 25) accepts data only; Database servers (Oracle, MySQL, PostgreSQL) at ports 1521, 3306 and 5432 accept data only. Since remote exploits are typically executable code, this observation indicates that if we can precisely distinguish (service requesting) messages that contain code from those that do not contain any code, we can protect most Internet services (which accept data only) from code-injection buffer overflow attacks by blocking the messages that contain code.[5]The merits of SigFree are summarized below. They show that SigFree has taken a main step forward in meeting the four requirements afore mentioned.

a. SigFree is signature free, thus it can block new and unknown buffer overflow attacks
b. Without relying on string-matching, SigFree is immunized from most attack-side obfuscation methods.
c. SigFree uses generic code-data separation criteria instead of limited rules. This feature separates SigFree from, an independent work that tries to detect code-embedded packets.
d. Transparency. SigFree is an out-of-the-box solution that requires no server side changes.
e. SigFree has negligible through output degradation

II. ANALYSIS

Software engineering is an extremely difficult task and of all software creation Related professions, software architects have quite possibly the most difficult task. Initially, software architects were only responsible for the high-level design of the products. More often than not this included protocol selection, third-party component evaluation and selection, and communication medium selection. We make no argument here that these are all valuable and necessary objectives for any architect, but today the job is much more difficult. It requires an intimate knowledge of operating systems, software languages, and their inherent advantages and disadvantages in regards to different platforms. Additionally, software architects face increasing pressure to design flexible software that is impenetrable to wily hackers. A near impossible feat in itself.

SQL attacks, authentication brute-forcing techniques, directory traversals, cookie poisoning, cross-site scripting, and mere logic bug attacks when analyzed via attack packets and system responses are shockingly similar to those of normal or non-malicious HTTP requests.[8]

Today, over 70 percent of attacks against a company's network come at the "Application layer," not the Network or System layer.—The Gartner Group[4].

Buffer overflows are the most feared of vulnerabilities from a software vendor's perspective. They commonly lead to Internet worms, automated tools to assist in exploitation, and intrusion attempts. With the proper knowledge, finding and writing exploits for buffer overflows is not an impossible task and can lead to quick fame especially if the vulnerability has high impact and a large user base.[3]

III. EXISTING SYSTEM

Detection of Data Flow Anomalies There is 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]

It takes considerable effort to prevent buffer overflows. On the one hand static methods produce false / negative results, which cause manual corrections in the source code by the developer. On the other hand instrumentation methods have lot of overhead and they are not transparent. Stack based methods does not prevent from all attacks. Hardware methods provide less overhead but need to have deeper architectural changes.

The dynamic methods are too expensive to protect the systems against buffer overflow attacks.

Methods used in existing system are

a. Stack based: Adding redundant information / routines to protect the stack or parts of stack.
b. Instrumentation: Replacing of standard functions / objects like pointers to equip them with tools.
c. Hardware based: Architecture check for illegal operations and modifications
d. Static: Checking the source code for known vulnerable functions, do flow analysis check the correct boundaries, use of heuristics. [1]
e. Operation system based: Declare the stack as non-executable to prevent code execution

IV. STACK GUARD

A simple approach to protect programs against stack smashing and with little modification against EBP overflows. This is achieved by a compiler extension that adds so called canary values before the EIP saved at the function prologue (see Figure 3.1). Before the return of a protected function is executed, the canary values are checked. An attacker could guess the canary values. This is quite hard, if the values are chosen randomly for each guarded function, but it is also possible to choose a canary value made of terminator characters, which makes every string/file copy function to stop at the canary value. So even restoring the canary value would not lead to a successful program flow detour. Another way to thwart Stackguard is to find a pointer to overflow that it points to the address of the saved EIP and use that pointer as target for a copy function. This way the EIP is overwritten without modifying the canary values.[9]

Overhead produced is moderate with up to 125% and that this method is not transparent, meaning that the source code is needed for recompilation. This fact makes Stackguard useless for many legacy software products on the market, because they are not open source.

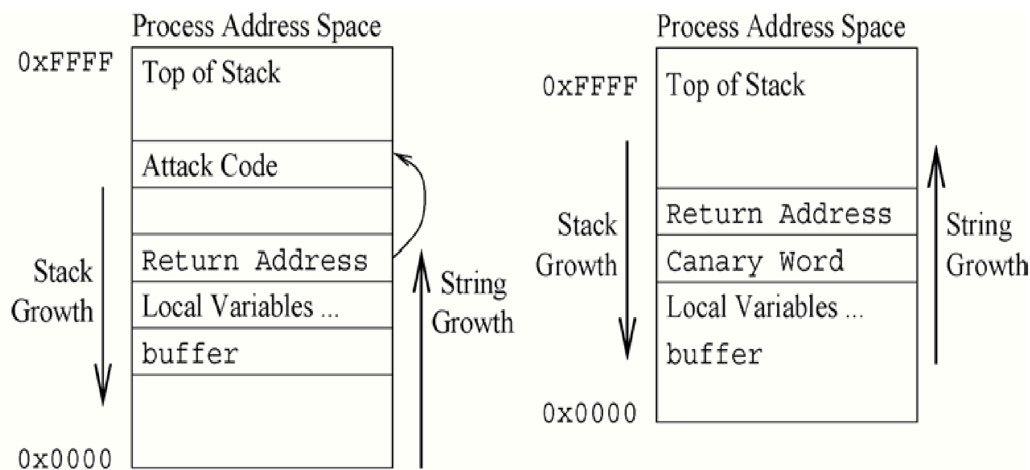

Figure 1: Stack layout using stack guard

V. LIBSAFE AND LIBVERIFY

Two methods that should protect against buffer overflow attacks. The first method is libsafe, a transparent approach set up in a DLL that replaces standard (vulnerable) functions by standard bounds checked functions (e.g. strcpy could be replaced by strncpy). The upper limit of the bounds is calculated based on the EBP, so the maximum amount written to a buffer is the size of the stackframe. This method only works if the EBP can be determined, since there exist compiler options that make this impossible; further compatibility issues could arise with legacy software.

VI. INSTRUMENTATION

Safe pointer

Safe pointer structure is to detect all pointer and array access errors. Meaning that both, temporal and spatial errors are detected.

The structure consists of five entries:

a. Value (the value of the safe pointer, it may contain any expressible address)
b. Base (the base address of the referent)
c. Size (the size of the referent in bytes)
d. Storage class (either Heap, Local or Global)
e. Capability (unique capability. Predefined capabilities are forever and never, else it could be an enumerated number as long as its value is unique)

Base and size are spatial attributes capability and storage class are temporal attributes. The capability is also stored in a capability store when it is issued and deleted if the storage is freed or when the procedure invocation returns. This ensures that storage that is not available (like freed heap allocated memory) is not accessed anymore. The transformation of a program from unsafe to safe pointers involves pointer conversion (to extend all pointer definitions), check insertion (to instrument the program to detect memory access errors) and operator conversion (to generate and maintain object attributes).

C Range Error Detector (CRED)

CRED is the idea to replace every out-of-bounds (OOB) pointer value with the address of a special OOB object created for that value. To realize this, a data structure called object table collects the base address, and size information of static, heap and stack objects. To determine, if an address is in-bounds, the checker first locates the referent object by comparing the in-bounds pointer with the base and size information stored in the object table. Then, it checks if the new address falls within the extent of the referent object. If an object is out-of-bounds, an OOB object is created in the heap that contains the OOB address value and the referent object. If the OOB value is used as an address it is replaced by the actual OOB address. The OOB objects are entered into an out-of-bounds object hash table, so it is easy to check if a pointer points to an OOB object by consulting the hash table. The hash table is only consulted if the checker is not able to find the referent object in the object table or cannot identify the object as unchecked. Arithmetic and comparison operations to OOB objects are legal, since the referent object and its value is retrieved from the OOB object. But if a pointer is dereferencing, it is checked if the object is in the object table or if it is unchecked else the operation is illegal. Buffer overflows are not prevented but the goal is thwarted because copy functions need to dereference the OOB value, the program is halted before more damage happens. This fact could be still used as DoS attack, since the program (service) is halted and needs to be restarted or even worst if it has to be re-administrated. This method works, by comparing the non instrumented code, the CRED instrumented code and a code where the instrumentation is which is the base for CRED. Since recompilation is needed, this method is not transparent. But the instrumented code is fully compatible to non-instrumented code. The overhead of this approach ranges from 1% to 130%, but it do not show how certain kind of buffer overflow attacks, like signed/unsigned and off-by-one overflows are handled. The same arguments as on the safe pointers in the previous section can be applied here.

VII. HARDWARE BASED

The approach deals with an architectural change implementing a Secure Return Address Stack (SRAS), which is a cyclic, finite LIFO structure that stores return addresses. At a return call the last SRAS entry is compared with the return address from the stack and if the comparison yields that the return address was altered the processor can terminate the process and inform the operation system or continues the execution based on the SRAS return address. Since the SRAS is finite and cyclic, n/2 of the SRAS content has to be swapped on an under- or overflow.

Two methods:

a. OS-managed SRAS swapping. The operation system executes code that transfers contents to or from memory which is mapped to physical ages that can only be accessed by the kernel
b. Processor-managed SRAS swapping. The processor maintains two pointers to two physical pages that contain spilled SRAS addresses and a counter that indicates the space left in the pages. If the Pages over or underflow, the OS is invoked to de-/allocate pages, else the processor can directly transfer contents to and from the pages without invoking the OS.The problem is that the SRAS is not compatible with non LIFO routines, such as C++ exception handling. This makes it necessary to change the non LIFO routines to LIFO routines or it must be possible to turn off the SRAS protection.

VIII. STATIC

This deals with the idea to comment the source code that LCLint can interpret them and generate a log file which can be used to identify possible vulnerabilities. If a source code is analyzed, LCLint evaluates conditions to fulfill safe execution of the finally compiled program. These conditions are written to the log file so the programmer an check if these conditions are true for every case that could happen while execution. Then the programmer can write control comments into the source to let LCLint what conditions are fulfilled or if LCLint should ignore parts of the source code. These way errors can be found before compilation, but this method has certain shortcoming. Since it is not possible to efficiently determine invariants, to take advantage of idioms used typically by C programmers. Since this method is not exact, the rate of false positives grows. Further LCLint is a lightweight checker, meaning that the program flow is also checked using heuristics, since determining all possible program states might need exponential time. The false positives can be commented out, but this means more work to the developers of the software and since heuristics are used, false negatives are produced either. All these facts and the fact that this method is not transparent makes it only suitable for new or small projects. The last aspect we want to point out is that this is the only method so far that produces no overhead, since the compilers skip comments.

IX. OPERATION SYSTEM BASED

Data Execution Prevention (DEP)

With the release of the Service Pack 2 for Windows XP and Service Pack 1 for Windows 2003 a new protection was introduced to machines using these operation systems, the DEP. Microsoft explains a bit how the

DEP works. In cooperation with Intel (Execute Disable bit feature) and AMD (no-execute page-protection processor feature) a new CPU flag was implemented called the NX-Flag. It marks all memory locations in a process as non-executable unless the location explicitly contains executable code. [10] If the machine running with DEP support has no NX-Flag, the DEP can be enforced by the operation system (software enforcement). This protection prevents execution of injected code, if the code was injected in a non executable area. The DEP can be bypassed. Further this method requires that even valid, working processes are (sometimes) recompiled. Another shortcoming is, that this method does not prevent the buffer overflow itself, so attacks like variable attack or BSS/heap overflows are not prevented.

Solar Designer

The Solar Designer patch does nearly the same as the DEP, but it makes the stack non executable. Since Linux needs the executable stack for signal handling, this restricts the normal behavior of Linux. If the attack is able to determine code that would act like a shellcode and execute this code instead of injected code the patch can be bypassed. To conclude, buffer overflows are not prevented, only the code execution. Attacks like the variable attack or BSS/heap overflows are still possible, and heap overflows can also be used to execute arbitrary code.

X. PROPOSED SYSTEM

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.

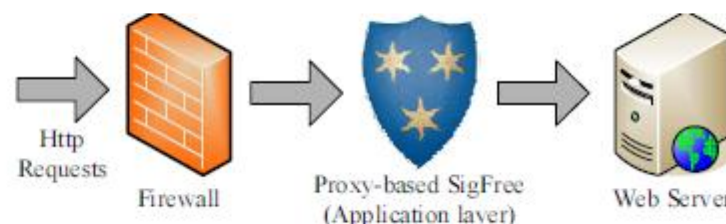

Figure 2: Signature Free prototype

We have implemented a SigFree prototype as a proxy to protect web servers. Our empirical study shows that there exists clean-cut “boundaries” between code embedded payloads and data payloads when our code data separation criteria are applied. We have identified the “boundaries” (or thresholds) and been able to detect/ block all 50 attack packets generated by Meta spoilt framework, all 200 polymorphic shellcode packets generated by two well-known polymorphic shellcode engine ADMmutate and CLET , and worm Slammer, CodeRed and a CodeRed variation, when they are well mixed with various types of data packets. Also, our experiment results show that the throughput degradation caused by SigFree is negligible.

XI. BUFFER OVERLOW VARIANTS

Today buffer overflow attacks are known and well understood. In general every buffer that can be accessed by an attacker might be compromised if vulnerable functions are used. Such variables are located on the stack and heap. The attacks are partitioned as follows:

a. Stack smashing used to execute inject code
b. Variable attack used to modify program state
c. Heap overflow used to execute arbitrary code or to modify the variables
d. Off-by-one a classic programmers error , only one byte is overwritten [11]

XII. INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

a. What data should be given as input?

b. How the data should be arranged or coded?
c. The dialog to guide the operating personnel in providing input.
d. Methods for preparing input validations and steps to follow when error occur.

XIII. OBJECTIVES

Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system. It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow.

XIV. OUTPUT DESIGN

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making. Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.
Select methods for presenting information. Create document, report, or other formats that contain information produced by the system. The output form of an information system should accomplish one or more of the following objectives.

a. Convey information about past activities, current status or projections of the
b. Future.
c. Signal important events, opportunities, problems, or warnings.
d. Trigger an action.
e. Confirm an action.

XV. SYSTEM STUDY

FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.
Three key considerations involved in the feasibility analysis are:

a. ECONOMICAL FEASIBILITY
b. TECHNICAL FEASIBILITY
c. SOCIAL FEASIBILITY

ECONOMICAL FEASIBILITY

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified.Only the customized products had to be purchased. [12]

TECHNICAL FEASIBILITY

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system. [13]

SOCIAL FEASIBILITY

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity.His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the computer system.[14]
In this phase, we understand the software requirement specifications for the research work. We arrange all the required components to develop the project in this phase itself so that we will have a clear idea regarding the requirements before designing the project. Thus we will proceed to the design phase followed by the implementation phase of the project.

XVI. DESIGN ARCHITECTURE

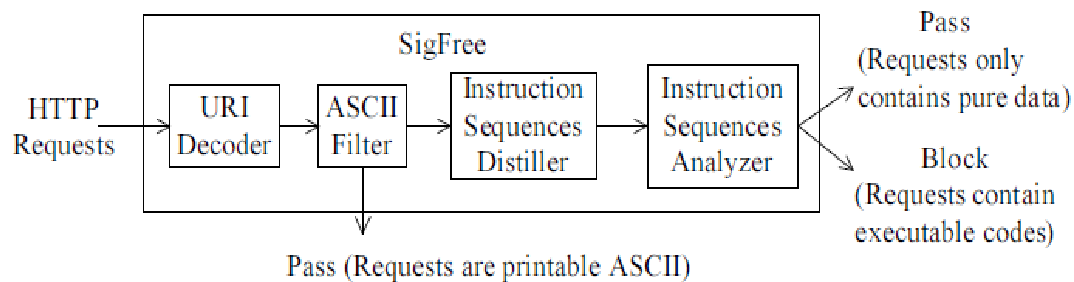

Figure 3: Architecture of SigFree

DATA FLOW DIAGRAM / USE CASE DIAGRAM

The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of the input data to the system, various processing carried out on these data, and the output data is generated by the system. Data flow diagram shows step by step flow.

a. Use case diagram is a description of a set of sequences of actions, including variants that a system performs to yield an observable result of value to an actor.
b. Class diagram that shows a set of classes, interfaces and collaborations and their relationships.
c. Activity diagram is a flowchart, showing flow of control from activity to activity.
d. A component diagrams shows the organization and dependencies among set of components.
e. An interaction diagram that emphasizes the time ordering of messages.

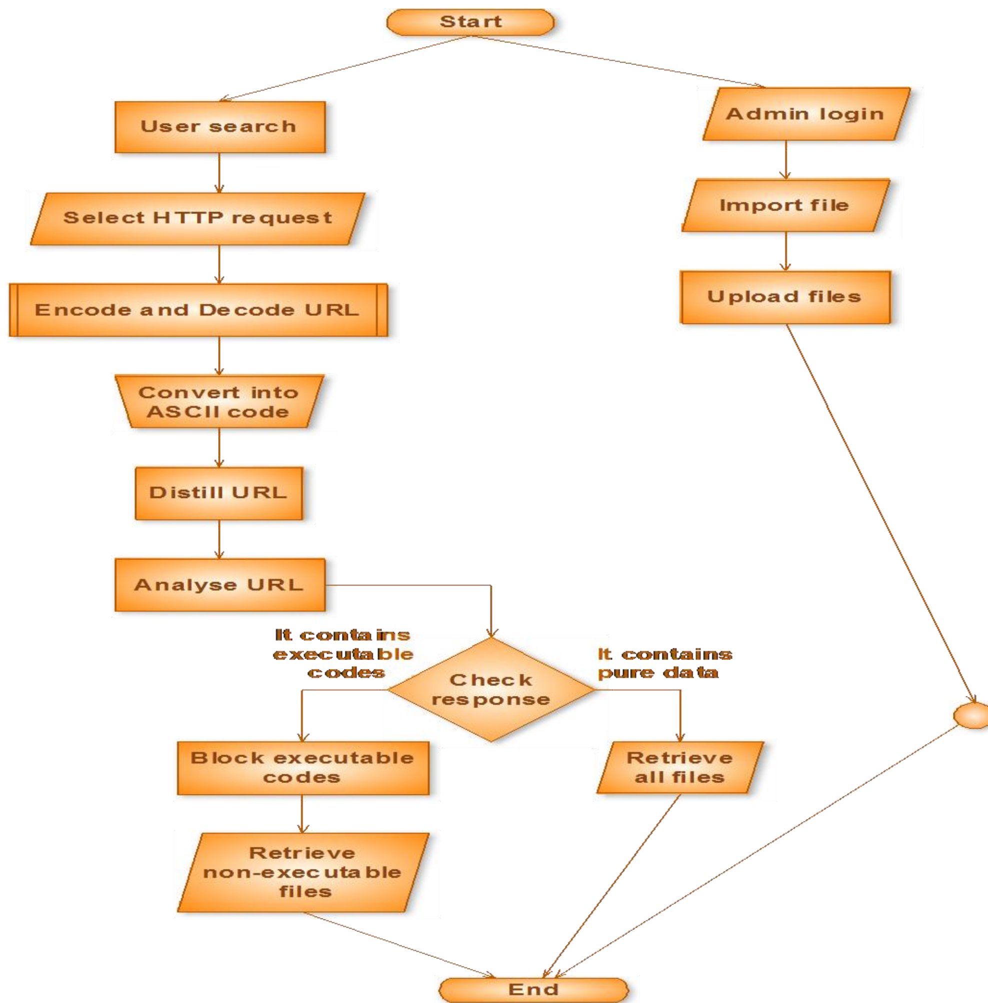

Figure 4: Data Flow Diagram

UML DIAGRAMS

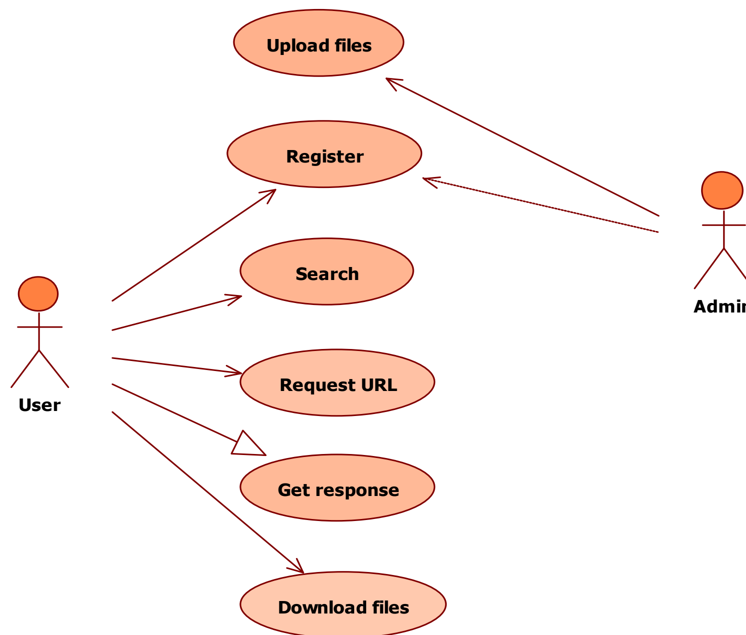

Figure 5: Use Case Diagram

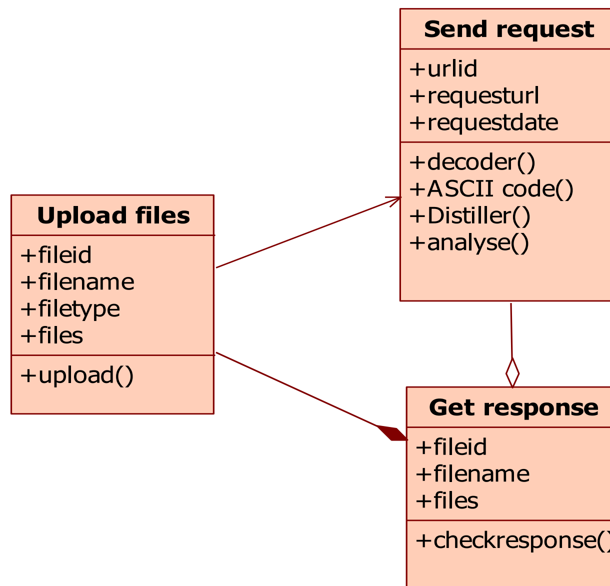

Figure 6: Class Diagram

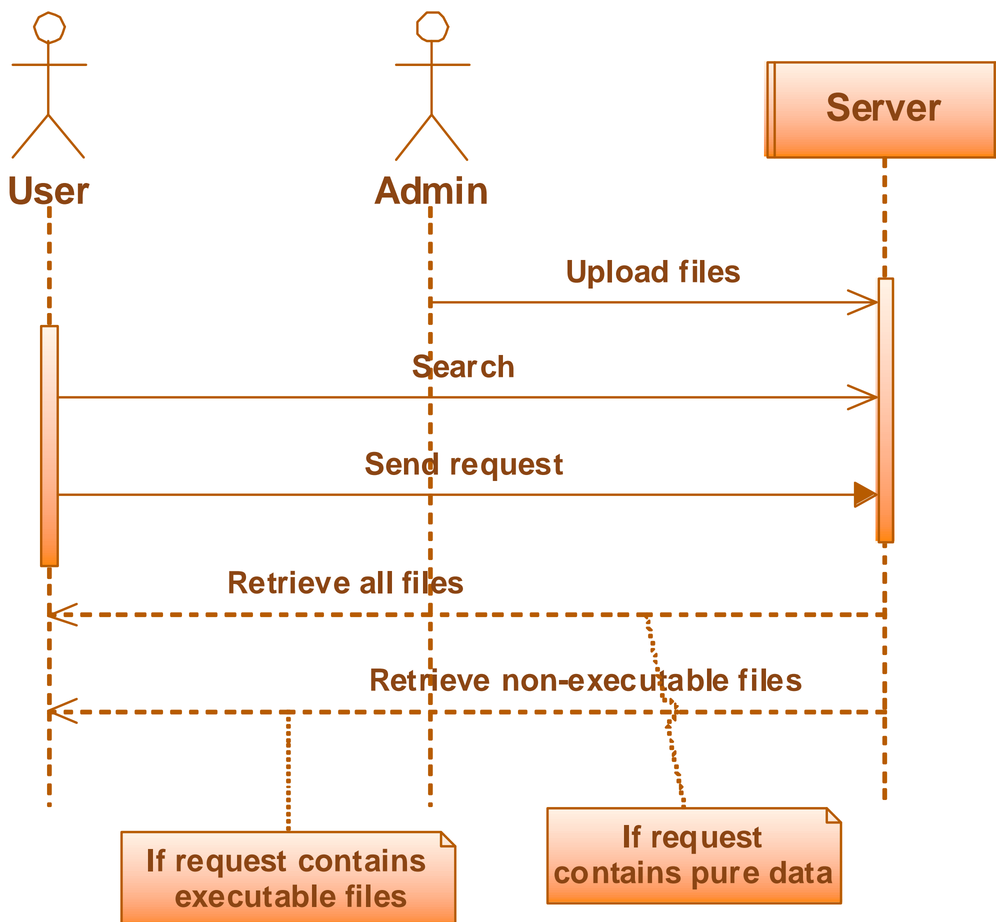

Figure 7: Sequence Diagram

User search
Admin login
Send HTTP request
Import files
Encode and Decode URL
Upload files
Convert into ASCII code
If request contains pure data
Distill and analyse URL
If request contains executable files
Check URL
Block executable files
Retrieve all files
A
Retrieve non-executable files

Figure 8: Activity Diagram

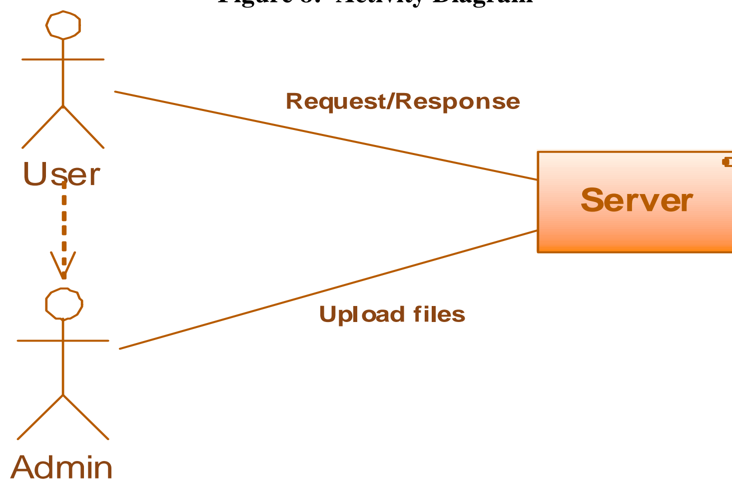

Figure 9: Component Diagram

In this way we can design the layout of the project which is to be implemented during the construction phase. Thus we will have a clear picture of the project before being coded. Hence any necessary enhancements can be made during this phase and coding can be started and program compiled and executed successfully keeping in view of the proposed needs and requirements in the beginning of the research work.

The Testing Process- Overview

The testing process for web engineering begins with tests that exercise content and interface functionality that is immediately visible to end-users. As testing proceeds, aspects of the design architecture and navigation are exercised. The user may or may not be cognizant of these WebApp elements. Finally, the focus shifts to tests that exercise technological capabilities that are not always apparent to end-users—WebApp infrastructure and installation/implementation issues.

a. Content Testing
b. Interface Testing
c. Navigation Testing
d. Component Testing
e. Configuration Testing
f. Performance Testing
g. Security Testing

The following figure shows the testing flow:

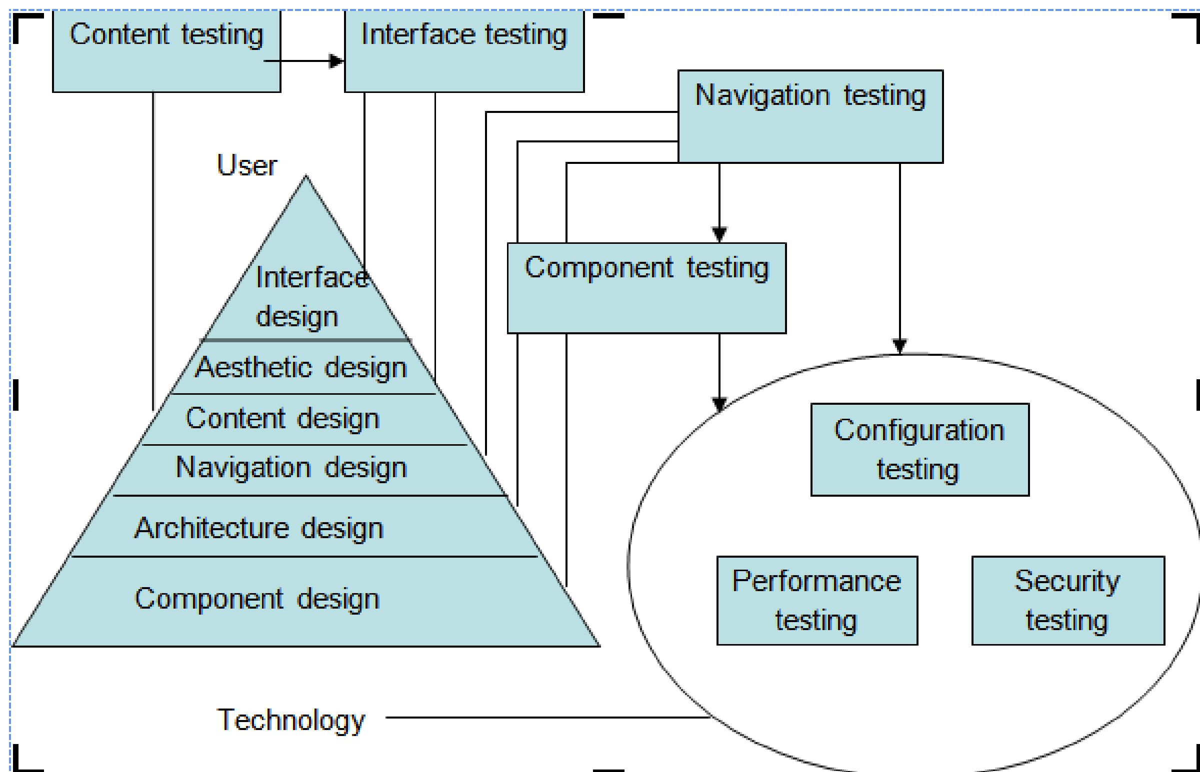

Figure 10: Testing Flows

XVII. RESULTS & CONCLUSION

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. SigFree is immunized from most attack-side code obfuscation methods, good for economical Internet wide deployment with little maintenance cost and negligible throughput degradation and can also handle encrypted SSL messages.

A combination of developer education for defensive programming techniques as well as software reviews is the best initial approach to improving the security of custom software. Secure programming and scripting languages are the only true solution in the fight against software hackers and attackers.

REFERENCES

[1] SigFree: Signature Buffer overflow attack blocker by Xinrang Wang, Chi-Chun Pan, Peng Liu, Senchu Zhu pp 1-3,6-18,47-68
[2] Buffer overrun in jpeg processing (gdi+) could allow code execution 833987 pp 47-68http://www.microsoft.com/technet/security/bulletin/ MS04-028.mspx
[3] Web application vulnerabilities: detect exploit and prevent by Michael cross pp 47-68
[4] Buffer overflows vulnerability diagnosis for commodity software by jiang.pp 6-18
[5] Buffer overflow attacks by James C Foster pp 47-49
[6] Intel ia-32 architect software developer's manual volume 1: Basic architecture.pp 9
[7] Metasploit project. http://www.metasploit.com. pp 47-68
[8] Security advisory: Acrobat and adobe reader plug-in buffer overflow. http://www.adobe.com/support/techdocs/321644.html. pp 13-15
[9] Stunnel – universal ssl wrapper. http://www.stunnel.org.
[10] Symantec security response: back door.hesive. pp 6- 18 http://securityresponse.symantec.com/ avcenter/venc/data/backdoor.hesive.html
[11] Winamp3 buffer overflow. http://www.securityspace.com/ smysecure/ catid.html?id=11530. pp 6-18
[12] Pax documentation. http://pax.grsecurity.net/docs/pax.txt, November 2003. against stack smashing attacks. In Proc. 2000 USENIX Technical Conference (June 2000). pp 14
[13] Professional ASP .NET by Wrox Publications pp 25-46
[14] Effective methods for software testing by William Perry 69-83