INTERNATIONAL JOURNAL OF RESEARCH IN COMPUTER APPLICATION & MANAGEMENT

IJRCM

A Monthly Double-Blind Peer Reviewed (Refereed/Juried) Open Access International e-Journal - Included in the International Serial Directories

Indexed & Listed at:

Ulrich's Periodicals Directory ©, ProQuest, U.S.A., EBSCO Publishing, U.S.A., Cabell's Directories of Publishing Opportunities, U.S.A.,

Open J-Gage, India [link of the same is duly available at Inflibnet of University Grants Commission (U.G.C.)],

Index Copernicus Publishers Panel, Poland with IC Value of 5.09 & number of libraries all around the world.

Circulated all over the world & Google has verified that scholars of more than 2980 Cities in 165 countries/territories are visiting our journal on regular basis.

Ground Floor, Building No. 1041-C-1, Devi Bhawan Bazar, JAGADHRI – 135 003, Yamunanagar, Haryana, INDIA

http://ijrcm.org.in/

CONTENTS

CALL FOR MANUSCRIPTS

We invite unpublished novel, original, empirical and high quality research work pertaining to recent developments & practices in the areas of Computer Science & Applications; Commerce; Business; Finance; Marketing; Human Resource Management; General Management; Banking; Economics; Tourism Administration & Management; Education; Law; Library & Information Science; Defence & Strategic Studies; Electronic Science; Corporate Governance; Industrial Relations; and emerging paradigms in allied subjects like Accounting; Accounting Information Systems; Accounting Theory & Practice; Auditing; Behavioral Accounting; Behavioral Economics; Corporate Finance; Cost Accounting; Econometrics; Economic Development; Economic History; Financial Institutions & Markets; Financial Services; Fiscal Policy; Government & Non Profit Accounting; Industrial Organization; International Economics & Trade; International Finance; Macro Economics; Micro Economics; Rural Economics; Co-operation; Demography: Development Planning; Development Studies; Applied Economics; Development Economics; Business Economics; Monetary Policy; Public Policy Economics; Real Estate; Regional Economics; Political Science; Continuing Education; Labour Welfare; Philosophy; Psychology; Sociology; Tax Accounting; Advertising & Promotion Management; Management Information Systems (MIS); Business Law; Public Responsibility & Ethics; Communication; Direct Marketing; E-Commerce; Global Business; Health Care Administration; Labour Relations & Human Resource Management; Marketing Research; Marketing Theory & Applications; Non-Profit Organizations; Office Administration/Management; Operations Research/Statistics; Organizational Behavior & Theory; Organizational Development; Production/Operations; International Relations; Human Rights & Duties; Public Administration; Population Studies; Purchasing/Materials Management; Retailing; Sales/Selling; Services; Small Business Entrepreneurship; Strategic Management Policy; Technology/Innovation; Tourism & Hospitality; Transportation Distribution; Algorithms; Artificial Intelligence; Compilers & Translation; Computer Aided Design (CAD); Computer Aided Manufacturing; Computer Graphics; Computer Organization & Architecture; Database Structures & Systems; Discrete Structures; Internet; Management Information Systems; Modeling & Simulation; Neural Systems/Neural Networks; Numerical Analysis/Scientific Computing; Object Oriented Programming; Operating Systems; Programming Languages; Robotics; Symbolic & Formal Logic; Web Design and emerging paradigms in allied subjects.

Anybody can submit the **soft copy** of unpublished novel; original; empirical and high quality **research work/manuscript** ***anytime*** in ***M.S. Word format*** after preparing the same as per our **GUIDELINES FOR SUBMISSION**; at our email address i.e. infoijrcm@gmail.com or online by clicking the link **online submission** as given on our website (***FOR ONLINE SUBMISSION, CLICK HERE***).

GUIDELINES FOR SUBMISSION OF MANUSCRIPT

1. **COVERING LETTER FOR SUBMISSION:**

DATED: _____

THE EDITOR
IJRCM

Subject: **SUBMISSION OF MANUSCRIPT IN THE AREA OF .**

(e.g. Finance/Marketing/HRM/General Management/Economics/Psychology/Law/Computer/IT/Engineering/Mathematics/other, please specify)

DEAR SIR/MADAM

Please find my submission of manuscript entitled '_____' for possible publication in your journals.

I hereby affirm that the contents of this manuscript are original. Furthermore, it has neither been published elsewhere in any language fully or partly, nor is it under review for publication elsewhere.

I affirm that all the author (s) have seen and agreed to the submitted version of the manuscript and their inclusion of name (s) as co-author (s).

Also, if my/our manuscript is accepted, I/We agree to comply with the formalities as given on the website of the journal & you are free to publish our contribution in any of your journals.

NAME OF CORRESPONDING AUTHOR:
Designation:
Affiliation with full address, contact numbers & Pin Code:
Residential address with Pin Code:
Mobile Number (s):
Landline Number (s):
E-mail Address:
Alternate E-mail Address:

NOTES:

a) The whole manuscript is required to be in ***ONE MS WORD FILE*** only (pdf. version is liable to be rejected without any consideration), which will start from the covering letter, inside the manuscript.
b) The sender is required to mentionthe following in the **SUBJECT COLUMN** of the mail:
New Manuscript for Review in the area of (Finance/Marketing/HRM/General Management/Economics/Psychology/Law/Computer/IT/ Engineering/Mathematics/other, please specify)
c) There is no need to give any text in the body of mail, except the cases where the author wishes to give any specific message w.r.t. to the manuscript.
d) The total size of the file containing the manuscript is required to be below **500 KB**.
e) Abstract alone will not be considered for review, and the author is required to submit the complete manuscript in the first instance.
f) The journal gives acknowledgement w.r.t. the receipt of every email and in case of non-receipt of acknowledgment from the journal, w.r.t. the submission of manuscript, within two days of submission, the corresponding author is required to demand for the same by sending separate mail to the journal.

2. **MANUSCRIPT TITLE:** The title of the paper should be in a 12 point Calibri Font. It should be bold typed, centered and fully capitalised.

3. **AUTHOR NAME (S) & AFFILIATIONS:** The author (s) **full name**, **designation**, **affiliation** (s), **address**, **mobile/landline numbers**, and **email/alternate email address** should be in italic & 11-point Calibri Font. It must be centered underneath the title.

4. **ABSTRACT:** Abstract should be in fully italicized text, not exceeding 250 words. The abstract must be informative and explain the background, aims, methods, results & conclusion in a single para. Abbreviations must be mentioned in full.

5. **KEYWORDS:** Abstract must be followed by a list of keywords, subject to the maximum of five. These should be arranged in alphabetic order separated by commas and full stops at the end.

6. **MANUSCRIPT:** Manuscript must be in ***BRITISH ENGLISH*** prepared on a standard A4 size ***PORTRAIT SETTING PAPER***. It must be prepared on a single space and single column with 1" margin set for top, bottom, left and right. It should be typed in 8 point Calibri Font with page numbers at the bottom and centre of every page. It should be free from grammatical, spelling and punctuation errors and must be thoroughly edited.

7. **HEADINGS:** All the headings should be in a 10 point Calibri Font. These must be bold-faced, aligned left and fully capitalised. Leave a blank line before each heading.

8. **SUB-HEADINGS:** All the sub-headings should be in a 8 point Calibri Font. These must be bold-faced, aligned left and fully capitalised.

9. **MAIN TEXT:** The main text should follow the following sequence:

INTRODUCTION

REVIEW OF LITERATURE

NEED/IMPORTANCE OF THE STUDY

STATEMENT OF THE PROBLEM

OBJECTIVES

HYPOTHESES

RESEARCH METHODOLOGY

RESULTS & DISCUSSION

FINDINGS

RECOMMENDATIONS/SUGGESTIONS

CONCLUSIONS

SCOPE FOR FURTHER RESEARCH

ACKNOWLEDGMENTS

REFERENCES

APPENDIX/ANNEXURE

It should be in a 8 point Calibri Font, single spaced and justified. The manuscript should preferably not exceed ***5000 WORDS***.

10. **FIGURES &TABLES:** These should be simple, crystal clear, centered, separately numbered &self explained, and **titles must be above the table/figure**. **Sources of data should be mentioned below the table/figure**. It should be ensured that the tables/figures are referred to from the main text.

11. **EQUATIONS:**These should be consecutively numbered in parentheses, horizontally centered with equation number placed at the right.

12. **REFERENCES:** The list of all references should be alphabetically arranged. The author (s) should mention only the actually utilised references in the preparation of manuscript and they are supposed to follow **Harvard Style of Referencing**. The author (s) are supposed to follow the references as per the following:

- All works cited in the text (including sources for tables and figures) should be listed alphabetically.
- Use (**ed.**) for one editor, and (**ed.s)** for multiple editors.
- When listing two or more works by one author, use --- (20xx), such as after Kohl (1997), use --- (2001), etc, in chronologically ascending order.
- Indicate (opening and closing) page numbers for articles in journals and for chapters in books.
- The title of books and journals should be in italics. Double quotation marks are used for titles of journal articles, book chapters, dissertations, reports, working papers, unpublished material, etc.
- For titles in a language other than English, provide an English translation in parentheses.
- The location of endnotes within the text should be indicated by superscript numbers.

PLEASE USE THE FOLLOWING FOR STYLE AND PUNCTUATION IN REFERENCES:

BOOKS

- Bowersox, Donald J., Closs, David J., (1996), "Logistical Management." Tata McGraw, Hill, New Delhi.
- Hunker, H.L. and A.J. Wright (1963), "Factors of Industrial Location in Ohio" Ohio State University, Nigeria.

CONTRIBUTIONS TO BOOKS

- Sharma T., Kwatra, G. (2008) Effectiveness of Social Advertising: A Study of Selected Campaigns, Corporate Social Responsibility, Edited by David Crowther & Nicholas Capaldi, Ashgate Research Companion to Corporate Social Responsibility, Chapter 15, pp 287-303.

JOURNAL AND OTHER ARTICLES

- Schemenner, R.W., Huber, J.C. and Cook, R.L. (1987), "Geographic Differences and the Location of New Manufacturing Facilities," Journal of Urban Economics, Vol. 21, No. 1, pp. 83-104.

CONFERENCE PAPERS

- Garg, Sambhav (2011): "Business Ethics" Paper presented at the Annual International Conference for the All India Management Association, New Delhi, India, 19–22 June.

UNPUBLISHED DISSERTATIONS AND THESES

- Kumar S. (2011): "Customer Value: A Comparative Study of Rural and Urban Customers," Thesis, Kurukshetra University, Kurukshetra.

ONLINE RESOURCES

- Always indicate the date that the source was accessed, as online resources are frequently updated or removed.

WEBSITES

- Garg, Bhavet (2011): Towards a New Natural Gas Policy, Political Weekly, Viewed on January 01, 2012 http://epw.in/user/viewabstract.jsp

SIGFREE WITH EXTENDED INSTRUCTION SEQUENCE GRAPH FOR DATA FLOW ANOMALY AND PROXY

SHAIK SHAFIA
ASST. PROFESSOR
COMPUTER SCIENCE & ENGINEERING DEPARTMENT
HYDERABAD INSTITUTE OF TECHNOLOGY & MANAGEMENT
R.R.DIST

ABSTRACT

I propose SigFree an online signature-free out of box application layer method for blocking 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 where as ligimate client requests never contain executables in most internet services, SigFree blocks attacks by detecting the presence of code. Unlike the previous code detection algorithms, SigFree uses a new data flow analysis technique called code abstraction that is generic, fast and hard for exploit code evade. 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. Since SigFree is a transparent deployment to the servers being protected, it is good for economical Internet wide deployment with very low deployment and maintenance cost .I implemented and tested SigFree our experimental study shows that the dependency degree based SigFree could block all types of code injection attack packets (above 750) tested in our experiments with very few false positives. Moreover, SigFree causes very small extra latency to normal client requests when some requests contain exploit code

KEYWORDS

Buffer Overflow, Code Abstraction, Data flow anomaly & proxy and SigFree.

INTRODUCTION

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 . 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. Although taking a broader viewpoint, buffer overflow attacks do not always carry binary code in the attacking requests or packets code injection buffer overflow such as stack smashing probably count for 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 t existing (legacy) server OS, application software and hardware; (R3) resiliency to obfuscation; (R4) economical Internet wide deployment. 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 we will review shortly. 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.

Accordingly, SigFree works as follow: SigFree is an application layer blocker that typically stays between a service and the corresponding firewall. When a service requesting message arrives at SigFree, SigFree first uses a new O(N) algorithm, where N is the byte length of the message, to disassemble and distill all possible instruction sequences from the messages payload, where every byte in the payload is considered as a possible starting point of the code embedded(if any) . However, in this phase, some data bytes may be mistakenly decoded as instructions. In phase 2 , SigFree uses a novel technique called code abstraction. Code abstraction first uses data flow anomaly to prune useless instructions in an instruction sequence, then compares the number of useful instructions(scheme2) or dependence degree(scheme 3) to a threshold to determine if this instruction sequence contains code. Unlike the existing code detection algorithms that are based on signatures, rules or control flow detection, SigFree is generic and hard for exploit code to evade.

The merits of SigFree are summarized as ,SigFree is Signature Free, thus it can block new and unknown buffer overflow attacks.Without relying on string machining, SigFree is immunized from most attack side obfuscation methods.SigFree uses generic code data separation criteria instead of limited rules. This feature separates SigFree from it, 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 aforementioned features.

REVIEW OF LITERATURE

COUNTING CODE INJECTION ATTACKS WITH INSTRUCTION SET RANDOMIZATION

I describe a new general approach for safeguarding systems against any type of code injection attack.I apply Kirchhoff's principle, by creating process specific randomized instruction sets of the system executing potentially vulnerable software. An attacker who does not know the key to the randomization algorithm will inject code that is invalid for that randomized processor, causing a runtime exception. To determine the difficulty of integrating support for the proposed mechanism in the OS,i modified the Linux kernel, the GNU binutils tools and the bochs-x86 emulator. Although the performance penalty is significant, this prototype demonstrates the feasibility of the approach and should be directly useable on a suitable modified processor.My approach is equally applicable against code injecting attacks in scripting and interpreted languages. The performance penalty in this case is minimal. Where my proposed approach is feasible, it can serve as a low overhead protection mechanism and can easily complement other mechanisms.

EFFICIENT TECHNIQUES FOR COMPREHENSIVE PROTECTION FROM MEMORY ERROR EXPLOITS

Despite the wide publicity received by buffer overflow attacks, the vast majority of today's security vulnerabilities continue to be caused by memory errors, with a significant shift away from stack smashing exploits to newer attacks such as heap overflows, integer overflows and format string attackers. While comprehensive solutions have been developed to handle memory errors, these solutions suffer from one or more of the following problems: high overheads (often exceeds 100%) , incompatibility with legacy C code and changes to the memory model to use garbage collection. Address space randomization (ASR) is a technique that avoids these drawbacks, but existing techniques for ASR do not offer a level of protection comparable to the above techniques.

PACKET VACCINE: BLACK BOX EXPLOIT DETECTION AND SIGNATURE GENERATION

In biblogy, a vaccine is a weakened strain of a virus or bacterium that is intentionally injected into the body for purpose of stimulating antibody production. Inspired by this idea, we propose a packet vaccine mechanism that randomizes address like strings in packets payloads to carry out fast exploit detection, vulnerability diagnosis and signature generation. An exploit with a randomized jump address behaves like a vaccine: it will likely cause an exception in a vulnerable program's process when attempting to hijack the **control-ow** and there by expose itself. Taking that exploit as a template, our signature generator creates a set of new vaccines to probe the program, in an attempt to uncover the necessary conditions for the exploit to happen.

STATIC ANALYSIS OF EXECUTABLES TO DETECT MALICIOUS PATTERNS

Malicious code detection is a crucial component of any defense mechanism. In this paper, i present a unique view point on malicious code detection.I regard malicious code detection as an obfuscation de-obfuscation game between malicious code writers and researchers working on malicious code detectors, such as

anti virus software.I tested the resilience of three commercial virus scanners against code obfuscation attacks. The results were surprising: the three commercial virus scanners could be subverted by very simple obfuscation transformation.I present an architecture for detecting malicious patterns in executables that is resilient to common obfuscation transformations.

OBFUSCATION OF EXECUTABLE CODE TO IMPROVE RESISTANCE TO STATIC DISASSEMBLY

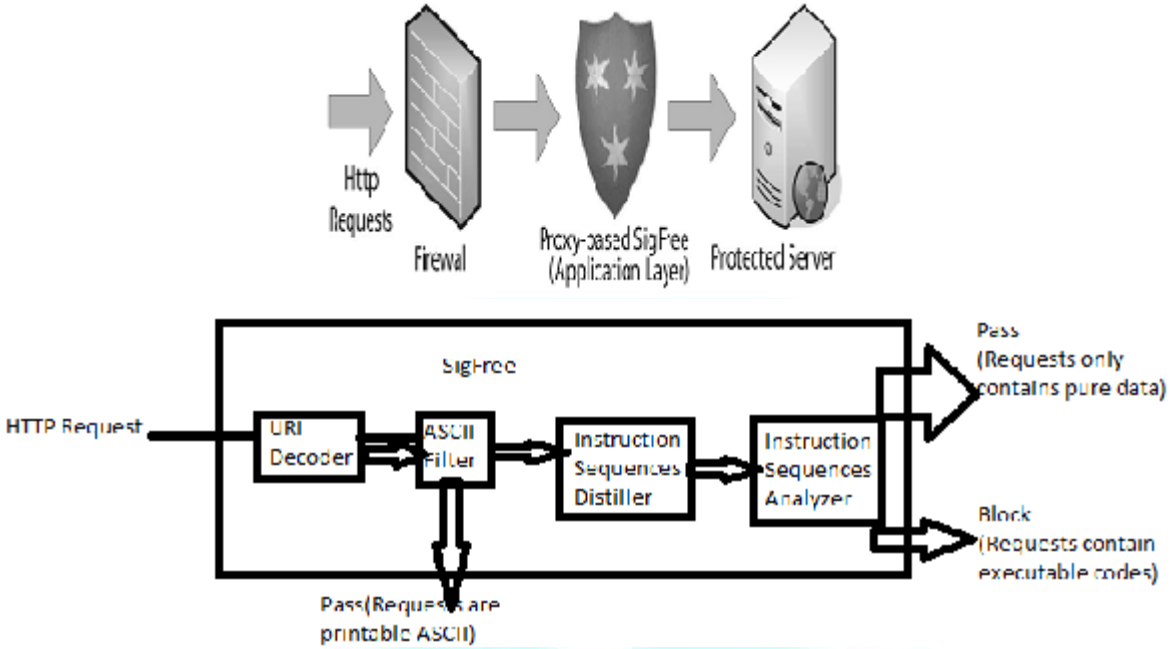
A great deal of software is distributed in the form of executable code. The ability to reverse engineer such executables can create opportunities for theft of intellectual; property via software piracy, as well as security breaches by allowing attackers to discover vulnerabilities in an application. The process of reverse engineering an executable program typically begins with disassembly, which translates machine code to assembly code. This is then followed by various recompilation steps that aim to recover higher level abstractions from assembly code. Experimental results indicate that significant portions of executables that have been obfuscated using techniques are disassembled in correctly, there by showing the efficacy of our methods.

IMPORTANCE OF THE STUDY

SigFree, an online signature free out of the box application layer method for blocking 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 where as legitimate client requests never contain executables in most Internet services, SigFree blocks attacks by detecting the presence of code. Unlike the previous code detection algorithms, SigFree uses new data flow analysis technique called code abstraction that is generic, fast and for exploit code to evade. SigFree is significant free, thus it can block new and unknown buffer overflow attacks. SigFree is also immunized from most attack side code obfuscation methods. Since SigFree is a transparent deployment to the servers being protected, it is good for economical Internet wide deployment with very low deployment and maintenance cost

METHODOLOGIES

ARCHITECTURE DIAGRAM



RESEARCH METHODOLOGY

PREVENTION/DETECTION OF BUFFER OVERFLOWS

Existing prevention/detection techniques of buffer overflows can be roughly broken down into six classes:

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 are not limited to model checking and bugs as deviant behavior .Class 1a technique are designed to handle source code only and they do not ensure completeness in bug finding. In contrast, SigFree handles machine code embedded in a request (message)

CLASS 1B: Compiler extensions. If source code is available, a developer can add buffer overflow detection automatically to a program by using a modified compiler". Tree such compilers are StakGuard, ProPolice and Return Address Defender(RAD), DIRA is another compiler that can detect control hijacking attacks , identify the malicious input and repair the compromised program. Class 1B techniques require the availability of source code. In contrast, SigFree does not need to know any source code.

CLASS 1C: OS modifications. Modifying some aspects of OS 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: Defence side obfuscation. Address Space Layout Randomization (ASLR) is main component of PaX, Address space randomization, in its general form, can defect exploitation of all memory errors. Instruction set Randomization, can detect all code injection attacks, where as SigFree cannot guarantee detecting all injected code. 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".

CLASS 1F: Capturing code running symptoms of buffer overflow attacks. Fundamentally, buffers overflow area code running symptom. If such unique symptoms can be captured, all buffer overflows can be detected.Class 1B, Class 1C and Class 1E techniques can capture some but not all of the running symptoms of buffer overflows.

WORM DETECTION AND SIGNATURE GENERATION

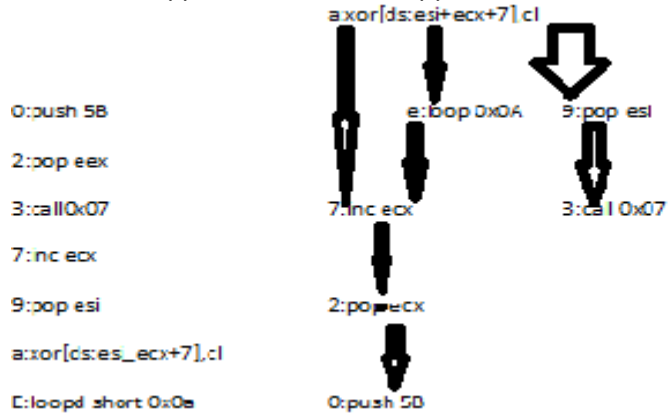
Because buffer overflow is a key target of worms when they propagate from one host to another, SigFree is related to worm detection. Based on the nature of worm infection symptoms, worm detection techniques can be broken down into 3 class [Class 2A] techniques use such macro symptoms as Internet background radiation to raise early warmings of Internet wide worm infection. [Class 2B] techniques use such local traffic symptoms as content invariance, content prevalence and address dispersion to generate worm signatures and/or block worms. [Class 2D] techniques use anomaly detection on packet payload to detect worms and generate. Wang and Stolfo first proposed Class 2D techniques called PAYL. PAYL is first trained with normal network flow traffic and then uses some byte level statistical measures to detect exploit code. Class 2B techniques are typically not very resilient to obfuscation. SigFree is immunized from most attack side obfuscation methods.

MACHINE CODE ANALYSIS FOR SECURITY PURPOSES

Although source code analysis has been extensively studied, in many real world scenarios, source code is not available and the ability to analyze binaries is desired. Machine code analysis has three main security purposes (P1) to malware detection, (P2) to analyze binaries and (P3) to identify and analyze the code contained in buffer overflow attack packets. 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. SigFree is an online attack blocker.

Four rules are discussed in this paper: Case 1 not only assumes the occurrence of the call/jmp instructions but also expects that the push instruction appears before the branch; Case 2 relies on the interrupt instruction; Case 3 relies on instruction set; Case 4 exploits hidden branch instructions. Besides, they used a special rule to detect polymorphic exploit code that contains loop

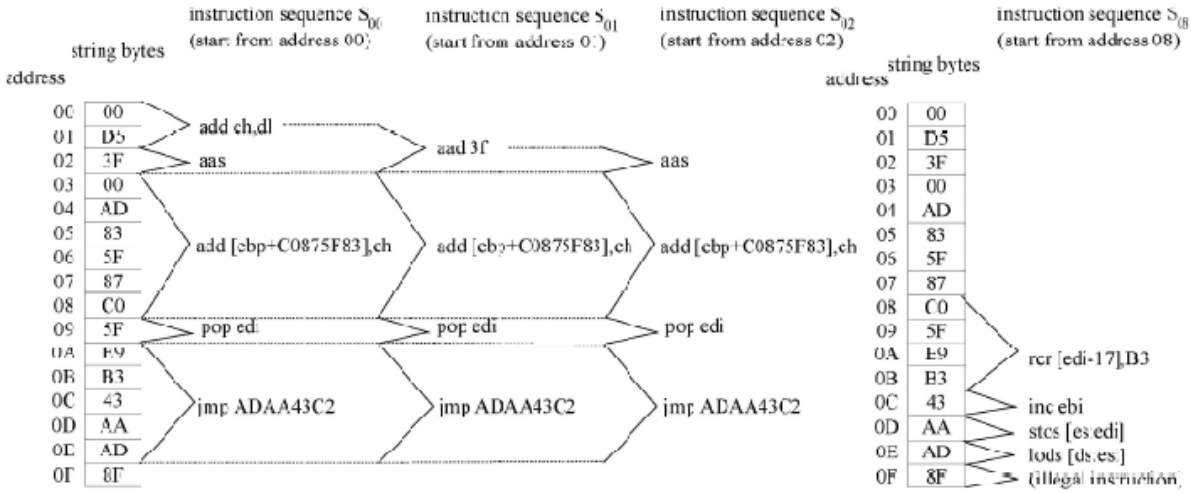
FIGURE 1: (a) A DECRYPTION ROUTINE (b) A DEF-USE GRAPH



INSTRUCTION SEQUENCE DISTILLER

This section first describes an effective algorithm to distill instruction sequences from requests, followed by several pruning techniques to reduce the processing overhead of instruction sequence analyzer

FIGURE 2: INSTRUCTION SEQUENCES DISTILLED



DISTILLING INSTRUCTION SEQUENCES

To distill an instruction sequence, first assign an address (staring from zero) to every byte of a request, where address is an identifier for each location in the request. Then I disassemble the request from a certain address until the end of 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. In this paper,i employ the recursive traversal algorithm, because it can obtain the control flow information during the disassembly process. To get all possible instruction sequences from an N-byte request, we simply execute the disassembly algorithm N times and each time i start from a different address in the request. This gives us a set of instruction sequences. The running time of this algorithm is O(N)**.** One drawback of the above algorithm is that the same instructions are decoded many time, i design a memorization algorithm by using a data structure, which is an EIFG defined earlier, to represent the instruction sequences. To distill all possible instruction sequences from request is simply to create the EIFG for the request . An EIFG is used to represent all possible instruction in a request. To traverse an instruction sequence, we simply traverse the EIFG from the entry instruction of the instruction sequence and fetch the corresponding instructions from the instruction array . FIGURE 3 shows the data structure for the request shown in FIGURE 2. The details of the algorithm for creating the data structure are described in Algorithm 1. Clearly, the running time of this algorithm is O(N), which is optimal as each address is traversed any once.

ALGORITHM 1: Distill all Instruction sequences from a request initialize EISG G and instruction arrray A to empty
for each address *i* of the request **do**
add instruction node *i* to G
i ← the start address of the request
while *i* <= the end address of the request **do**
inst ← decode an instruction at *i*
if *inst* is illegal **then**
A[i] ← illegal instruction *inst*
Set type of node *i* "illegal node " in G
else
A[i] ← instruction *inst*
if *inst* is a control transfer instruction **then**

for *inst* is a control transfer instruction **do**
if target *t* is an external address **then**
add external address node *t* to G
add edge *e*(node *i*,node *t*) to G
else
add edge e(node *i*,node *i+inst.length*) to G
i←*i*+1

EXCLUDING INSTRUCTION SEQUENCES

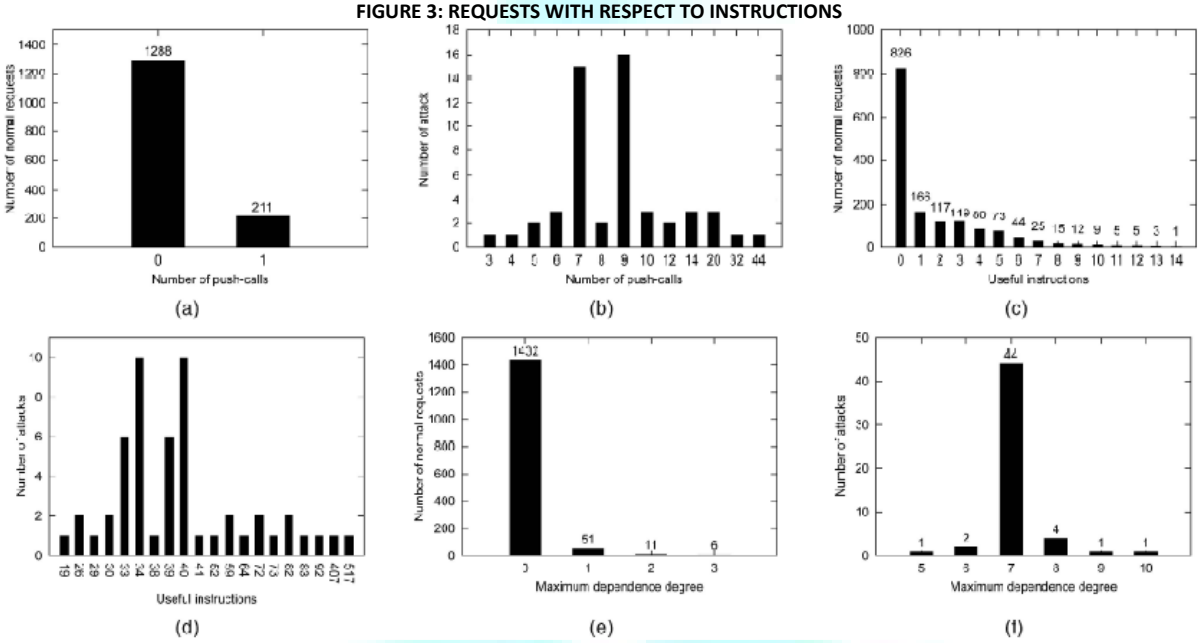
The previous step may output many instruction sequences at different entry points. Next, i exclude some of them based on several heuristics. Here, excluding an instruction sequences means that the entry of this sequence is not considered as the real entry for the embedded code. The fundamental rule in excluding instruction sequences is not to affect the decision whether a request contains code or not, this rule can be translated into the following technical requirements: if a request contains a fragment of a program, the fragment must be one of the remaining instruction sequence or a subsequence of a remaining instruction sequences or it differs from a remaining sequence only by few instructions.

STEP1: If instruction sequence s_a is a subsequence of instruction sequence s_b , we execute s_a . The rationale for excluding s_a is that if s_a satisfies some characteristics of programs, s_b also satisfies these characteristics with a high probability.

STEP2: If instruction sequence s_a merges to instruction sequence s_b after a few instructions and s_a is no longer than s_b, we exclude s_a. It is reasonable to expect that s_b will preserve s_a's characteristics. Many distilled instruction sequences are observed to merge other instruction sequences after a few instructions.

STEP 3: For instruction sequences, when they are executed, which ever execution path is taken, an illegal instruction is inevitably reacted. We say an instruction is inevitably reached if two conditions hold. One is that there are no cycles in EIFG of the instruction sequence; other is that there are no external address nodes in the EIFG of the instruction sequence.

FIGURE 3: REQUESTS WITH RESPECT TO INSTRUCTIONS



INSTRUCTION SEQUENCE ANALYZER

A distilled instruction sequence may be a sequence of random instructions or a fragment of a program in machine language.

SCHEME 1: A program machine language is dedicated to a specific OS; hence, a program has a certain characteristics implying the OS on which it is running, for example calls to OS or kernel library. A random instruction sequence does not carry this kind of characteristics. By identifying the call pattern in an instruction sequence, we can effectively differentiate a real program from a random instruction sequence. To address this issue, we use a pattern composed of several instruction used to transfer parameters.One possible obfuscation is that attackers may use other instruction is substituted to replace the "call" and "push" instructions. FIGURE 1 shows an example of obfuscation, where "call EAX" instruction is substituted by "push J4" and "jmp EAX". Although we cannot fully solve this problem, by recording this kind of instruction replacement patterns

SCHEME 2: Next, we use the detection of data flow anomaly in different way called code abstraction. We observe that when there are data flow anomalies in an execution path of an instruction sequence, some instructions are useless, where as in a real program at least one execution path has certain number of instructions. Data flow anomaly, the term data flow anomaly was originally used to analyze programs written in higher level languages in the software reliability and testing field. There are three data flow anomalies: define-define define-undefined and undefined reference. The define-define anomaly means that a variable was defined and is defined again, but it has never been referenced between these two actions. The undefined refence anomaly indicates that a variable was undefined receives a reference action. The defined-undefined anomaly means that a variable was defined and before it is used as undefined. Figure 1 shows an example

DETECTION OF DATA FLOW ANOMALY

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. As such, we propose a new method called code abstraction, which does not require real execution of code. As a result of the code abstraction of an instruction, a variable could be in one of the six possible states. The six possible states are state U: undefined, state D: defined but not reference; and state DU: abnormal state defines- undefined. FIGURE 6 depicts the state diagram of these states. Each edge in this state diagram is associated with d, r, or u which represents "define", "reference", and "undefined" respectively.I assume that a variable is in "undefined" state at the beginning of an execution path. Now,i start to traverse this execution path. If the entry instruction of the execution path defines this variable, it will enter the state "defined". Then, it will enter another state according to the next instruction, as we shown in FIGURE 6. Once the variable enters an abnormal state, a data flow anomaly is detected. I continue this traversal to the end of the execution path. ALGORITHM 2 shows 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 push down stack is available for storage. During the search process, the visited nodes is abstractly executed to update the states of variables , find data flow anomaly and prune useless instructions in an execution path.

ALGORITHM 2 check if the number of useful instruction in an execution path exceeds a threshold

INPUT: *entry* instruction of an instruction sequence, EISG G

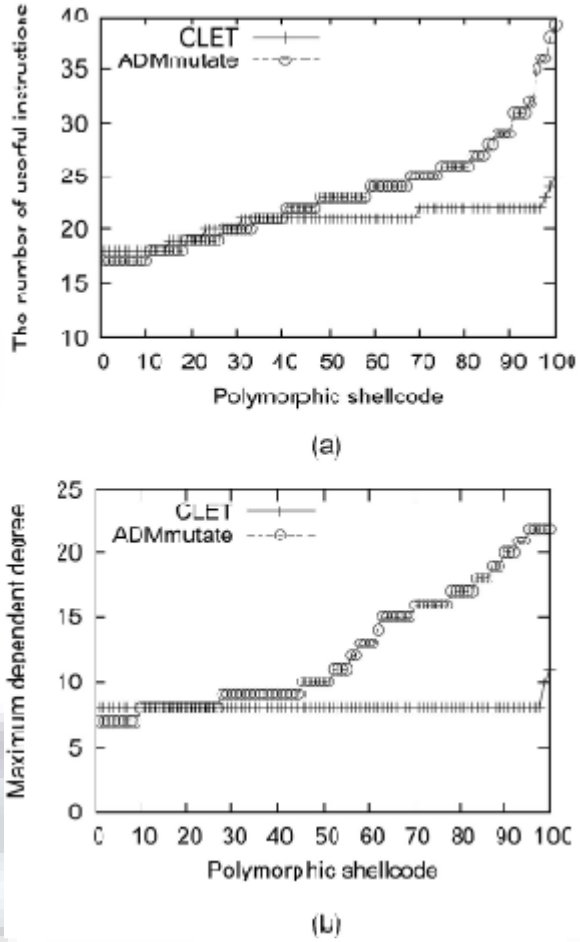
total← *0; useless*←*0;stack*←empty

```
  initialize the states of all variables to "undefined"
  push the empty instruction, states,total and useless to stack
 while stack is not empty do
   pop the top item of stack to i, states,total and useless
   if total←useless greater than a threshold   then
    return  true
  if i is visited then
    continue (passes control to the next iteration of the WHILE loop)
    mark i visited
    total←total+1
    Abstractly execute instruction i
    if  there is a define-define or define-undefine anomaly then
      useless←useless+1
   if  there is an undefine-reference anomaly then
     useless←useless+1
 for each instruction j directly following i in the G  do
    push j,states,total and useless to stack
return  false
```

HANDLING SPECIAL CASES

Next, i discuss several special cases in the implementation of scheme 2. General purpose instruction. The instructions in the IA-32 instruction set can be roughly divided into four groups: General purpose instructions, floating point unit instructions, extension instructions and system instructions. General purpose instructions are also the most often used instructions in malicious code. I believe that malicious codes must contain number of general purpose instructions to achieve the attacking goals. Other types of instructions may be leveraged by an attacker to obfuscate his real purpose code, e.g. used as garbage in garbage insertion. As such, we consider other groups of instructions as useless instructions.

FIGURE 4: POLYMORPHIC SHELL CODE DEPENDS ON INSTRUCTIONS DEGREE



INITIAL STATE OF REGISTERS

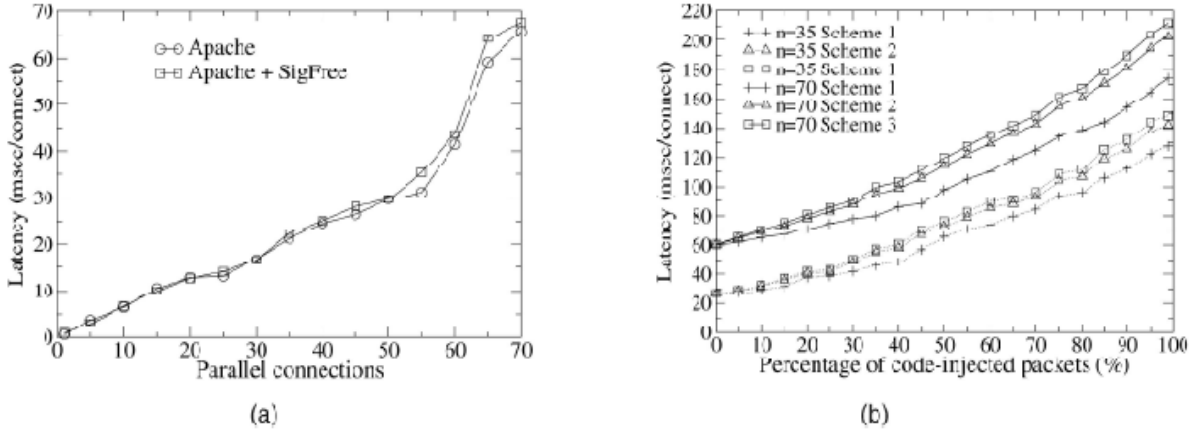
For registers, we get their initial states to "undefined" at the beginning of an execution path. The register "ESP", however, is an exception since it is used to hold the stack pointer. Thus, we set register ESP "defined" at the beginning of an execution path indirect address. An indirect address is an address that serves as a reference point instead of an address to the direct memory location.Thus, we always treat a memory location to which an indirect address points as state "defined" and hence no data flow anomaly will be generated. Indeed, this treatment successfully prevents an attacker from obfuscating his code using indirect addresses. Useless control transfer instructions (CTIs). Condition instructions Jcc(jump on condition code cc) and LOOPcc use one or more of the status flags as condition codes and test them for branch or end loop conditions. During a program execution at runtime, an instruction may affect a status flag on three different ways: set, unset, undefined. We consider both set and unset are defined in code abstraction.

SCHEME 3: I propose SCHEME 3 for detecting the aforementioned specially crafted code. Scheme 3 also exploits code abstraction to prune useless instructions in an instruction sequence. Unlike SCHEME 2, which compares the number of useful instructions with a threshold, SCHEME 3 first calculates the dependent degree of every instruction in the instruction sequence. If the dependence degree of any useful instructions in an instruction sequence exceeds a threshold, i conclude that the instruction sequence is a segment of a program. Dependency is a binary relation over instructions in an instruction sequence.

STAND ALONE SIGFREE

Implemented a stand alone SigFree prototype using the C programming language in the Win32 environment. The stand alone prototype was compiled with Borland C++ version 5.5.1 at optimization level 02. The experiments were performed in Windows 2003 server with Intel Pentium 4, 3.2 GHz CPU and 1 GByte memory. I measured the processing time of the stand alone prototype over all 0-10 Kbytes images collected from the above real traces. I set the upper limit to 10 Kbytes because the size of a normal web request is rarely over that if it accepts binary inputs. The types of the images include JPEG, GIF, PNG and X-ICON. FIGURE 5 shows that the average processing time of the three schemes increases linearly when the sizes of the image files increase. It also shows that SCHEME 1 is the fastest among the three schemes and SCHEME 3 is a little bit slower than SCHEME 2. In all three Schemes, the processing time over a binary file of 10 Kbytes is no more than 85ms.

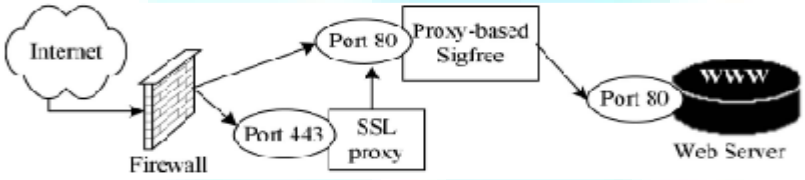
FIGURE 5:PERFORMANCE IMPACT OF SIGFREE ON APACHE HTTP SERVER



PROXY BASED SIGFREE

To evaluate the performance impact of SigFree to web servers,i also implemented a proxy based SigFree prototype. FIGURE 6 depicts the implementation architecture. It is comprised of the following modules *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.*ASCII filter,* malicious executable codes are normally binary strings. In order to guarantee the throughput and response time of the protected web systems, if a request is printable ASCII ranging from 20 to 7E hex, SigFree allows the request to pass. Note that ASCII filter does not prevent the service from receiving non-ASCII strings. All non-ASCII strings will analyzed by ISD and ISA. The proxy based prototype was also compiled with Borland C++ version 5.5.1 at optimization level O2. The proxy based prototype implementation was hosted in the windows 2003 server with Intel Pentium 4, 3.2 GHz CPU and 1-GByte memory.

FIGURE 6: SIGFREE WITH AN SSL PROXY



RESULTS

DATABASE TABLES

FIGURE 7: ADMIN TABLE

	TABLE_QUALIFIER	TABLE_OWNER	TABLE_NAME	TABLE_TYPE	REMARKS
18	sigfree	dbo	systypes	SYSTEM TABLE	NULL
19	sigfree	dbo	sysusers	SYSTEM TABLE	NULL
20	sigfree	dbo	dtproperties	TABLE	NULL
21	sigfree	dbo	Username	TABLE	NULL
22	sigfree	dbo	sysconstraints	VIEW	NULL
23	sigfree	dbo	syssegments	VIEW	NULL

FIGURE 8:USER DATABASE TABLE

	FirstName	password	confirm	username	city	state	emailid
1	vivek	123455	123456	vivek	chennai	tamil nadu	vivek@gmail.com
2	null	null	null	null	null	null	null
3	obarai	123455	123456	obarai	chia	ffxgk	oba@gmail.com
4	veera	badra1234	badra1234	veera	Hyderabad	AP	veerabadra@gmail.com
5	anitha	anitha	anitha	anitha	Vijayawada	AP	veera@gmail.com
6	teja	123455	123456	teja	chennai	tamil nadu	teja143@gmail.com
7	Mahesh	mahesh	mahesh	mahesh	tirupati	AP	mahesh@gmail.com
8	seshu	seshu1	seshu1	seshu	bvrm	ap	seshu@gmail.com
9	vamshi	vamshi	vamshi	vamshi	hyderabad	AP	vamshi@gmail.com

FIGURE 9: CLIENT SIDE

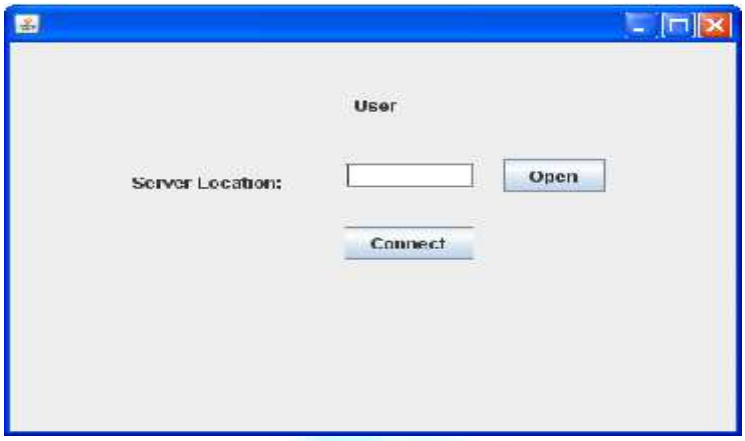


FIGURE 10 : CLIENT PROGRAM

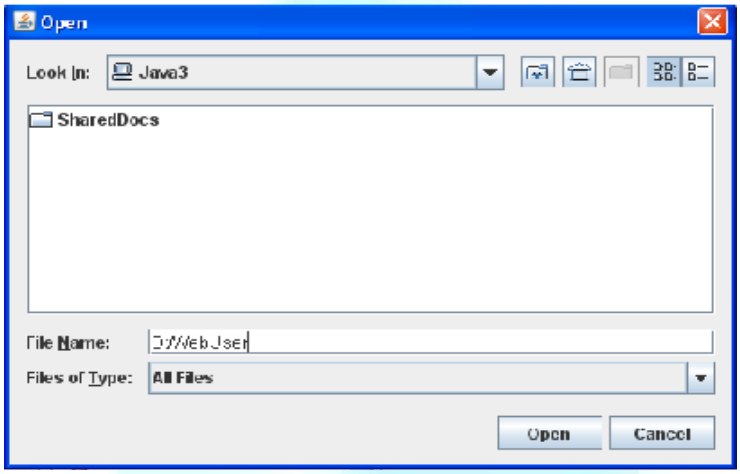


FIGURE 11: SIGFREE APPLICATION LOGIN PAGE

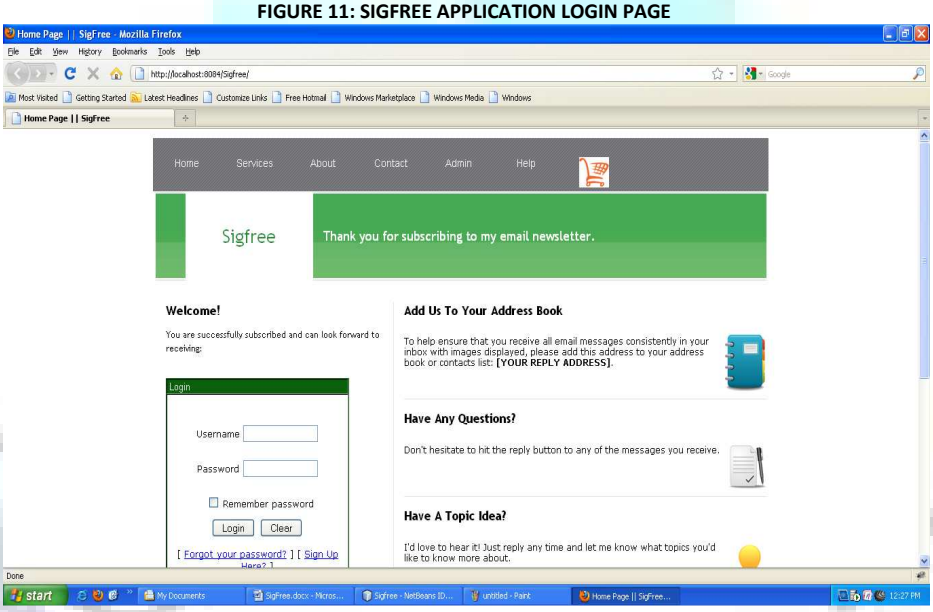


FIGURE 12: HACKER SENDS THE ADDS

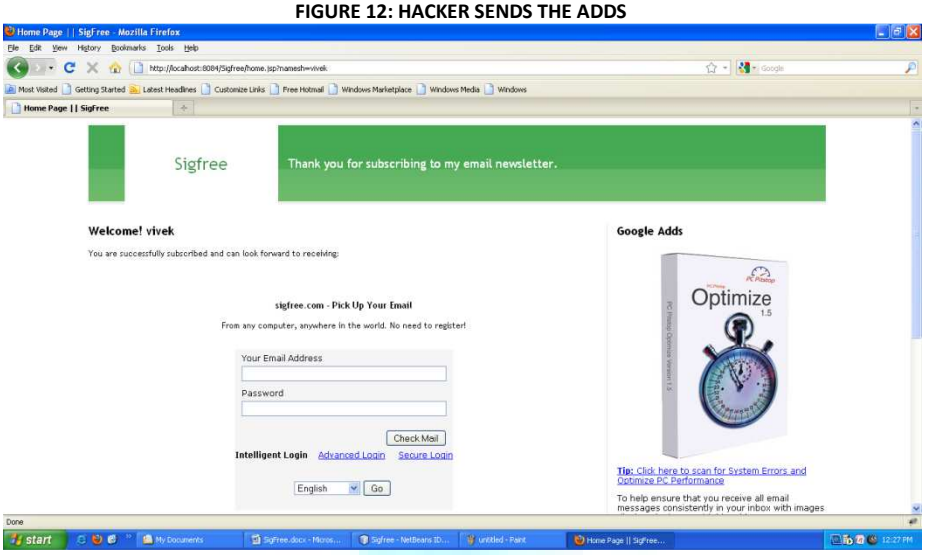


FIGURE 13: USER CLICKS ON ADD

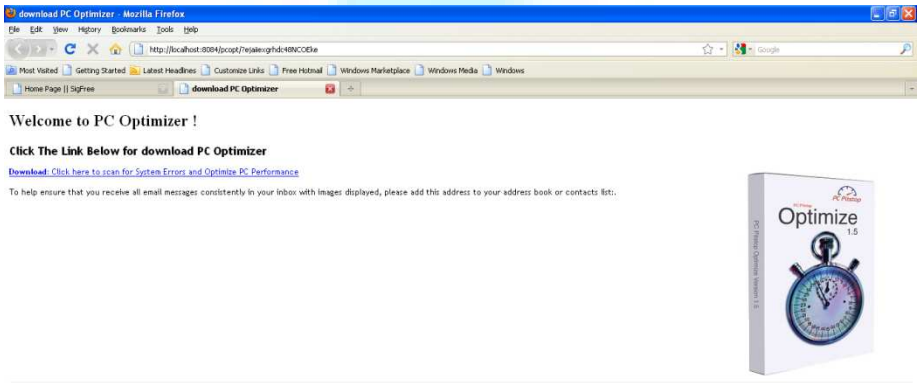
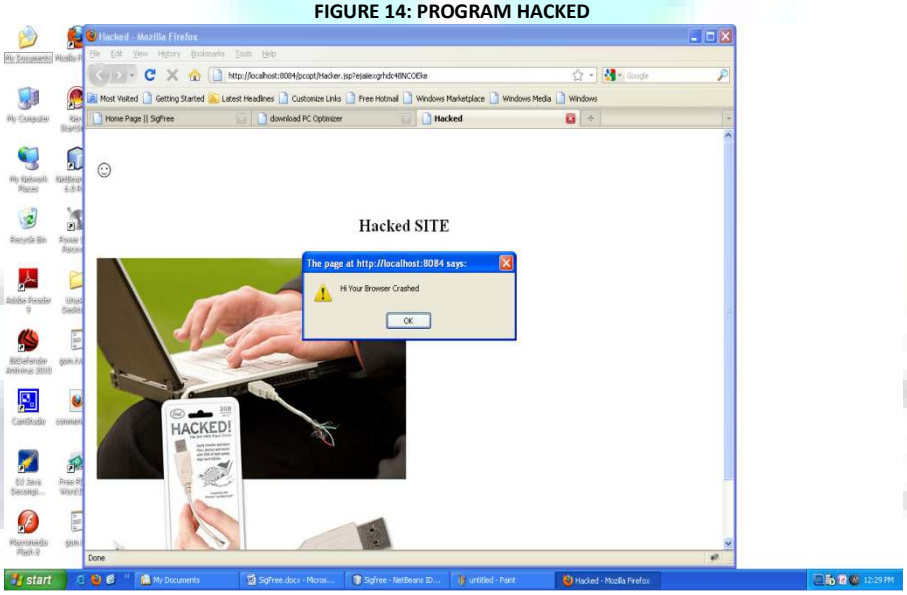


FIGURE 14: PROGRAM HACKED



ADMIN

FIGURE 15: ADMIN LOGIN PAGE

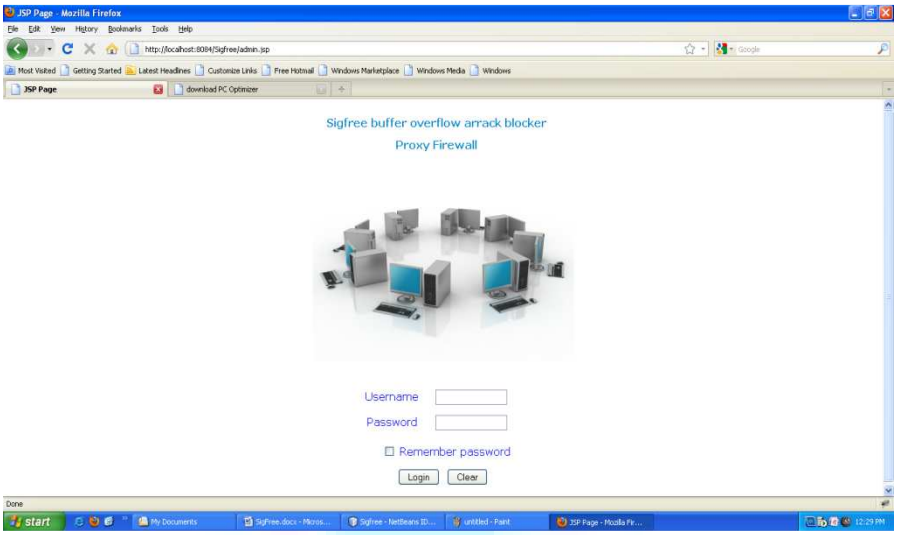


FIGURE 16: CLUSTER VIEW

FIGURE 17: AFTER SELECTING CLUSTER1

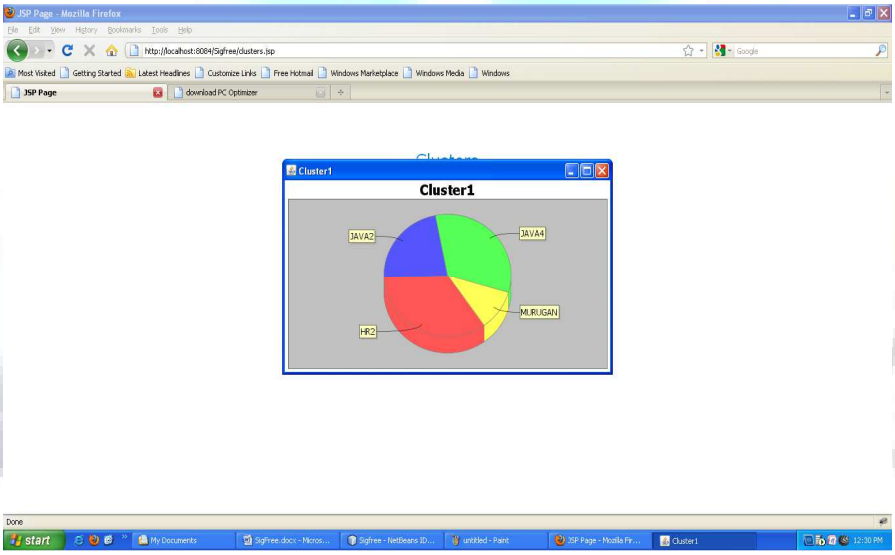


FIGURE 18: USER'S VIEW

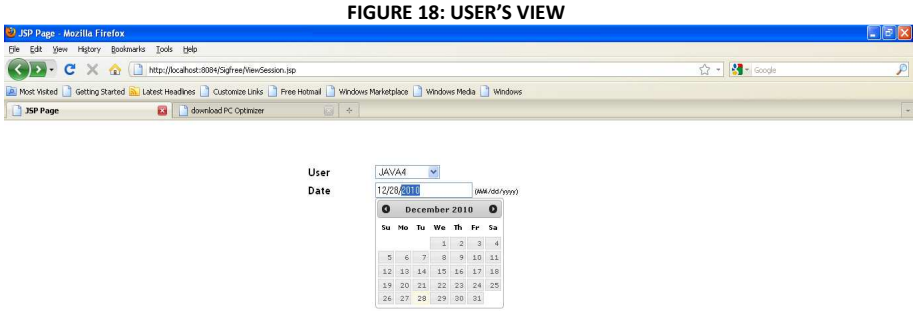
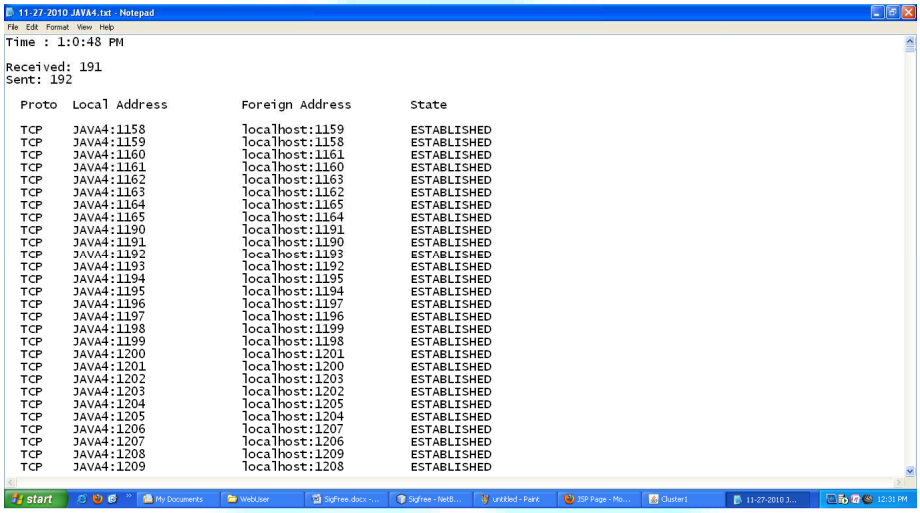


FIGURE 19: CONNECTIONS ESTABLISHED



CONCLUSION

A great deal of software is distributed in the form of executable code. Such code is potentially vulnerable to reverse engineering, in the form of disassembly followed by recompilation. This can allow an attacker to discover vulnerabilities in the software modify it in unauthorized ways or steal intellectual property via software privacy. This paper describes and evaluates techniques to make executable programs harder to disassemble. Our techniques are seen to be quite effective applied to the widely used SPECint-95 benchmark suite, they result in disassemble over 65%of the instructions and 85% of the functions in the obfuscated binaries. We SigFree does not require any Signatures, it can block new unknown attacks. It is immunized from most attack side code obfuscation methods and good for economical Internet wide deployment with little maintenance cost

REFERENCES

1. B.A. Kuperman, C.E. Brodley, H. Ozdoganoglu, T.N. Vijaykumar, and A. Jalote, "Detecting and Prevention of Stack Buffer Overflow Attacks," Comm. ACM, vol. 48, no. 11, 2005.
2. D. Wagner, J.S. Foster, E.A. Brewer, and A. Aiken, "A First Step towards Automated Detection of Buffer Overrun Vulnerabilities," Proc. Seventh Ann. Network and Distributed System Security Symp. (NDSS '00), Feb. 2000.
3. E. Barrantes, D. Ackley, T. Palmer, D. Stefanovic, and D. Zovi, "Randomized Instruction Set Emulation to Disrupt Binary Code Injection Attacks," Proc. 10th ACM Conf. Computer and Comm. Security (CCS '03), Oct. 2003.
4. G. Kc, A. Keromytis, and V. Prevelakis, "Countering Code- Injection Attacks with Instruction-Set Randomization," Proc. 10th ACM Conf. Computer and Comm. Security (CCS '03), Oct. 2003.
5. H.-A. Kim and B. Karp, "Autograph: Toward Automated, Distributed Worm Signature Detection," Proc. 13th USENIX Security Symp. (Security), 2004.
6. J. Newsome and D. Song, "Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software," Proc. 12th Ann. Network and Distributed System Security Symp. (NDSS), 2005.
7. J. Newsome, B. Karp, and D. Song, "Polygraph: Automatic Signature Generation for Polymorphic Worms," Proc. IEEE Symp. Security and Privacy (S&P), 2005.
8. J. Pincus and B. Baker, "Beyond Stack Smashing: Recent Advances in Exploiting Buffer Overruns," IEEE Security and Privacy, vol. 2, no. 4, 2004.
9. Z. Liang and R. Sekar, "Fast and Automated Generation of Attack Signatures: A Basis for Building Self-Protecting Servers," Proc. 12th ACM Conf. Computer and Comm. Security (CCS), 2005.

REQUEST FOR FEEDBACK

Dear Readers

At the very outset, International Journal of Research in Computer Application & Management (IJRCM) acknowledges & appreciates your efforts in showing interest in our present issue under your kind perusal.

I would like to request you tosupply your critical comments and suggestions about the material published in this issue as well as on the journal as a whole, on our E-mail**infoijrcm@gmail.com** for further improvements in the interest of research.

If youhave any queries please feel free to contact us on our E-mail infoijrcm@gmail.com.

I am sure that your feedback and deliberations would make future issues better – a result of our joint effort.

Looking forward an appropriate consideration.

With sincere regards

Thanking you profoundly

Academically yours

Sd/-
Co-ordinator

DISCLAIMER

The information and opinions presented in the Journal reflect the views of the authors and not of the Journal or its Editorial Board or the Publishers/Editors. Publication does not constitute endorsement by the journal. Neither the Journal nor its publishers/Editors/Editorial Board nor anyone else involved in creating, producing or delivering the journal or the materials contained therein, assumes any liability or responsibility for the accuracy, completeness, or usefulness of any information provided in the journal, nor shall they be liable for any direct, indirect, incidental, special, consequential or punitive damages arising out of the use of information/material contained in the journal. The journal, nor its publishers/Editors/ Editorial Board, nor any other party involved in the preparation of material contained in the journal represents or warrants that the information contained herein is in every respect accurate or complete, and they are not responsible for any errors or omissions or for the results obtained from the use of such material. Readers are encouraged to confirm the information contained herein with other sources. The responsibility of the contents and the opinions expressed in this journal is exclusively of the author (s) concerned.

ABOUT THE JOURNAL

In this age of Commerce, Economics, Computer, I.T. & Management and cut throat competition, a group of intellectuals felt the need to have some platform, where young and budding managers and academicians could express their views and discuss the problems among their peers. This journal was conceived with this noble intention in view. This journal has been introduced to give an opportunity for expressing refined and innovative ideas in this field. It is our humble endeavour to provide a springboard to the upcoming specialists and give a chance to know about the latest in the sphere of research and knowledge. We have taken a small step and we hope that with the active co-operation of like-minded scholars, we shall be able to serve the society with our humble efforts.

Our Other Journals