Proceedings

Hack.in 2009

3rd Hackers' Workshop on Computer and Internet Security

March 17-19, 2009

Organized by

Prabhu Goel Research Centre for Computer & Internet Security

Department of Computer Science and Engineering

Indian Institute of Technology Kanpur

Table of Contents

Hack.in 2009

The Hackers' Workshop

Keynote

Technical Session I: Attack Mechanisms

Technical Session II: Security Solutions

Technical Session III: Tools and State-of-the-Art

Hack.in 2009 Workshop Organization

General Chair

Manindra Agrawal, IIT Kanpur, India

Program Chair

Rajat Moona, IIT Kanpur, India

Program Committee

Bernard Menezes, IIT Bombay, India

Bezawada Bruhadeshwar, IIIT Hyderabad, India

Bimal Roy, ISI Calcutta, India

Deepak Gupta, Airtight Networks, India

Dheeraj Sanghi, LNMIIT Jaipur, India

Dhiren Patel, SVNIT Surat, India

Indranil Sengupta, IIT Kharagpur, India

Manoj S Gaur, MNIT Jaipur, India

Piyush Kurur, IIT Kanpur, India

Organizing Committee

Abhay Khoje, IIT Kanpur, India

Arnab Bhattacharya, IIT Kanpur, India

Navpreet Singh, IIT Kanpur, India

Nitish Dutt Sharma, IIT Kanpur, India

Om Prakash Gupta, IIT Kanpur, India

Rajni Moona, IIT Kanpur, India

Sajal Bhatia, IIT Kanpur, India

Salih K A, IIT Kanpur, India

Satyam Sharma, IIT Kanpur, India

Message from the General Chair

Hack.in 2009 General Chair

Manindra Agrawal, *Indian Institute of Technology Kanpur, India*

On behalf of the Hack.in 2009 organizing committee, I welcome the participants to the conference. The conference program committee has put together a very interesting programs consisting of keynote talks by experts in different aspects of the security area and contributed papers.

This is the third Hacker's Workshop being organized at IIT Kanpur under the auspices of Prabhu Goel Research Centre for Computer and Internet Security. The aim of the workshop is to get the practitioners in all aspects of computer security, particularly those working in systems security, together for exchange of ideas and discussions. Several recent events have served to strengthen the case for substantially expanding the need for research in cyber security in India. We hope that the through this workshop and other activities, Prabhu Goel Centre will play an important role in this.

Finally, I would like to thank all the authors and the members of the program committee whose efforts have made this conference a success.

Message from the Program Chair

Hack.in 2009 Program Chair

Rajat Moona, *Indian Institute of Technology Kanpur, India*

Welcome to Hack.in 2009. The security related to data has assumed a greater significance in computing and information management today. Several on-line frauds and subsequently the terror attacks have exploited the security vulnerabilities in the information systems.

Hack.in 2009 is the third workshop organized by Prabhu Goel Research Centre on Computer and Internet Security. This year, the workshop had invited the research contributions from various researchers and we received about 40 contributions. After the review process about 10 of these papers were accepted for publication in the proceedings of the workshop. In the coming years, we wish to make it a much bigger workshop with lot many research papers and participation from everyone.

Apart from technical papers, the workshop has been enriched by several exciting keynote talks and panel discussions from eminent international experts in the field. The workshop also brings people on a common forum to bring healthy interactions among the practitioners, researchers and experts. I am sure this workshop will nurture greater ideas and culminate on a promising future in the area.

Hack.in 2009 is a result of great teamwork from all the organizers and volunteers and Innumerable others who have taken time off from their busy schedules to make it a successful technical and interactive event. Wish you all a memorable time at Hack.in 2009.

Workshop Program

17 March, 2009	09:00 am	Inauguration and Registration
	09:30 am	Introduction to Workshop
	09:45 am	Welcome speech by Prof. Manindra Agrawal
	10:00 am	Keynote talk by Prof. D. J. Bernstein
	11:00 am	Vote of thanks to the keynote
	11:15 am	Tea
	12:00 pm	Technical Session I: Attack Mechanisms
	01:30 pm	Lunch
	02:45 pm	Technical Session II: Security Solutions
	04:15 pm	Tea
18 March, 2009	09:30 am	Keynote talk by Prof. Antonio Mana Gomez
	10:30 am	Tea
	11:00 am	Technical Session III: Tools and State-of-the-Art
	01:00 pm	Lunch
	02:00 pm	Workshop on wireless security by Airtight networks
	05.00 pm	Tea
19 March, 2009	09:30 am	Keynote talk by Prof. Bimal Roy
	10:30 am	Tea
	11.00 am	Panel Discussion
	12.45 pm	Vote of thanks

Keynote

Daniel Julius Bernstein

Keynote Speaker

D J Bernstein is currently a research professor at the University of Illinois, Chicago. He is a mathematician, a Cryptologist, and a Programmer and received his PhD in Mathematics from the University of California, Berkeley. Professor Bernstein has written a number of security-aware programs, including *qmail, djbdns, ucspi-tcp, daemontools and publicfile*. His current research interests include computer and network security, high speed cryptography, kernel minimization and high speed engineering for secure software. He has been a author of many academic papers on mathematics and computation, and has also written a famous survey titled, *"Multidigit multiplication for mathematicians"*. In August 2008 he announced *DNSCurve*, a proposal to secure the Domain Name System. DNSCurve uses techniques from elliptic curve cryptography to give a vast decrease in computational time over the RSA public-key algorithm used by DNSSEC, and uses the existing DNS hierarchy to propagate trust by embedding public keys into specially formatted (but backward-compatible) DNS records.

DNSCurve

The DNSCurve project adds link-level public-key protection to DNS packets. It uses high-speed, high-security elliptic-curve cryptography to drastically improve every dimension of DNS security. It provides confidentiality by encrypting all the DNS requests and responses, integrity by cryptographically authenticating all DNS responses, and availability by quickly recognizing and discarding forged packets. It uses public-key cryptography, specifically 255-bit elliptic-curve cryptography, to detect forgeries. 255-bit elliptic-curve cryptography is billions of times harder to break than 1024-bit RSA by current cryptanalytic algorithms. Despite its extremely high level of security, DNSCurve is very easy for software authors to implement, and very easy for administrators to deploy. It can be even used to protect queries to top-level DNS servers, such as the .com servers.

Antonio Maña Gómez

Keynote Speaker

Antonio Maña is currently Professor of Software Engineering in the Computer Science Department of the University of Malaga (SPAIN). His current research activities include security and software engineering, information and network security, ubiquitous computing and ambient intelligence, application of smart cards to digital content commerce, software protection, DRM and mobile applications. He is currently the research director of the EC-supported SERENITY Integrated Project, which deals with the provision of security and dependability in ambient intelligence systems based on the concept of Security and Dependability Pattern.

Towards Information Security as an Engineering Discipline: The SERENITY experience

Current practices for developing secure systems are still closer to art than to an engineering discipline. Security is still treated as an add-on and is therefore not integrated into software development practices and tools. Experienced security artisans are still the key to achieving acceptable levels of security.

Several approaches and research strands have tried to address this situation in order to introduce rigour and engineering approaches in the treatment of security aspects in information systems, mainly focusing on the development phases.

Traditionally, the term security engineering has been used to denote partial approaches that cover only small parts of the processes that are required in order to create a secure system, like modelling, verification, programming, etc. Even in the cases that the approach is closer to a methodology, and has achieved a certain level of maturity, the key concepts and workflows are highly influenced by the way had been treated by the security artisans. Therefore, one finds in the literature that the main books about security engineering describe threat-based engineering approaches.

The main drawbacks of these approaches is that they fail to provide a reasonable support for systematic engineering since the identification, characterization and specification of the threats as well as the selection of appropriate mechanisms and countermeasures depends on the experience of the engineers. Consequently, these approaches represent only minor improvements over the security art. However, they have been used for some time with uneven results. The talk will discuss this aspect and

will advocate a change of paradigm based on the precise and formal specification of security properties and the use of these formally verified properties as the basis for the engineering of secure systems.

Today, the current trend towards distributed and open systems, has revealed the important limitations of threat-based security engineering. In particular, threat-based security engineering creates systems that are very context-dependent, and therefore, fail to address the needs of the future open and distributed systems paradigms. The talk will show that threat-based security engineering (i) is the origin of penetrate-and-patch situation; (ii) does not result in specifications that can survive evolution of systems and context; and (iii) does not capture the requirements, but ways of maintaining them, which results in poor maintainability of the systems produced.

The main problems that the new computing paradigms introduce are the high levels of heterogeneity, dynamism and autonomy, as well as the large scale. The result is that engineers have to deal with runtime situations that are unpredictable at design time. The final part of the talk will introduce the SERENITY model of secure and dependable systems and will show how it supports the creation of secure and dependable systems for these new computing paradigms.

Finally some conclusions will be drawn and a proposal for establishing a renewed security engineering discipline will be advocated.

Bimal Roy

Keynote Speaker

Bimal Roy is Professor-in-Charge of Applied Statistics Unit, Cryptography Research Group, Indian Statistical Institute (ISI) Kolkata. He received his B.Stat. (Hons) and M. Stat. from ISI Kolkata and PhD from University of Waterloo, Canada. His current research interests are combinatorics, design of experiments, optimization and cryptology. He has been a recipient of Reliance Platinum Jubilee Award for innovation in physical sciences and IBM faculty award for research, teaching and initiative in cryptology and security. He has been the general secretary of Cryptology Research Society of India and program chair for FSE 2004, Asiacrypt 2005 and Indocypt 2000 and 2009.

On Distinguishing Attacks

A numeric sequence is said to be statistically random when it contains no recognizable patterns or regularities. Sequences such as the results of an ideal die roll, or the digits of p exhibit statistical randomness. There exists several tests for non-randomness such as Frequency test, Serial test, Poker test, Gap test, Run Test, Autocorrelation test, Maurer's universal test etc. Distinguishing attack aims at differentiating the output of a block cipher from a random permutation, or a stream cipher's keystream from a random bitstream. Such attacks employ techniques from tests of non-randomness on a specific event of the concerned cipher.

The basic model for distinguishing attack is built on principles of statistical hypothesis testing. By theoretically analyzing the cipher, a specific event is identified, which occurs with probability p in the cipher keystream and with probability q for a random stream, where p and q are different. Given a stream, the null hypothesis is that the event in that stream occurs non-randomly (i.e. the stream under consideration is indeed from the cipher keystream) and the alternative hypothesis is that the event occurs randomly (i.e. stream under consideration was generated from a random source). If such an event can be identified for a cipher, then that event is called a DISTINGUISHER and identifying the event is a DISTINGUISHING ATTACK.

In this talk, we present two case studies, one for the popular stream cipher RC4 and the other for Py (Roo). For RC4, we present four distinguishers and for Py, we present one distinguisher.

Technical Session I

Attack Mechanisms

Hybrid Analysis of Executables to Detect Security Vulnerabilities

Pranith Kumar D
Dept. Computer Sc. & Engg.
IIT Kharagpur
Kharagpur, WB 721302, India
Email: bobby.prani@gmail.com

Anchal Nema
Dept. Computer Sc. & Engg.
IIT Kharagpur
Kharagpur, WB 721302, India
Email: anchaliitkgp@gmail.com

Rajeev Kumar
Dept. Computer Sc. & Engg.
IIT Kharagpur
Kharagpur, WB 721302, India
Email: rkumar@cse.iitkgp.ernet.in

***Abstract*—Detection of vulnerabilities in executables is one of the major problems facing the software industry. Cracking has increased the complexity where even the programs from reputed companies might be made malicious if they are vulnerable. The main challenge in analysis of executables is due to the unavailability of the source code. Results generated using dynamic analysis alone are unsound, i.e., they do no generalize. On the other hand, results generated using static analysis are usually conservative and it will report weaker properties which might be true but useless. Also, using a model in static analysis which is complete, will render the analysis complex and slow. In this work, we present a hybrid approach, which uses a combination of static and dynamic analysis to identify vulnerabilities. In this approach, we first instrument the executable to extract the control flow of the program. We get the assembly code of the instructions being executed. We then perform static analysis on the assembly code using control flow and register bounds. Static analysis uses slicing on the disassembled code and constraint bound checking is performed on the slice generated. In this way, we exploit the synergy between static and dynamic analysis. Memory errors including memory leaks, buffer overflow and dangling pointers found during the hybrid analysis are reported.**

***Index Terms*—Hybrid analysis, security vulnerabilities, malicious code, memory errors, instrumentation, slicing, validation.**

I. Introduction

Errors introduced by programmers can also be exploited by malicious code writers. These errors can be logical or are introduced when programmers do not take sufficient care at the time of writing the code. For example, low level programming languages like C and C++ provide the flexibility for programmers to have arbitrary memory accesses within the programs address space in the form of pointers. When the programmers are not careful enough when using these features, it might result in illegal memory reads and writes. These illegal reads and writes are then exploited by malicious code writers to tamper with the system.

Buffer overflows accounts for approximately half of all security vulnerabilities [5] Simplest buffer overflow attack, stack smashing [2] overwrites a buffer on the stack to return the return address.When the function returns, control will jump to the address that was replaced on the top of the stack by the attacker which provides him the ability to execute arbitrary code. Many standard C library functions provides unsafe functions(such as gets) that writes an unbounded amount of user input into a fixed size buffer without any bounds checking which can be exploited by attackers. Further, dangling pointers can also be exploited in much the same way as buffer overruns to compromise system security[15].Dangling pointers are created when a memory block pointed to by more than one pointer is freed using another pointer, whilst this pointer keeps pointing to the freed up space.

Researchers have devised various ways of doing analysis on source code as well as on executables. In presence of source code, it can be checked to follow coding standards listed in [7] to ensure that it is free from known security vulnerabilities like dangling pointers, buffer overflows and others. LCLint [11], a source code analyzer, uses the information provided in semantic comments embedded in source code to detect buffer overflow. In the absence of source code, the above methods cannot be used to detect security vulnerabilities.

Various static and dynamic analysis techniques on the binary code are adapted to detect security vulnerabilities. Both the static and dynamic techniques have their pros and cons.Static analysis is conservative giving rise to weaker properties which may not be useful whereas in dynamic analysis, results generated cannot be generalized for all possible inputs.

In our work, we present a hybrid approach which utilizes the strength of both dynamic and static analysis to efficiently detect security vulnerabilities like buffer overflow, dangling pointers and memory leaks. Executable is first instrumented using PIN to extract the exact control flow and register bounds. Executable is disassembled to get the assembly code. Control flow and register bounds are then used in static analysis in which constraint bound check is performed on the slice generated. Finally memory errors obtained as discussed earlier are reported.

Section II presents the previous work done in analysis, techniques and tools along with the motivation of the work. Section III-A discusses the overall architecture of the approach adopted to detect memory errors. Section III-B presents the static slicing used in the approach discussed in Subsection III-A. Section III-C provides the details of PIN and its applications in our analysis. Section IV provides an overview of type of memory errors and the approach adopted to detect them. Section IV-A presents the consequences of memory leaks and our method of detection with examples. Section

IV-B presents the hazards of buffer overflow and our method of detection with examples. Section IV-C presents the dangling pointers detection algorithm and the results obtained. Section V concludes the work presented in this paper.

II. Motivation and Background

In the scenario where the source code is available, static analysis can audit the source code for checking the compliance with coding standards. Graff lists several secure coding practices in [7]. Several tools [13] have been developed which check the source code for known security vulnerabilities. These may include checks for buffer overflows, dangling pointers, use of uninitialized variables, etc., Most vulnerabilities are exploited through malicious I/O. So, validating the I/O data in the source code will prevent most of the exploits. This can also be done at runtime through data flow analysis in the absence of the source code [16]. BOON [18], CodeWizard [1], FlawFinder [20], [19] are a few examples of source code analyzers which create a database of general rules which affect the programs. They also check if the program uses any vulnerable library functions like *strcpy(), strcat(), gets()* etc., LCLint [11] is another source code analysis tool which uses the information provided by the programmer in the semantic comments to detect likely buffer overflow vulnerabilities. This requires extra effort on part of the programmer which is not feasible for already written code. This kind of analysis is difficult in the absence of the source code. The binary code can be disassembled to get to assembly, but constructing a model for assembly with the same features as for a higher level language is difficult. Static analysis for binaries usually rely on flow-based methods. In this approach, they statically try to determine the flow of the program and data. Depending on the model constructed, this kind of analysis can be hugely complex [3].

The static analysis done by Tevis in [17] involves the scanning of executable files for software security vulnerabilities. The methodology uses the PE format (designed for software running on Windows NT/XP) and extracts information located in the headers, sections and tables of an executable files, along with the overall content of the file as a means to detect specific anomalies and security vulnerabilities without having to disassemble the code. Anomalies include inconsistent table sizes, zero filled regions of bytes, unknown region of bytes, compressed file placed in the file, sections that are both writable and executable, use of functions that are susceptible to buffer overflow attacks. This methodology extracts a very little information about the behavior of executable.

The work by Bergeron, in [4], uses static slicing of the disassembled code obtained from the executable. The static slicing is useful to extract the code fragment that are critical for the security of the system. Once these fragments are extracted, they are subjected to behavior specifications to decide whether they are malicious or not. The methodology is incapable to find dynamic aspects of the code which can only be found while executing the code.

Hybrid analysis attempts to erase the boundaries between static and dynamic analysis and create unified analyses that can operate in either mode or in a mode that utilizes the strength of both approaches. Static or dynamic analyses can enhance one another by providing information that would otherwise be unavailable. Hybrid analyses would sacrifice a small amount of the soundness of static analysis and a small amount of accuracy of dynamic analysis to obtain new techniques whose properties are better suited for particular cases than purely static or dynamic analysis [6].

Consider the C-code given in Fig. 6. Memory access out of bound (a[10]) occurs in it. Since the source code is generally unavailable we just have the disassembled code obtained from the executable shown in Fig. 7. Whenever a *malloc()* is called, we extract the start address and the size of the memory block using the memory profiler of PIN during the execution to get the upper and lower bounds of the allocated memory space. Therefore whenever a write to a memory block occurs, using static backward slicing, we determine the upper and lower bounds of the register involved in memory write. If the memory access is invalid, we report the memory access error. Upper and lower bounds of the allocated memory area can be easily found using dynamic analysis. Upper and lower bound of any register (related to memory write) can be found easily using static analysis as explained later in Subsection III-B. In this way, we efficiently use the strengths of both static and dynamic analysis to detect memory access errors.

III. Hybrid Analysis

In this section, we discuss the methodology and the major steps of the analysis, namely static slicing and dynamic analysis, which we report in this work.

A. Methodology

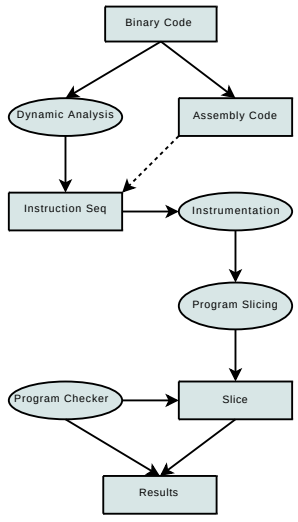


Fig. 1. Overall Scheme

As shown in Fig. 1, in the first step, the executable is dynamically analyzed using PIN tools to get the virtual address

of the instructions being executed. This virtual addresses are then mapped to the corresponding assembly code extracted using the *objdump* of the executable. This gives us the exact sequence of instructions being run on the system. In the second step, we instrument the code to make it analyzable. In the third step, we perform static slicing of this assembly code on constraints extracted during runtime to get a slice of code on which we run the program checker. Here we use hybrid analysis to get the more precise slice. Also we build up tables using dynamic analysis as we explain later and decide upon the contents of these tables if there are any errors present in the executable.

B. Static Slicing

One of the steps in the overall scheme is to slice the binary code to retain only the relevant set of instructions that are useful in analysis. There are basically two types of dependences between the instructions present in the program – Data dependence and Control dependence. Slicing algorithm uses both dependencies between the instructions present in the program. Given the pair (s,V), called the *slicing criterion*, where s is the node in the control flow graph and V is the subset of program variables, it produces a set S_c of program instructions that are relevant for the computation of the variables in V. The set S_C is called a slice. There are two types of slices that can be computed – Forward Slice and Backward Slice. A forward Slice to a program with respect to the *slice criterion*(s,V) at a program point p consists of statements and predicates in the program that may be affected by the variables in V at p. Similarly, a backward slice of a program with respect to the *slice criterion* at a program point p consists of all statements and predicates in the program that may affect the value of variables in V at p.

The algorithm presented in this section uses Procedure dependence graph (PDG). PDG is constructed using Control Flow Graph (CFG) by adding new data-dependent and control-dependent edges between the nodes of the program. Each node in the PDG represents the different instruction and each edge is either a control dependent edge or a data dependent edge from node i to node j for i, j belong to the node of the program. CFG is constructed using PIN instrumentation and is discussed later in this paper.

Intuitively, a statement j is data dependent on statement i if a value computed at i is used at j in some program execution. Control dependence is defined in terms of post dominance. A node i in the CFG is post-dominated by a node by j if all paths from it to end pass through j. A node j is control dependent on a node i if(i)there exists a path P from i to j such that j post-dominates every node in P, excluding i and j, and(ii)i is not post-dominated by j.

In this work we show analysis on the PDG of the program. This work can be extended to System Dependence Graph (SDG) formed by the collection of PDGs. For vertex s of PDG G, the slice of G with respect to s, denoted by slice(G,s) are all the set of nodes reachable from s in the PDG. Thus, slicing a program can be viewed as a graph reachability problem [8] in the Procedure Dependence Graph.

C. Dynamic Analysis

Dynamic analysis is the method of analyzing the properties of a running program. In contrast to static analysis, where we examine a program's text to derive properties which hold for all possible executions, in dynamic analysis we extract properties which may hold for only that particular execution. The advantage here being that, the properties extracted are accurate. This kind of analysis is usually done through instrumentation [12]. Dynamic analysis is helpful in detecting violations of certain properties and also provides useful information to programmers about the behavior of their programs.

We specifically use *itrace* and *malloc*. *itrace* extracts the virtual addresses of the intructions being executed and *malloc* extracts the start address and size of the dynamic memory blocks allocated. We use PIN instrumentation library [9] to perform dynamic analysis on the executable. PIN allows us to instrument the executable by placing calls at arbitrary locations. It provides an easy to use and rich sets of APIs. It includes several sample architecture independent Pintools like profilers, trace analyzers. We use the instruction trace tool and the memory profiling tool [9].

In the instruction trace tool we extract the virtual addresses of all the instructions being executed. We now map these addresses to the addresses in the disassembly of the executable and extract the corresponding instruction being executed. (The disassembly is extracted by using generic tools like objdump). By this method, we get the exact control flow of the program. The CFG so constructed is efficient since the set of possible targets of indirect jump sites is reduced by using dynamic information similar to the work in [10]. A loop running for five times will have all its instructions appear five times in this flow. Also, we use a memory profiling tool *malloc* which gives us the address of a memory block and the size of the memory block during allocation time. We use these to form the Allocated Memory Table (AMT) the contents of which are explained in the next section.

IV. Detection of Memory Access Errors

In this paper, we concentrate on finding memory errors which are the main source of vulnerabilities in an executable, because of which programmers spend much time looking for these errors, but end-users may experience them first. The survey in [14] empirically shows the continuing prevalence of access errors in many widely used UNIX programs.

We maintain an Allocated Memory Table (AMT) which stores the various information regarding a given block. The information pertains to the start address of a memory block, the size of the memory block. These together give us both upper bound and lower bound address for that memory block. We also store a list of pointers pointing to this memory block which is calculated using the slicing algorithm. There is also a parent section for each memory block which helps us to keep

track of memory blocks which point to other memory blocks. A memory block which points to another memory block is its parent. There is also an active field which indicates whether an allocated memory block is active, i.e., in use or has been freed (inactive). The exact use of these fields is illustrated with examples in the Application section.

Memory access errors are typically classified into three types. They are – memory leaks, array index out of bounds (buffer overflows) and dangling pointers.

In this paper, we concentrate on finding such memory errors which are the main source of vulnerabilities in an executable.

A. Memory Leaks

Memory leak is a scenario in which memory allocated to a program is no longer accessible to the program due to some logic flaw on the part of the programmer. Memory leaks are more difficult to detect than other memory access errors. Memory leaks occur because a block of memory was not freed, and hence are errors of omission, rather than commission. In addition, memory leaks rarely produce directly observable errors, but instead cumulatively degrade overall performance.

A memory leak can diminish the performance of the computer by reducing the amount of available memory. Eventually, in the worst case, too much of the available memory may become allocated and all or part of the system or device stops working correctly, the application fails, or the system slows down unacceptably due to thrashing.

1) Challenges: Once found, memory leaks remain challenging to fix. If memory is freed prematurely, memory access errors can result. Since access errors can introduce intermittent problems, memory leak fixes may require lengthy testing. Often, complicated memory ownership protocols are required to administer dynamic memory.

Incorrectly coded boundary cases can lurk in otherwise stable code for years. Both memory leaks and access errors are easy to introduce into a program but hard to eliminate. Without facilities for detecting memory access errors, it is risky for programmers to attempt to reclaim leaked memory aggressively because that may introduce freed-memory access errors with unpredictable results.

Conversely, without feedback on memory leaks, programmers may waste memory by minimizing free calls in order to avoid freed-memory access errors. A facility that reported on both a program's memory access errors and its memory leaks could greatly benefit developers by improving the robustness and performance of their programs.

2) Detection of Memory leaks: We consider the following two cases of memory leaks:

Case 1. *When the last Reference to the memory block is overwritten:*

Consider the code snippet given in Fig. 2 for Case 1. All the C codes including Fig. 2 are presented along with the assembly code for the better understanding of the latter. However, these codes are not used, since the analysis is done on the disassembled code obtained from the executable assuming the absence of the source code.

```
void mem_leak1(void)
{
        char *p;
        p = (char *)malloc(10 * sizeof(char));
        // last reference lost
p = (char *)malloc(10 * sizeof(char));
        return;
}
```

Fig. 2. Memory leak example C code

The corresponding assembly code extracted using PIN is given in Fig. 3.

```
mem_leak1:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        movl    $10, (%esp)
        call    malloc
        movl    %eax, -4(%ebp)
        movl    $10, (%esp)
        call    malloc
        movl    %eax, -4(%ebp)
        leave
end
```

Fig. 3. Assembly code of Fig. 2

During program execution we dynamically construct the AMT (Allocated Memory Table) as explained before. After the execution of the first three instructions, the AMT is given in Table I.

```
movl $10, (%esp)
call malloc
movl %eax, -4(%ebp)
```

We keep track of the memory blocks which have been allocated and the corresponding pointers which are pointing to these memory blocks. Whenever a pointer is assigned to a new pointer, we keep track of this information using the forward slicing as explained.

Now for the next three instructions, which are the same,

```
movl $10, -4(%ebp);
call malloc
movl %eax, -4(%ebp)
```

Here, let us assume the starting address to be 5000. Since the size of the block is 10 the upper-bound for this memory block will be 5009. After the execution of next three instructions, i.e., after the next allocation the table is given in Table II. Here a new memory block is allocated to the same

S.No.	Address	Size	No. Of Ref.	List of Ref.	Parent	Active
1	5000	10	1	-4(%ebp)	-	1

TABLE I
ALLOCATED MEMORY TABLE FOR MEMORY LEAK, CASE 1, STEP 1

S.No.	Address	Size	No. Of Ref.	List of Ref.	Parent	Active
1	5000	10	0	-	-	1
2	5010	10	1	-4(%ebp)	-	1

TABLE II
ALLOCATED MEMORY TABLE FOR MEMORY LEAK, CASE 1, STEP 2

pointer making the first memory block inaccessible.

Here we use forward slicing to update the pointer field of the table. When we slice with respect to the pointer which is currently pointing to a memory block, we get all the pointers if any, to which this pointer has been assigned. At the end of execution we go over the AMT and if we find any memory block which is active and does not contain any references to it, we declare that we found a memory leak.

Case 2. *When the last reference to a memory block is stored in a memory location which is freed, thereby loosing that reference and creating a memory leak.*Consider the C code given in Fig. 4.

```
void mem_leak3()
{
node *n1;
n1 = (node *)malloc(sizeof(node));
n1->next = (node *)malloc(sizeof(node));
free(n1);
n1 = (node *)malloc(sizeof(node));
return;
}
```

Fig. 4. Memory leak example C-code Case-2

The corresponding assembly code for this is given in Fig. 5.

```
movl $8, (%esp)
call malloc
movl %eax, -4(%ebp)
```

The allocated memory table after the first step is given in Table III.

The next step is as follows:

```
movl -8(%ebp), %ebx
movl $8, (%esp)
call malloc
movl %eax, 4(%ebx)
```

The AMT after the execution of the next four instructions is given in Table IV:

```
  mem_leak3:
pushl   %ebp
movl    %esp, %ebp
pushl   %ebx
subl    $20, %esp
movl    $8, (%esp)
call    malloc
movl    %eax, -8(%ebp)
movl    -8(%ebp), %ebx
movl    $8, (%esp)
call    malloc
movl    %eax, 4(%ebx)
movl    -8(%ebp), %eax
movl    %eax, (%esp)
call    free
movl    $8, (%esp)
call    malloc
movl    %eax, -8(%ebp)
addl    $20, %esp
popl    %ebx
popl    %ebp
ret
```

Fig. 5. Assembly code of Fig. 4

We can observe in Table IV that the first allocated memory block contains a pointer to the newly allocated memory block. Hence the parent of the new memory block is the first memory block. Hence both the memory blocks are active at this point of time.

The last remaining step is as follows:

```
movl -8(%ebp), %eax
movl %eax, (%esp)
call free
```

In the instructions mentioned above, the first memory block is freed. We mark this block inactive now. At the end of execution we can the Allocated Memory Table, when we find a block with a parent we follow to that parent block and check if it's active or not. If the parent block in inactive we declare the child block as a memory leak. Here also, we keep track of multiple pointers to the same memory block using forward slicing on the initial pointer. This yields us all the

S.No.	Address	Size	No. Of Ref.	List of Ref.	Parent	Active
1	5000	8	1	-4(%ebp)	-	1

TABLE III
ALLOCATED MEMORY TABLE FOR MEMORY LEAK, CASE 2, STEP 1

S.No.	Address	Size	No. Of Ref.	List of Ref.	Parent	Active
1	5000	8	1	-4(%ebp)	-	1
2	5008	8	1	4(%ebx)	1	1

TABLE IV
ALLOCATED MEMORY TABLE FOR MEMORY LEAK, CASE 2, STEP 2

S.No.	Address	Size	No. Of Ref.	List of Ref.	Parent	Active
1	5000	8	1	-	-	0
2	5008	8	1	4(%ebx)	1	1

TABLE V
ALLOCATED MEMORY TABLE FOR MEMORY LEAK, CASE 2, STEP 3

other pointers pointing to this block, which gets updated in the AMT.

B. Buffer Overflows

A Buffer Overflow, or Buffer Overrun, is an anomalous condition where a process attempts to store data beyond the boundaries of a fixed-length buffer. The result is that the extra data overwrites adjacent memory locations. The overwritten data may include other buffers, variables and program flow data, and may result in erratic program behavior, a memory access exception, program termination (a crash), incorrect results or - especially if deliberately caused by a malicious user - a possible breach of system security.

Buffer overflows can be triggered by inputs specifically designed to execute malicious code or to make the program operate in an unintended way. As such, buffer overflows cause many software vulnerabilities and form the basis of many exploits. Sufficient bounds checking by either the programmer, the compiler or the runtime can prevent buffer overflows.

1) Detection of Buffer Overflow: Here we concentrate on dynamically allocated buffers, since statically allocated buffers, if there are more than one in the current function, are present on the stack and do not have fixed bounds which are detectable in the assembly code. We use *Hybrid Analysis* for solving this case. In static part of the analysis forward slicing is used to keep track of pointers pointing to a memory block.

2) Methodology with Example:

1) When a malloc is called, by dynamic instrumentation using the PIN memory profiling tool we extract the start address of the memory block, it's size and the address of the pointer in which this address is stored in.
2) This pointer is given an upper and lower bound depending on the size of the memory block.
3) When this pointer is assigned to a register which happens during a read or write access, the upper bound and lower bound values are passed onto the register.
4) When a write to a memory area occurs, as in
   movl $20, (%eax)
   we extract the value of eax register and check if its bounds are present and if this is within those bounds.
5) The bounds are calculated using Backward Slicing as explained previously in III-B.

Let us consider the simple example in Fig. 6:

```
void array_oob(void)
{
        char *a;
        char c;
        a = (char *)malloc(10);
        // array out of bound
        // reading and writing
        a[10] = 'a';
        c = a[10];
        return;
}
```

Fig. 6. A Simple Example for Buffer Overflow

The corresponding assembly code for Fig. 6 is shown in Fig. 7.

The first 6 steps are as follows:
1 movl $10, (%esp)
2 call malloc
3 movl %eax, -8(%ebp)
4 movl -8(%ebp), %eax
5 addl $97, %eax
6 movb $1, (%eax)
The corresponding AMT for this code in Table VI. Assume that the start address of the allocated memory block is 5000.

Lower Bound for -8(%ebp): 5000

S.No.	Address	Size	No. Of Ref.	List of Ref.	Parent	Active
1	5000	10	1	-8(%ebp)	-	1

TABLE VI
ALLOCATED MEMORY TABLE FOR BUFFER OVERFLOW

```
array_oob:
..
        subl    $24, %esp
        movl    $10, (%esp)
        call    malloc
        movl    %eax, -8(%ebp)
        movl    -8(%ebp), %eax
        addl    $10, %eax
        movb    $97, (%eax)
        movl    -8(%ebp), %eax
        addl    $10, %eax
        movzbl  (%eax), %eax
        movb    %al, -1(%ebp)
        leave
        ret
```

Fig. 7. Assembly code for Fig. 6

Upper Bound for -8(%ebp): 5009

In the step-6 **movb $1, (%eax)**, we extract the value of the eax register. When we *backward slice* from this step on *eax*, we can see that *eax* was most recently defined in step-4. Here the bounds on *-8(%ebp)* are passed onto the register *eax*.

Now from the AMT we get the upper bounds and lower bounds on this particular register as 5000 and 5009 respectively. Here instructions like addl, subl etc., do not affect the bounds already present on a register since this register is being used here. Only movl like instructions will affect the bounds of the register. If we find out that the value of *eax* is out of these bounds, then we can say that a buffer overflow has occurred.

C. Dangling Pointers

Consider the code given in Fig. 8. Equivalent assembly code (after instrumentation) is given in Fig. 9. We observe the following in the assembly code given in Fig. 9.

```
int *call_func(void);
int main(){
int *p, *q, *r;
p = (int *)malloc(2);
q = p;
*p = 2;
*q = 3;
free(p);  // Memory pointed to by p is freed.
r = (int *)malloc(2);
*r = 500;
*q= 10;   // q is the dangling pointer
here since it is pointing to
the location which is already freed.
printf("\n The value of r = \%d\n" ,*r);
}
```

Fig. 8. Example: Dangling Pointers

```
main:
1: movl %ebp , temp1
2: movl %esp, %ebp
3: subl $24, %esp
4: andl $-16,%esp
5: movl $0, %eax
6: addl $15, %eax
7: addl $15, %eax
8: shrl $4, %eax
9: sall $4, %eax
10: subl %eax, %esp
11: movl $2, (%esp)
12: call malloc
13: movl    temp10 , %eax   #instrumented
14: movl %eax, -4(%ebp)
15: movl -4(%ebp), %eax
16: movl %eax, -8(%ebp)
17: movl -4(%ebp), %eax
18: movl $2, (%eax)
19: movl -8(%ebp), %eax
20: movl $3, (%eax)
21: movl -4(%ebp),%eax
22: movl %eax, (%esp)
23: call free
24: movl $2, (%esp)
25: call malloc
26: movl    temp10 , %eax    #instrumented
27:   movl %eax, -12(%ebp)
28: movl -12(%ebp), %eax
29: movl $500, (%eax)
30: movl -8(%ebp),%eax
31: movl $10, (%eax)
32: movl -12(%ebp),%eax
33: movl (%eax), %eax
34: movl %eax, 4(%esp)
35: movl $.LC0, (%esp)
36: call printf
37: leave
38: ret
```

Fig. 9. Assembly Code of Fig. 8

1) -4(%ebp) is the location where the content of p(refer C code) is saved (line 14).
2) -8(%ebp) is the location where the content of q(refer C code) is saved (line 16).
3) Memory pointed to by p is freed on line 23.
4) On line 30, q is again accessed, after the location it pointed to was already freed (line 23), hence it is a dangling pointer.

1) Instrumentation: We perform the following transformations in the code to enhance analysis.
1. Make the code stackless

pushl ebp → movl ebp , temp1

This is done to make the code stackless and slicing easier. Note that this equivalent code gives the same result with respected to used and defined variables. *ebp* is the used variable in both the cases:
2. Inclusion of instruction 13 and 26

13: movl temp10 , %eax
26: movl temp10 , %eax

The pointer returned after calling malloc is stored in *eax*. To make it explicitly visible in the code. Instruction 13 and 26 are added. *temp10* is a temporary variable. Here *eax* is the defined variable to remove implicit assigning of *eax*

Steps for Detection Algorithm are as follows:-

1) Backward slice with respect to the pointer(say p) which is used in freeing the memory. This gives us all the instructions which can effect the value of the pointer. (Here instruction 22 is used to perform backward slicing. List of instructions obtained are as follows: 22, 21, 14, 13, 10, 9, 4).
2) For each instruction obtained above, perform forward slicing to get the list of instructions.
3) Take the union of all the sets of instructions obtained in step 2 and store them in a cumulative slice array.
4) If there exists an instruction in cumulative slice array which does not contain esp and comes after the instruction *call free* is executed, then report the presence of dangling pointers in the code.

2) Example Results: Forward slices of each instruction in step 2 in our example are shown in Table VII :-

Forward Slice start node	List of nodes obtained in the slice
22	22
21	21,22
14	14,21,22,17,18,15,16,30,31,19,20
13	13,14,21,22,17,18,15,16,30,31,19,20
10	10,35,34,24,22,11
9	9,10,35,34,24,22,11
4	4,10,35,34,24,22,11

TABLE VII
FORWARD SLICES OBTAINED IN STEP 2 (DANGLING POINTERS)

- Cumulative slice obtained in Step 3 is as follows:- 22, 21, 14, 17, 18, 15, 16, 30 ,31, 19, 20, 13, 10, 35, 34, 24, 11, 9 and 4.
- List of instructions that qualify the condition in Step 4 are 30, 31.

The results obtained in Step 4 show the presence of dangling pointer.

V. CONCLUSIONS

Since the source code is not usually available, analysis on the executable has gained importance to ensure trustworthiness in the system. It can be seen that static or dynamic analysis alone is inefficient and insufficient to extract security vulnerabilities. Hence, in this a hybrid approach is taken to detect security vulnerabilities like memory leaks, buffer overflows and dangling pointers. In dangling pointers, the code is made stackless by adding new instructions to retain the semantics of the code in order to enhance the detection analysis. PDG is constructed to get the slice with respect to the slice criterion, so that the analysis is more focussed. PIN tool is used to extract the control flow and register bounds which are used in static analysis to detect the memory errors mentioned above. This approach can also be extended to analyzing components of the system.

REFERENCES

[1] Parasoft. codewizard. parasoft inc. www.parasoft.com, nov. 1, 2003.
[2] Aleph One. Smashing the stack for fun and profit. *Phrack*, 7(49), November 1996.
[3] J. Bergeron, M. Debbabi, J. Desharnais, M. M. Erhioui, Y. Lavoie, and N. Tawbi. Static detection of malicious code in executable programs. *Int. J. of Req. Eng*, 2001:184–189, 2001.
[4] J. Bergeron, M. Debbabi, M. M. Erhioui, and B. Ktari. Static analysis of binary code to isolate malicious behaviors. In *WETICE '99: Proceedings of the 8th Workshop on Enabling Technologies on Infrastructure for Collaborative Enterprises*, pages 184–189, Washington, DC, USA, 1999. IEEE Computer Society.
[5] C. Cowan, P. Wagle, C. Pu, S. Beattie, and J. Walpole. Buffer overflows: Attacks and defenses for the vulnerability of the decade. *Foundations of Intrusion Tolerant Systems*, 0:227, 2003.
[6] M. D. Ernst. Static and dynamic analysis: synergy and duality. In *In WODA 2003: ICSE Workshop on Dynamic Analysis*, pages 24–27, 2003.
[7] M. G. Graff and K. R. V. Wyk. *Secure Coding: Principles and Practices*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2003.
[8] S. Horwitz, T. Reps, and D. Binkley. Interprocedural slicing using dependence graphs. *ACM Trans. Program. Lang. Syst.*, 12(1):26–60, 1990.
[9] C. keung Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. Janapa, and R. K. Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *In Programming Language Design and Implementation*, pages 190–200. ACM Press, 2005.
[10] A. Kiss, J. Jasz, and T. Gyimothy. Using dynamic information in the interprocedural static slicing of binary executables. *Software Quality Journal*, 13(3):227–245, September 2005.
[11] D. Larochelle and D. Evans. Statically detecting likely buffer overflow vulnerabilities. In *SSYM'01: Proceedings of the 10th conference on USENIX Security Symposium*, pages 14–14, Berkeley, CA, USA, 2001. USENIX Association.
[12] J. R. Larus and T. Ball. Rewriting executable files to measure program behavior. *Softw. Pract. Exper.*, 24(2):197–218, 1994.
[13] C. Michael and S. R. Lavenhar. Source code analysis tools - overview.
[14] B. P. Miller, L. Fredriksen, and B. So. An empirical study of the reliability of unix utilities. *Commun. ACM*, 33(12):32–44, 1990.
[15] V. Tenebras, D. Y. Younan, P. Prof, and D. D. Vermeir. de informatica.
[16] J.-E. J. Tevis and J. A. Hamilton. Methods for the prevention, detection and removal of software security vulnerabilities. In *ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference*, pages 197–202, New York, NY, USA, 2004. ACM.
[17] J.-E. J. Tevis and J. John A. Hamilton. Static analysis of anomalies and security vulnerabilities in executable files. In *ACM-SE 44: Proceedings of the 44th annual Southeast regional conference*, pages 560–565, New York, NY, USA, 2006. ACM.
[18] D. Wagner, J. S. Foster, E. A. Brewer, and A. Aiken. A first step towards automated detection of buffer overrun vulnerabilities. In *Network and Distributed System Security Symposium*, pages 3–17, San Diego, CA, February 2000.
[19] D. Wheeler. Secure programming for linux and unix howto. david a. wheeler. www.dwheeler.com/secure-programs, oct. 28, 2003.
[20] D. Wheeler. www.dwheeler.com/flawfinder/, oct. 28, 2003.

An Anti-Phishing Application for the End User

Kapil Oberoi
Department of Electronics and Computer Engineering
Indian Institute of Technology
Roorkee, India

Anil K. Sarje
Department of Electronics and Computer Engineering
Indian Institute of Technology
Roorkee, India

Abstract - **Seeking sensitive user data in the form of online banking user_id and passwords or credit card information, which may then be used by 'phishers' for their own personal gain is the primary objective of the phishing e-mails. With the increase in the online trading activities, there has been a phenomenal increase in the phishing scams which have now started achieving monstrous proportions. In this paper we present an Anti-Phishing application for the end user which keeps track of the sites with which the user indulges in financial transactions, scans his e-mail account for mails which appear to have come from these institutions and warns him against suspected phishing e-mails, if the same are detected in his mailbox.**

Keywords – Phishing, Privacy, Security, Spam

I. INTRODUCTION

Committing a crime in complete anonymity, having gained someone else's identity is a dream come true for any criminal. Phishing is a form of online identity theft which employs both social engineering and technical subterfuge to steal consumers' personal identity data and financial account credentials **[1]**.

Prior to the advent of Internet such efforts on part of criminals were limited to isolated individuals, who were lured into divulging their personal information through social engineering. However with the rapid growth of internetworking around the globe and the phenomenal popularity gained by Internet since the early days of 1990's, the going has never been easier for such criminals or 'phishers' as this community has better come to be known as.

The preferred strategy of the phishers today is to send out millions of spam mails to potential targets around the globe, masquerading as if these came from original institutions such as banks, insurance companies etc. These mails urge the recipients to click on the embedded URLs which lead them to fraudulent but apparently official looking phishing websites where the gullible users are made to divulge with their personal information such as passwords, account numbers and such. These are then collected by the phishers from the server side using web tools such as key loggers and used for their personal gains.

Since it first occurred in the mid 1990's by attacking America Online, phishing has become a profound threat to online services provided by financial organizations, ISPs, retailers and governments. On an estimate, almost 5% recipients of phishing e-mails give away their personal information to these phishing sites while they are in operation **[2]**. A survey conducted by Gartner Inc., found that 3.6 million adults lost money due to phishing attacks during the period from Sep '06 to Aug '07, leading to a huge financial loss assumed to be of the tune of $3.2 billion in US alone **[3]** compared to $2 billion lost in the year 2006 **[4]**. This loss is not only due to the financial loss which is borne by the individuals and the financial institutions on account of the fraudulent transactions by the phishers. It is also due to the dent in the confidence and the resultant hesitancy of prospective clients of making use of web based businesses and services from the fear of being duped of their hard earned money.

There are a host of reasons responsible for the success of phishing attacks **[5]** a few of which are: lack of user's computer knowledge, use of obfuscated URLs, rapid technological advancements in the field of computer science, lack of awareness amongst internet users about elements phishing for their sensitive information etc.

In this paper we present a desktop based Anti-Phishing application for a naïve user against spoofed website based phishing attacks. The design of the application is based on the premise that a user is more susceptible to fall victim to a phishing e-mail which appears to have been sent from an institution like a bank, insurance company, investment company or an e-commerce site with which he has an existing relationship rather than from one with which he has no relationship. So if the user receives an e-mail claiming to be from, say Axis bank, but he does not have an account with them, then he is unlikely to forward any sensitive information to the phishing site. The application keeps track of names and URLs of websites with which the user has a relationship, scans the e-mail account of the user for e-mails which apparently have been sent by these institutions, looks for embedded URLs in these messages and generates a phishing warning for the mails which appear to be phishing e-mails.

The paper is structured as follows: In the next section we talk about various types of phishing attacks. Section 3 describes the design and working of our application along with a live example of how a warning about a phishing e-mail is

generated. In Section 4 we discuss the related work. Section 5 talks about the future work followed by Section 6 wherein we conclude our paper.

II. PHISHING ATTACK VECTORS

An attack vector may be defined as the path taken or the means adopted by a hacker/phisher to reach a destination computer **[6]**. In this section we present an overview of various attack vectors which are adopted by the phishers in order to try and lure the users to reach their phishing sites.

A number of methods have been devised by phishers to trick the users into doing what they want them to do. These methods can broadly be classified into two categories, those which rely on use of spoofed e-mails and websites (social engineering), and others which can be termed as exploit based phishing attacks. Exploit based phishing attacks are more sophisticated then the spoofing attacks and make use of certain inherent weaknesses in the web browsers, which are exploited by phishers to install certain malware in the user's machine, such as a key-logger or a screen-grabber, and use the same to steal information. The proposed design of the Anti Phishing Module revolves around the ways and means to mitigate the spoofing e-mails and web sites attacks.

A. Spoofing E-mails and Web Site

In their earlier days, the phishing attacks were e-mail based wherein the users were sent spam mails asking them to verify some form of their personal information via a reply e-mail. Today however it is very unlikely that any user will fall pray to such an attack (unlikely but not impossible). The primary reason being that users today understand that the financial institutions do not carry out sensitive transactions such as account verification through e-mail (it is, however, to be noted that there is a need to educate the users about safe online transactions in developing countries like India where Internet banking is a relatively recent phenomenon, and an average user is not aware about the unseen threats posed by the phishing community). Such organisations use their websites to provide interactive services to their clients which allow them to make use of encrypted web pages, such as use of SSL technology.

Many phishing attacks today, therefore make use of a combination of spoofed websites and e-mails to try and extract sensitive information from the users. These attacks generally employ some form of URL obfuscation techniques **[7]** to trick the user into visiting fake web sites which look and feel exactly similar to the original websites. More often than not the e-mails sent to the users ask them to verify their credentials with the organisation at the earliest (usually within 24 hrs) by clicking on an embedded URL. The weblink leads the user to an authentic looking but fake website of the organisation or even to the authentic website of the organisation but with obfuscated login and password dialog boxes using borderless windows. Some attacks also make use of hidden frames, images or Javascript code to control the way a webpage is rendered on the user's browser.

Lately, in addition to the use of e-mails, phisher community has started exploiting the Instant Message services to lure users into visiting the phishing sites **[9]**. During the chat sessions the phisher tries to convince the user to visit the spoofed website, the obfuscated URL for which the phisher forwards during the chat session itself, and divulge with his personal information.

B. Exploit Based Attacks

Exploit based attacks are more sophisticated when compared to the family of attacks described above. These attacks exploit some inherent weaknesses in the users' browsers or install some other malware such as a key-logger or a screen grabber which are programmed to keep tab of the user activity over his computer **[8]**. The data so gathered may be collected by the phisher through continuous streaming, local collection and batching of information which may then be uploaded on the phisher's server subsequently or by use of Trojan programs which allow the attacker to collect the user information as and when required. In addition, an attacker may use HTML or DHTML **[8]** to manipulate the display of information on the users' web browsers. These can be used to (a) Deliver additional content such as overriding page contents or graphics. (b) Executing screen grabbing / key logging observation code. (c) Provide a fake secure https wrapper for sites content, i.e., display a fake image of padlock at an appropriate location on the browser. (d) Hiding HTML code from the customers. (e) Loading images and HTML content in the background for later use by a malicious application.

Countering such attacks is beyond the scope of the Anti-Phishing application discussed in this paper. To counter these attacks what is required is that these security issues be taken up by the respective browser manufacturing teams who should try and fix these security bugs. The aim of our application is to try and mitigate spoofed e-mail and web site attacks by forewarning the users about presence of phishing e-mails in their e-mail account, thus reminding him to tread with caution when dealing with such mail messages.

III. DESIGN OF ANTI PHISHING APPLICATION

This anti phishing module is based on the following premise:-

(a) A user is more likely to fall victim to a phishing attack if the phishing e-mail received by him seemingly came from a financial/trading institution with which the user has a transaction relationship.

(b) For a naïve and inexperienced user, it would be better that the task of checking the authenticity of the URLs embedded in the e-mail be left to an application which cannot be fooled by the obfuscation techniques employed by the phishers.

A. *Main Functionality*

As brought out above, this application works on the premise that a user is more worried about the authenticity of the e-mails which appear to have come from institutions with which he has a relationship, rather than e-mails received from all and sundry. As an example, consider a user who has an account with the ICICI Bank and not with Axis Bank. This user then is more likely to fall prey to a phishing mail which claims to be from ICICI Bank rather than a mail which asks her to verify her details with Axis Bank.

To ascertain the websites which are of interest to a user, there is thus a need of user interaction with the application. Accordingly the application initially asks the user to enter the name and the trusted URLs of such institutions/websites where he sends his login details, i.e., username and password. The application fetches the IP Address corresponding to the URL from the DNS server, calculates its message digest (using MD5) and stores this data in a table within the database.

On its subsequent run, the application connects to the e-mail service provider of the user (in this case Gmail from Google) and scans for URLs in the message bodies of only those messages which appear/claim to have been sent by the websites of interest to the user. IP Addresses of URLs so located are fetched by the application and their MD5 values calculated. These values are compared against the values stored in the database for that institution. A warning message which includes subject fields of all the e-mails which report a mismatch in the values of message digest are then displayed, cautioning the user that these mails are suspected to be phishing mails, as shown in figure 1 below.

B. *Connecting to E-Mail Server*

POP3 (Post Office Protocol version 3) and IMAP4 (Internet Message Access Protocol) are the two most prevalent Internet standard protocols used by the local e-mail clients for e-mail retrieval from a remote server over a TCP/IP connection. Although IMAP4 is more user friendly and offers a host of facilities to the users, it is not supported by most of the ISPs (e.g. Yahoo mail does not support IMAP4). The wide popularity of the POP3 protocol is largely due to its appeal to ISPs, not to the users. Using the POP3 protocol, ISPs can elect not to allow the user to leave a copy of the mail on the Mail Server, thus minimizing hard drive storage space.

The present prototype of the Anti Phishing Module uses POP3 to connect to the Gmail account of the user to retrieve the desired information (Gmail incidentally supports both POP3 and IMAP4). POP3 works over a TCP/IP connection using TCP on network port 110. Gmail however uses the deprecated alternate-port method, which uses TCP port 995.

One of the major disadvantages of using POP3 for mail retrieval is that it does not distinguish between a new message and a message which has already been fetched. As a result every call made to run the application results in it checking the e-mail inbox folder from the very beginning. Although this does not seem to be much of a problem in case there are a limited number of messages in the user's inbox, the time taken to complete run of the application increases manifold in case there are say a couple of thousand messages in the inbox folder (in our trial runs over an inbox containing about 300 messages, the application took an average of 6-7 minutes to give the result).

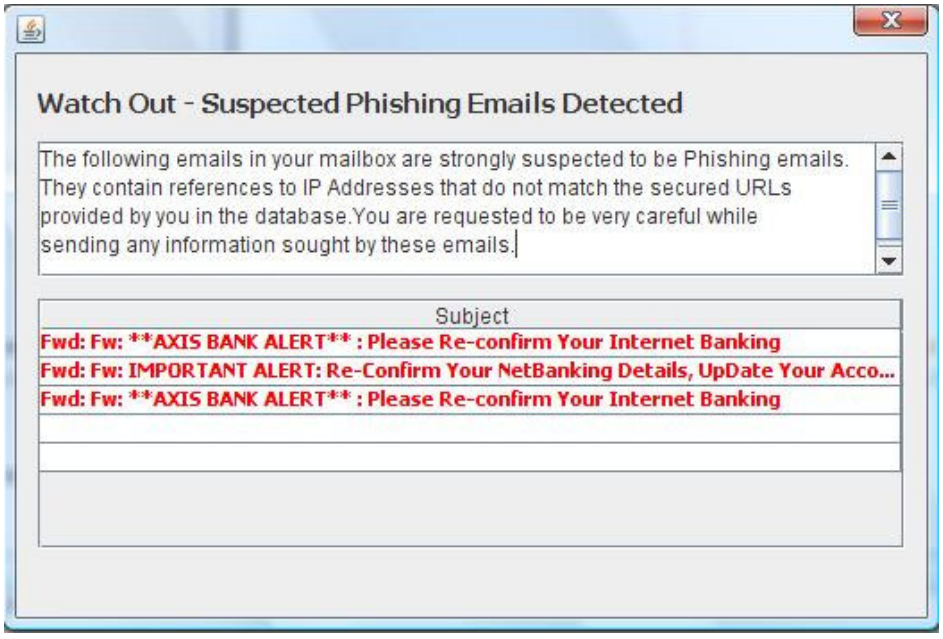


Figure 1: Warning message to the user

To get over the problem, this application makes use of the Sent Date field of the e-mail header (POP3 does not support Received Date field). The maximum value of the Sent Date field of all the messages checked is saved by the application and in the next run is starts checking the messages whose Sent Date is 5 days behind this maximum value. This is done because:

- It might so happen that due to some problem, the mails sent to user's e-mail server get delayed and are not delivered by the time when the application is run.
- The average life time of a phishing site is 5-6 days. Thus it may safely be assumed that if a mail is sent 5 days back and is yet to be delivered to the user it would have been rendered harmless by the time it arrives in his mailbox.

The workflow chart of the application is given in Fig 2.

C. Implementation Details

This work has been implemented in Java programming language using the Netbeans 6.0.1 Integrated Development Environment (IDE). The IDE is available as a free download from http://www. netbeans.org/community/releases/60/.

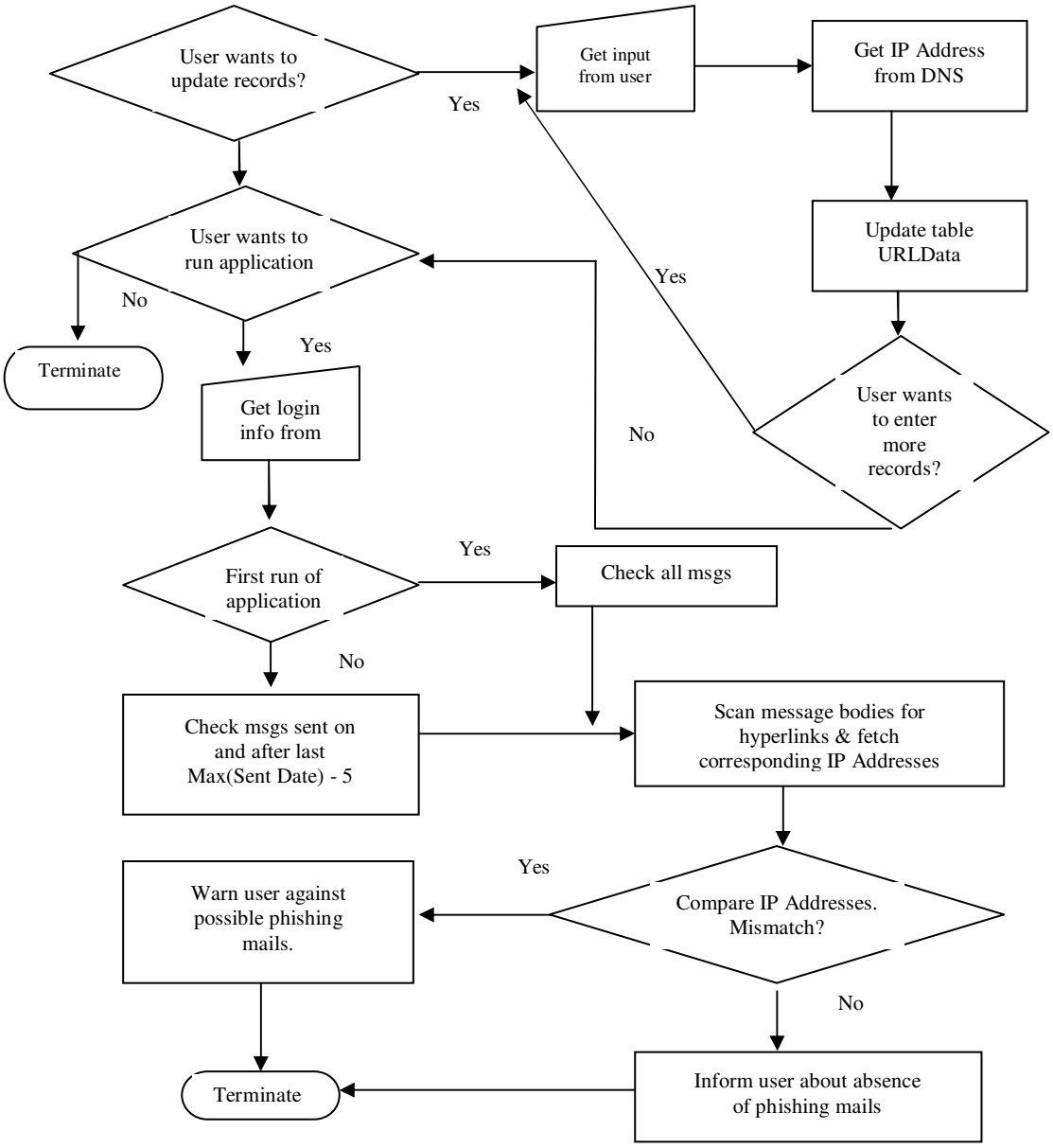


Figure 2. Workflow chart of the Application

Besides the standard Java APIs, JavaMail API has been extensively used in the module. This API is an optional package which is used for reading, composing and sending e-mails. It provides protocol independent access for sending and receiving messages. To use this API, there is a requirement to download JavaMail implementation, unbundle the javamail-[version].zip file, and add the mail.jar file to the project. JavaMail API also requires the JavaBeans Activation Framework, which is also to be downloaded. The framework adds support for typing arbitrary blocks of data and handling it accordingly. After downloading the framework, it is required that we unbundle the jaf-[version].zip file, and add the activation.jar file to the project's CLASSPATH.

For the purpose of handling the data fed in by the user and that fetched from the mail server Microsoft Access Relational Database Management System (RDBMS) has been employed.

D. Working Example

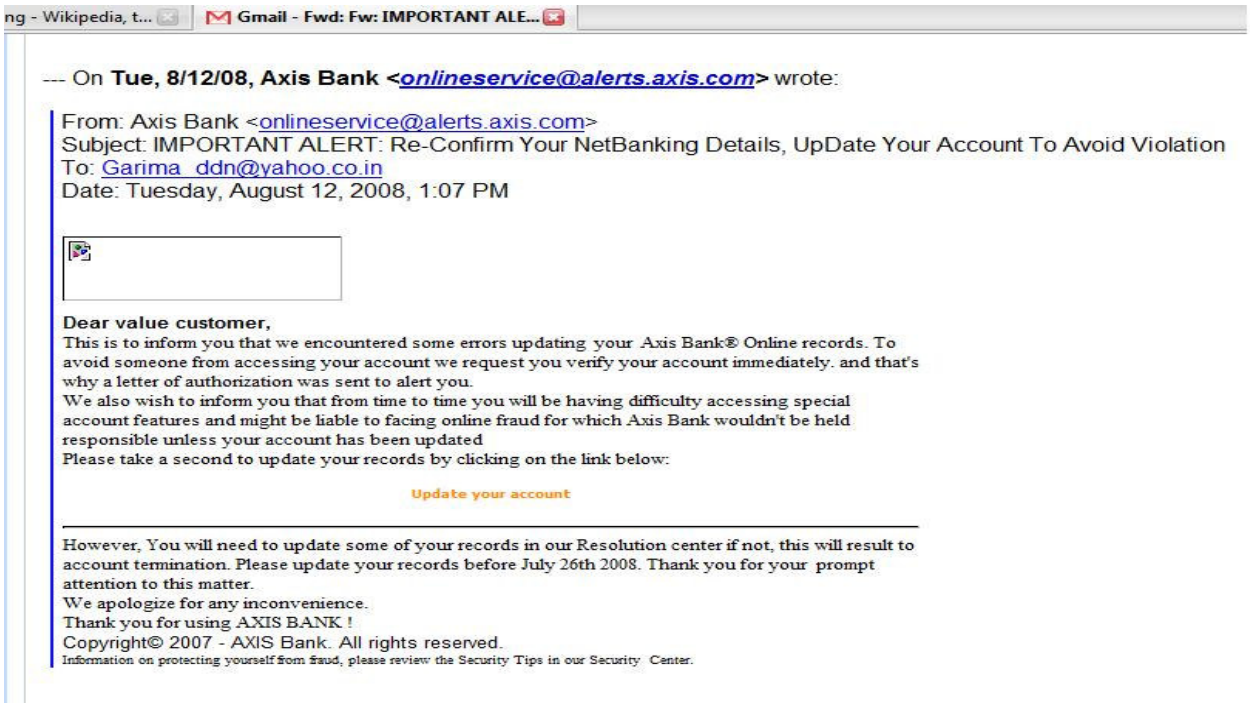
Suppose a naïve user is a customer of Axis Bank and has registered for their online banking services. Also suppose that the said user chooses to use our Anti Phishing application.

When the user runs the application for the first time, he is asked to enter the organisation's name and its secure URL address (as provide to the user by the bank). Accordingly he enters the bank's name as Axis bank and the URL as www.axisbank.com. The application now contacts the DNS and retrieves the corresponding IP address which in this case is 210.210.17.218. The MD5 value of this IP address is then calculated and is stored in the database.

In the next stage of the application, if the user wants to check his e-mail account, he is asked to provide his username and password to logon to his account. Once connected, the application shortlists the mails to be checked, i.e., either all the mails in the user's inbox (if it the first run of the application) or only those whose sent date is at the most 5 days less than the maximum sent date that was stored when the application was run the last time. The application looks for the substring "axis" in the 'From' header field of the short listed messages. Let there be a mail from onlineservice@alerts.axis.com with its subject being "**IMPORTANT ALERT: Re-Confirm Your Net Banking Details, Update Your Account To Avoid Violation**" (refer fig 3).

The message body of this e-mail is scanned for embedded URLs. It should be noted from the phishing e-mail shown in fig 3 that the phisher has tried to hide the identity of the destination URL behind a button titled "Update your account". The trick might fool a naïve or even an experienced internet user but the application's search returns the destination URL as ***http://www.erainfo.es*.** The corresponding IP address of this URL is fetched from the DNS and its MD5 value is matched against the value stored in the database provided by the user. A mismatch produces a warning against suspected phishing e-mails (as shown in fig 1 above).

The user is thus warned against the existence of likely phishing e-mails in his account even before he physically opens his e-mail service. Forewarned about the same, he is unlikely to fall victim of the phisher's trap set for him.


ng - Wikipedia, t... | Gmail - Fwd: Fw: IMPORTANT ALE...

--- On **Tue, 8/12/08, Axis Bank <onlineservice@alerts.axis.com>** wrote:

From: Axis Bank <onlineservice@alerts.axis.com>
Subject: IMPORTANT ALERT: Re-Confirm Your NetBanking Details, UpDate Your Account To Avoid Violation
To: Garima_ddn@yahoo.co.in
Date: Tuesday, August 12, 2008, 1:07 PM

Dear value customer,
This is to inform you that we encountered some errors updating your Axis Bank® Online records. To avoid someone from accessing your account we request you verify your account immediately. and that's why a letter of authorization was sent to alert you.
We also wish to inform you that from time to time you will be having difficulty accessing special account features and might be liable to facing online fraud for which Axis Bank wouldn't be held responsible unless your account has been updated
Please take a second to update your records by clicking on the link below:

Update your account

However, You will need to update some of your records in our Resolution center if not, this will result to account termination. Please update your records before July 26th 2008. Thank you for your prompt attention to this matter.
We apologize for any inconvenience.
Thank you for using AXIS BANK !
Copyright© 2007 - AXIS Bank. All rights reserved.
Information on protecting yourself from fraud, please review the Security Tips in our Security Center.

Figure 3. An example of a Phishing E-Mail apparently from Axis Ban

IV. RELATED WORK

Engin Kirda et. al. **[9]** have developed an anti phishing solution called AntiPhish to guard users against a spoofed web site based phishing attack. The tool keeps track of the sensitive information of a user and generates warnings whenever sensitive information is typed on a form generated by a website that is considered untrusted. One of the drawbacks of the solution is that it lets the user go up to a stage where he is allowed to type in sensitive information on a form and then if the tool finds out that the website is untrustworthy; it warns the user against it. The user is thus susceptible to losing his sensitive data if the phisher employs tools such as a key-logger or a malware which is programmed to send screenshots of the user's console every few seconds.

Neil Chou et. al. **[10]** have proposed an Anti Phishing solution named Spoofguard, which is another plug-in solution developed to provide phishing protection to the end user. It uses a symptom based approach to judge whether a website is spoofed or not. The symptoms include similar sounding domain names, embedded obfuscated URL links, etc. Based on the number of symptoms detected, a phishing alert is generated to warn the user. Moher Aburrous et al. **[11]** have proposed a similar phishing website detection system using fuzzy logic techniques. Fuzzy logic is applied to determine the site status based on 27 parameters which are considered to be the hallmark of phishing sites. These parameters are subdivided into six criteria, (which are further categorized into three layers): URL & Domain identity (Layer 1), Security & Encryption, Source Code & Java Script (Layer 2), Page Style & Contents, Web Address Bar and Social Human Factor (Layer 3). Websites are analyzed based on their overall phishing rating generated and the alert is generated accordingly. Juan Chen et. al. **[12]** have proposed an algorithm named LinkGuard which analyzes the generic characteristics of the hyperlinks in the phishing attacks to deduce whether a site is spoofed or not. The algorithm makes use of a set of rules to analyze the URL viz, mismatch between the actual destination link and the link as seen by the user, use of IP addresses in dotted decimal format, absence of destination information in the text as seen by the user, etc. How our approach differs from the approach suggested in LinkGuard is the fact that our application makes use of user provided data to check the authenticity of the destination URL and hence is able to give a more accurate prediction about the validity of the destination website.

Besides latest web browsers come equipped with anti phishing solutions wherein they maintain a list of black listed sites which are confirmed phishing sites. Every site which the user wishes to open is checked against this list and the operation blocked in case the site appears in the list. This however is, at best, a passive approach against phishing and provides no protection against newly created phishing sites. Also, the quality of protection provided relies heavily on the quality of black list maintained by the browser.

V. FUTURE WORK

This application is just a small step towards tackling the phishing menace an a lot more needs to be done. Some of the features that may be added to this application in order to make it more useful are:

- It is likely that large institutions use multiple servers/mirror sites in order to manage the web traffic on their websites. These mirror sites will have different IP addresses and thus there is a chance that the application may yield false positives when it is run. To mitigate this problem, work is currently underway to fetch and store the digital signature of the login pages corresponding to the authentic URL as provided by the users. Since the source code of the login page of an institution remains the same on all the mirror sites, a mismatch in the same would point to a suspicious website.
- Work is also underway to try and receive alerts as and when the login pages of the institutions change. As and when such an alert is received, the corresponding signature in the database will be amended.
- The present application is a single user application. Making it a multi user application will enhance its utility.
- Integrating the application with web browsers will make it more useful.
- Anti Phishing Working Group (APWG), a pioneer institution formed with the aim to tackle the phishing problem, maintains a black list of sites which are reported upon and confirmed to be phishing sites. Integrating this list with the application so that a passive search of the websites being visited by the user may be carried out and a warning issued in case the site is listed in the black list.
- Automated mechanism to report a phishing site to the authorities so that steps may be initiated to shut them down.
- The email service providers may be approached to integrate the application with their product so that the dependence of the user to access his emails from a single machine, in order to make use of the application, can be done away with. This would also speed up the application significantly since then each incoming mail would be checked for its authenticity. The suspected emails can then be segregated and kept in a separate folder, as is done in the case of spam emails presently.

VI. CONCLUSION

The specter of online identity threat was never so real as it is today primarily due to rapid growth of the Internet and increase in online trading activities which offer a cost effective

method to service providers, such as banks, retailers etc., to reach out to their customers via this medium. This has also provided the phishing community an excellent tool to try and fool the netizans into divulging sensitive information about their banking accounts, credit cards details, etc. Recent years have witnessed a host of phishing scams with each doing the other in terms of reach to the users and the level of sophistication.

Indeed the best measure available against such scams is user awareness. Users should be trained against following blind links and the tendency to part with sensitive information over e-mails which may later cause heavy loss to them. However with the ever increasing reach of the Internet, this in itself is a Herculean task. There is thus a need to develop tools which may be of some assistance to the users in dealing with the menace of phishing. This work to try and develop an Anti Phishing application for the end user is a small attempt in this direction.

REFERENCES

[1] Anti-Phishing Working Group, http://www.antiphishing.org/index.html

[2] Rachna Dhamija, J.D.Tygar, "The Battle Against Phishing: Dynamic Security Skins" in Proceedings of the symposium on Usable privacy and security, pp. 77-88, yr 2005.

[3] Chenfeng Vincent Zhou, Leckie C., Karunasekara S. and Tao Peng, "A Self-healing, Self-protecting Collaborative Intrusion Detection Architecture to Trace-back Fast-flux Phishing Domains" in Networks Operations and Management Symposium (NOMS) Workshops 2008, Salvador, Brazil. IEEE, pp. 321-327, 7-11 Apr 2008.

[4] Gartner Inc., "Gartner Survey Shows Phishing Attacks Escalated in 2007; More than $3 Billion Lost to These Attacks, 2007 Press Release, 17-Dec-2007," https://www.gartner.com/it/page.jsp?id=565125.

[5] Dhamija R., Tygar, J.D. and Hearst, M., "Why Phishing Works" in Conference on Human factors in Computing Systems (SIGCHI 2006), Montreal, Canada, pp. 581-590, 22-27 Apr 2006.

[6] SearchSecurity.com Definitions, http://searchsecurity .techtarget.com/dictionary/definition/1005812/attackvector. html

[7] The Honeynet Project & Research Alliance, "Know your Enemy: Phishing". The Honeynet Project & Research Alliance 2005, http://www.honeynet.org/papers/phis-hing/.

[8] Ollmann, G., "The Phishing Guide". NGS Software Insight Security Research 2005, http://www.ngs software.com/papers/NISRWPhishing.pdf.

[9] Engin Kirda and Christopher Kruegel, "Protecting Users Against Phishing Attacks" in Computer Software and Applications Conference, 2005 (COMPSAC 2005), Edinburgh, Scotland. 29th Annual International Volume 1, pp. 517 – 524, Issue: 26-28 July 2005.

[10] Neil Chou, Robert Ledesma, Yuka Teraguchi and John C. Mitchell, "Client-Side Defense Against Web-Based Identity Theft" in 11th Annual Network and Distributed System Security Symposium (NDSS '04), San Diego, February, 2004.

[11] Maher Aburrous, M.A. Hossain, Fadi Thabatah and Keshav Dahal, "Intelligent Phishing Website Detection System using Fuzzy Techniques" in 3rd International Conference on Information and Communication Technologies: From Theory to Applications, 2008 (ICTTA 2008), pp. 1-6, dt 7-11 Apr 2008.

[12] Juan Chen and Chuanxiong Guo, "Online Detection and Prevention of Phishing Attacks" in Communications and Networking in China, 2006 (ChinaCom '06) pp. 1-7, dt 25 -27 Oct 2006.

New Avatars of Honeypot Attacks on WiFi Networks

Prabhash Dhyani
Wireless Security Researcher,Airtight Networks,Pune
Email: prabhash.dhyani@airtightnetworks.com

Abstract

WiFi has become mainstream technology in offices, homes and public places. While WiFi infrastructure has proliferated over the years, WiFi security awareness lags behind. Honeypot (Evil Twin) is one of the many security threats that WiFi infrastructure and clients face today. Moreover, due to discoveries of several new 802.11 vulnerabilities over last few years, Honeypot threat has also evolved. In this paper, we describe how the recently discovered vulnerabilities can increase sophistication of Honeypot attack thereby exposing even larger genre of clients to this attack. We provide results from our experimentation with these new vulnerabilities in Honeypot attack sequence. Several remedies to protect from these sophisticated Honeypot attacks are also described.

1 Introduction

Honeypot is a type of Wi-Fi attack, where a hacker sets its service identifier (SSID) to be the same as an access point at the local hotspot or corporate wireless network. This results in the unassuming Wi-Fi client connecting to hackers access point (AP) instead of legitimate hotspot or corporate network AP. After the client connects to the hackers AP, it can be exploited in variety of ways.
With newer wireless vulnerabilities and attack tools available to exploit them, conceivable Honeypot attacks can be much more sophisticated than what is possible with traditional attack tools. Due to several new 802.11 vulnerabilities recently discovered, larger number of AP/Client configurations are vulnerable to Honeypot attacks today. Here we describe how the recently discovered vulnerabilities can increase sophistication of Honeypot attack and provide results from our experimentation with these vulnerabilities. This paper is structured as follows: In Section 2, we describe basics of client connection behaviour required to understand how Honeypot attack works. In Section 3, we describe several Honeypot attacks which are conceivable today due to several new 802.11 vulenrabilities that have been discovered in recent years after the era of traditional Honeypot tools. In Section 4, we describe several higher layer attack sequences riding on Honeypot connection that we have experimented with. In Section 5, we present a real life data from several airports showing that large percentage of clients today are vulnerable to Honeypot attacks. Finally some remedies to protect for Honeypot attacks and conclusions are included in Section 6.

2 WiFi Client Connection Behaviour

At the core of honeypot attacks is the WiFi clients connection behavior. APs advertise their presence with security settings and supported rates in their beacons. Alternatively, client uses probe request to look for the available APs in the range and APs respond by sending probe response to advertise their presence to the clients. AP will send their security information and supported rates in probe responses. This is how the AP list is populated in client. Now as the use of wireless for network connectivity is increasing, some client operating systems implement automatic wireless network discovery and known network identification to facilitate wireless networking for the end-user (Windows XP, Apple MacOS). These operating systems (most of the end users use these two Operating Systems) remember the networks the client has connected to in the past. They define two network lists, PNL (preferred network list), the network list to which client connected to in the past and ANL (Available network list) , the APs available at this time.

The algorithm below describes the way client parses ANL and PNL to connect to wireless networks, and where it exposes the network names (SSIDs) client is willing to connect to. First client scans for available networks by sending "any" probe request and then prepares an ANL by probe responses(3).Then it parses PNL and if there is a match connects to that AP(6). If there is no match in the ANL and PNL,

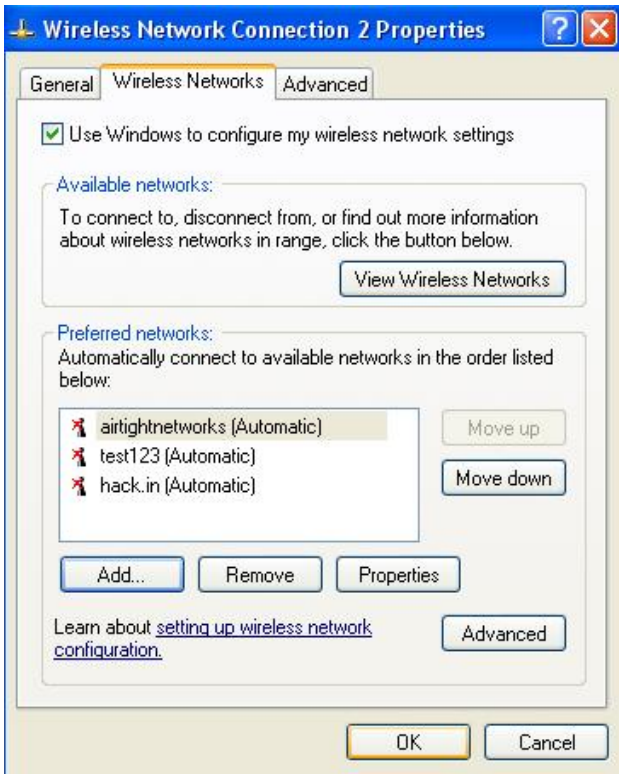


Figure 1: Preferred Network List(PNL)

then client starts sending probe requests for those APs which are in the PNL(9). Here the client leaks out the information of network names (SSID) to which it is willing to connect. Attacker can use this information to set up an AP/Soft AP with the same SSID so that client will connect to it. For the misconfigured clients, it becomes easier to launch honeypot attacks. If client had connected to any ad-hoc network in the past, it configures its wireless interface in the ad-hoc mode in which case any attacker can connect to that client(12).

```
1.Begin:
2.State = Unconnected
3.AvailableNetworks = ScanForAvailableNetworks()
4.Foreach n in PreferredNetworks
5.          if AvailableNetworks contains n
6.             then ConnectToWirelessNetwork(n)
7.                  if State == Connected then return
8.foreach n in PreferredNetworks
9.             ConnectToWirelessNetwork(n)
10.                 if State == Connected then return
11.if PreferredNetworks contains an Ad-Hoc network
12.   then ConfigureAdHocNetwork
13.else
14.      if ConnectToNonPreferredNetworks == True
15.            then foreach n in AvailableNetorks
16.                 ConnectToWirelessNetwork(n)
17.                      if State == Connected then return
18.SetSSID(GenerateRandomSSID())
19.SleepForOneMinute()
20.Goto Begin
```

Automatic Wireless Selection[1]

3 Expanded Genre of WiFi Clients Vulnerable to Honeypot Connections

Initially, honeypot attacks worked only on Open SSIDs. Then advanced tools were created (e.g., hotspotter[2]) which passively listens for the probe requests (refer section 2, how it exposes SSID) and if any probed SSID matches with the given list of SSIDs, becomes AP with the same SSID and could run some commands pre-AP mode and post-AP mode (e.g., running DHCP server etc). This tool needed a SSID list which was compared with the probed SSIDs. Exploiting the same vulnerability (described in section 2), more advanced tools were made (e.g., Karma[3]) which passively listen for probe requests and automatically puts wireless interface in AP mode with the same SSID and can start some configurable services (DNS, DHCP, POP3 etc) to steal clients credentials. Some tools were specifically created to force AP reveal their SSID by deauthenticating connected clients and then listening for probe responses in the reassociation phase (e.g, essid-jack[4]). All these initial honeypot tools work well with the limitation of only clients connecting to Open APs being susceptible to attack. Clients configured to connect to Open APs were vulnerable to traditional honeypot attacks as we could send any data (dhcp, dns etc) to the client. Over the years, several new vulnerabilities were discovered in 802.11 protocol and client connection behavior. These vulnerabilities can be incorporated into honeypot to greatly increase the genre of WiFi clients which are vulnerable to honeypot attacks.

3.1 WEP Clients Connected to AP

Attacking clients communicating with WEP APs is easy because of the several weaknesses of WEP encryption. Weaknesses like weak IV, short IV space, no replay protection , weak integrity check etc. have been used by different tools to crack the WEP key. Fluhrer,

Mantin and Shamir published their famous FMS paper[5] which demonstrated key recovery attack on WEP in 2001. This paper was based on the idea of weak IV (initialization vector) because of which bytes of the keystream can be recovered by guessing first few bytes of plaintext. With a large number of packets (4,000,000 to 6,000,000) , the success probability of the attack is $\tilde{5}0\%$. In 2004, KoreK[6] found 16 additional correlations between first l bytes of an RC4 key, the first two bytes of the generated keystream, and the next keybyte K[l]. The number of captured packets reduced to about 700,000 for 50% success probability with this attack. In 2007, a new generation of WEP attacks was published by Tews, Weinmann, and Pyshkin (PTW attack)[7]. This attack needs just about 35,000 to 40,000 packets for 50% success probability, which can be collected in less than 60 seconds on a fast network. All these WEP attacks require a large number of encrypted packets to crack the key, which is easy to get on a busy network. The number of packets can be increased also on the network by replaying packets as WEP has no replay protection. All these attacks have been implemented in aircrack-ng suite[8]. Once WEP key is recovered , impersonating the legitimate AP with correct key is easy in honeypot. Experimental results show that continuously sending deauth packets to the legitimate AP will force the client to connect to honeypot AP (even with the weaker signal strength).

3.2 WEP Clients Not Connected to AP

For the clients using Open Auth/WEP encryption configuration and not connected to any AP, WEP key configured in the clients can be recovered using the concept of caffe-latte attack. Caffe-latte is used to gather WEP encrypted packets from the client only, it is not a key cracking tool itself. WEP cracking tools require 65k WEP encrypted packets (refer section 3.1) to crack the WEP key. To generate this amount of traffic only by the client, caffe-latte[8] exploits the WEP checksum behaviour, to modify the encrypted data on the fly by flipping some bits and send it back to client to get more packets. Client, when associated (It will be successful in associating as client is using Open Auth) to honeypot will send a Grat-ARP packet advertising its IP address. This Grat-ARP can be modified to become an ARP request by flipping some bits and then is sent to the client again. Client will respond to this ARP request giving us one more encrypted packet. This can repeated till we get sufficient number of packets to crack the WEP key.
Attacking clients using Shared Key Auth/ WEP encryption is a bit complex as clients send Grat-ARP only after association. Client using Shared Key Auth/ WEP encryption would not associate without the correct key. Experiments show that these clients can be attacked by setting up an AP at application layer (custom software sending beacon frames and having response handlers for particular requests) and exploiting the fact that WEP uses one way authentication (AP authenticates client, client does not authenticate AP). In a typical scenario of Shared Key Authentication, there is a 4 way handshake before association:

- Client sends authentication request.
- AP sends 32 byte cleartext challenge.
- Client encrypts this challenge with its WEP key.
- AP decrypts the message with its WEP key, and if the challenge text matches, sends Auth success to client.

Fourth message can be sent as success irrespective of the encrypted challenge, so the client associates to software based AP, and then we can use the same bit-flip concept to modify packets to get enough number of encrypted packets to crack WEP key. Integration of such tool with honeypot will make all WEP configured clients vulnerable to attack.

3.3 WPA-PSK Clients Connected to AP

Attacking clients using WPA-PSK security is difficult because of the security enhancements in WPA-PSK. It is still vulnerable to dictionary attacks, if strong shared key is not used. For this attack attacker needs to capture the initial association phase of the client with the AP. Attacker can passively wait for a client to associate to the AP or can deauthenticate an associated client to capture the EAPOL handshake in the reassociation phase. The EAPOL key exchange phase gives the attacker all the information needed in cleartext to generate our own MIC hashes (with the passphrases given in the dictionary) and compare them with the captured MIC hash in the key exchange phase. If the hashes match, we have found the correct key. Setting up honeypot AP with the correct key and launching DoS attack on the legitimate AP will make client associate to honeypot AP. Though the success of this attack depends on how strong the shared key is and whether it is in the dictionary or not, integrating it with honeypot increases the domain of vulnerable clients.

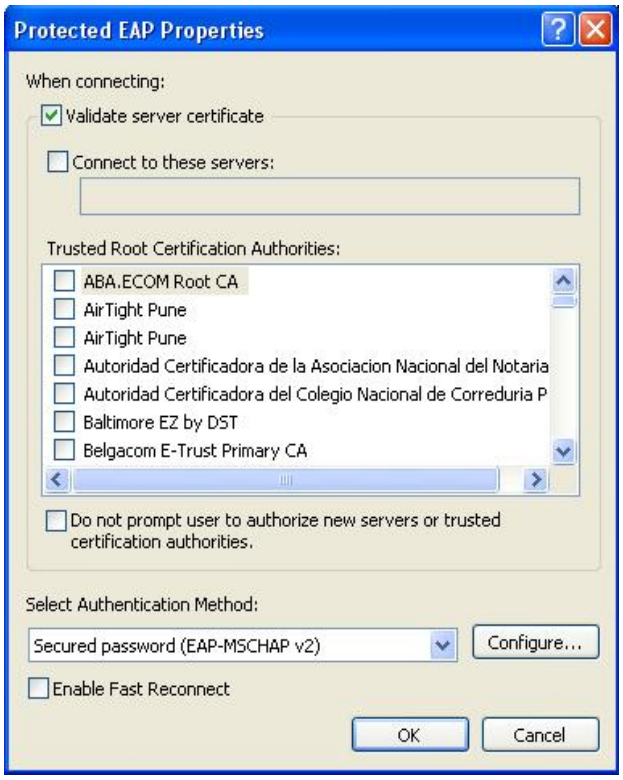


Figure 2: PEAP configuration

3.4 WPA2 Clients Connected to AP

Enterprises are moving to more secure wireless deployments (e.g. LEAP,PEAP) which use Radius server as authentication server. LEAP is also vulnerable to dictionary attacks as it uses modified MSCHAP-v1 for challenge/response[9]. PEAP deployments are more secure as it requires certificate on RADIUS server for client to validate server validity. After server validation, outer tunnel using using TLS secures inner tunnel leveraging legacy authentication. As client receives the certificate from the RADIUS server and then client/server authentication proceeds with inner authentication method, misconfigured clients can grant access to inner authentication methods (e.g. MSCHAP v-2 etc) which are vulnerable to dictionary attacks.

An attacker impersonating the Radius server[10] can provide fake certificate to the clients. Some misconfigured clients may have disabled the server validation (worst configuration) or they can accept it because of ignorance. In both cases ,attacker can get the information needed to mount dictionary attacks revealing username/passwords. This attack needs an AP which is configured to use attackers machine as radius server.A patch for radius server(freeradius-wpe) is available which can be used for logging challenge/response. Hence, misconfigured clients using even WPA2 can be attacked

4 Higher Layer Security Attacks on Honeypot Connections

Past works on honeypot attacks mainly focused on demonstrating layer 2 connections to honeypots. Some works went a step further and demonstrated assigning IP addresses to laptop from honeypot and putting up fake DNS servers to reroute clients connections requests. We have experimented with many higher layer attacks on clients over honeypot connections. Once HoneyPot gets the client associated to it at layer 2 , wireless interface can be bridged with the interface having internet connectivity. In case of network having DHCP server, bridge will get an IP from the DHCP server as well as the client connecting to it. If the network uses static IP, bridge can be given a static IP and DHCP server can be started on the Honeypot itself to give the client an IP from the IP pool. In both cases, honeypot will have IP layer connectivity with the client.
Once layer 3 connectivity has been established, some higher layer attacks can be launched on clients, which are described below:

4.1 Web MITM

Web MITM attack can be launched on clients to steal their credentials if they try to access internet while connected to Honeypot. MITM attacks have been in existence for a long time for wired networks, in which case attacker needed to poison ARP cache of the nodes by changing the MAC address of the gateway to its own MAC address. In wireless scenario, being man-in-the-middle is a lot easier as the client connected to our honeypot has to send all traffic through honeypot. Being an MITM we can log all traffic from the client stealing their credentials. To pass their traffic to the intended website, a transparent proxy server(e.g. delegated) runs on HoneyPot (on port 80 for http and on port 443 for https) while logging all their traffic to some specified log file. Honeypot also spoofs DNS(e.g. dnsspoof) requests from the client to make sure that all traffic goes through our proxy server by resolving all DNS requests to honeypots bridge IP address. In case of https requests, a fake certificate will be provided to the client. Unsuspecting clients accepting that fake certificate will also give honeypot their https

passwords. End users reading the warnings thrown by the browser carefully and not accepting fake certificates are safe from this attack.

4.2 Trojan injection

Trojan injection is also possible in Honeypot. As all http/https request/responses from the client go through our honeypot, it is possible to inject some data on the fly(e.g. ettercap) in HTTP responses. This data can be some malicious javascript (e.g. a pop-up saying This is a software update) followed by a downloadable file (e.g. a Trojan) . Clients accepting this will download the Trojan. Our experiments show that injecting malicious content in the client through browser, given that client accesses internet through our honeypot, is possible.

4.3 Metasploit Attack

This layer 3 attack removes the limitation of client accessing internet through honeypot. This attack is passive and most powerful as it can give the system level access of the client.

This attack uses metasploit framework[11] to attack clients. With many exploits being discovered for different OS very frequently, and end users not applying security patches that often, the probability of client being vulnerable to one of the exploits is very high.

With the latest framework, even knowing which exploit will work for which service on the client is not necessary. All this has been automated (autopwn)[12]. It scans the host for open ports and look for applicable exploits in its database, and if found will executes them one by one. If the client is vulnerable to that exploit, it may give system level access of that client. Integration of MSF with Honeypot makes it very powerful tool for penetration testing of not only networks but also of system security of the systems present in the networks.

5 Case Study

This is a study conducted by Airtight Networks on 23 different airports in the world which shows that still many client/AP configurations use very weak security and are easily exploitable. As use of Wi-Fi by business users is steadily increasing, launching advanced HoneyPot attack on these locations is very easy and many unassuming clients can be attacked. The key findings of the study are:

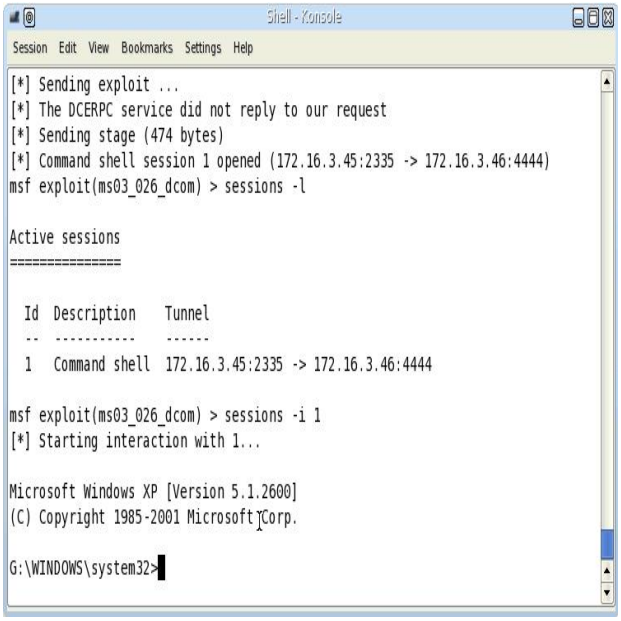


Figure 3: Exploiting Windows Client with MSF

- ~80% of the private Wi-Fi networks at Airports are OPEN / WEP! Honeypot attack on both networks is possible. WEP key can be cracked by aircrack-ng, caffe-latte. Clients can be forced to connect to honeypot by deauthenticating legitimate AP.
- Over 10% laptops found to be using ad-hoc networks! Honeypot can connect to these ad-hoc networks and directly launch layer 3 attacks integrated with it.

6 Conclusion

Experimental results have shown that not only clients which are connected to APs using weak security mechanism are susceptible to attack, but also misconfigured clients using strong security can be attacked. Places like coffee-shops, airports etc. where clients use hotspots to access free internet, are most vulnerable to this type of advanced honeypot attack as most hotspots use Open APs and end users (clients) are also not well aware of the technologies. The possibility of success of higher layer attacks at these locations is very high. Best practices for clients to be safe from this type of advanced honeypot attack are:

- If not using Wi-Fi, it is recommended to turn off wireless interface altogether.
- Unsecure(e.g. Open) or weakly secured(e.g. WEP) Wireless profiles (Preferred Network List) should be deleted.
- Clients should apply security pacthes for the OS as and when they are released.
- End users should not accept any untrusted file/certificate.
- Clients should be configured properly (e.g. server certificate validation enabled)

References

[1] Attacking Automatic Wireless Network Selection http://www.theta44.org/karma/aawns.pdf

[2] Hotspotter-Automatic wireless client penetration http://www.remote-exploit.org/codes_hotspotter.html

[3] Karma Main http://wirelessdefence.org/Contents/KARMAMain.htm

[4] Wireless Vulnerabilities & Exploits ..WVE ID: WVE-2005-0018 http://www.wirelessve.org/entries/show/WVE-2005-0018

[5] Fluhrer, S., Mantin, I., and Shamir, A., Weaknesses in the key scheduling algorithm of RC4, Eighth Annual Workshop on Selected Areas in Cryptography (August 2001)

[6] KoreK. Next generation of WEP attacks? http://www.netstumbler.org/showpost.php?p=93942\&postcount=35,2004

[7] Erik Tews, Ralf-Philipp Weinmann, and Andrei Pyshkin. Breaking 104 bit wep in less than 60 seconds. In Sehun Kim, Moti Yung, and Hyung-Woo Lee, editors, WISA, volume 4867 of Lecture Notes in Computer Science, pages (188-202) Springer, 2007

[8] Caffe Latte attack http://www.airtightnetworks.com/home/resources/knowledge-center/caffe-latte.html

[9] Weaknesses in LEAP Challenge/Response http://asleap.sourceforge.net/asleap-defcon.pdf

[10] PEAP: Pwned Extensible Authentication Protocol http://www.willhackforsushi.com/papers/shmoocon-rfp-joshua-wright.pdf

[11] The Metasploit Project http://www.metasploit.com

[12] Automated Penetration testing with Metasploit Framework http://www.spylogic.net/downloads/Automated_Penetration_Testing_with_the_Metasploit_Framework.pdf

Technical Session II

Security Solutions

An Efficient Secret Sharing Scheme for n out of n scheme using POB-number system

A. Sreekumar and Dr. S. Babu Sundar
Department of Computer Applications
Cochin University of Science and Technology, Kochi, Kerala, India
Email: sreekumar@cusat.ac.in ; dr_s_babusundar@yahoo.com

Abstract

Secret sharing is concerned with the problem of how to distribute and share a secret among a group of n participating individuals, or entities, so that only pre-designated sub-collections of individuals are able to recreate the secret by collectively combining their shares of secret. Sharing schemes are useful in military and civilian applications. In the traditional Secret Sharing Schemes, a shared secret information cannot be revealed without any cryptographic computations. Various Secret Sharing Schemes have been proposed. However, the size of the shares and implementation complexity in these schemes depend on the number of participants. In other words, when a great number of participants are involved, the scheme will become impractical. A secret sharing scheme is called efficient if the total length of the n shares is polynomial in n. In the traditional Visual Secret Sharing Schemes, a shared secret information can be revealed without any cryptographic computations. In this paper we present a method to construct an n out of n Secret Sharing Scheme based on a new number system, called Permutation Ordered Binary Number System (POB-number system). This scheme provides an efficient way to hide a secret information in different shares. Furthermore, the size of the shares is less than the size of the secret.

1. Introduction

Secret sharing scheme is a method of sharing a secret information among a group of participants. In a secret sharing scheme, each participant gets a piece of secret information, called a share. When the allowed coalitions of participants pool their shares, they can together recover the shared secret; on the other hand, any other subsets, namely non-allowed coalitions, cannot recover the secret information by pooling their shares. The collection of subsets of participants that can reconstruct the secret in this way is called access structure. Secret Sharing was introduced by Blakley [4] and Shamir [1] in 1979. Shamir's solution is based on the property of polynomial interpolation in finite fields; Blakley formulated and solved the problem in terms of finite geometries. The first secret sharing schemes considered were threshold schemes. A (k, n) threshold scheme allows a secret to be shared among n participants in such a way that any k of them can recover the secret, but any $k-1$, or fewer, have absolutely no information on the secret. Various Secret sharing schemes were proposed in the past, but most of them need a lot of computations to decode the secret. In 1994, Naor and Shamir [8] invented a new type of Secret sharing scheme called visual cryptography scheme. It could decode the secret (printed text, hand written notes, pictures, etc.) directly, without performing any computation, and the decoder of this scheme was the human visual system. For example, in a (k, n) visual cryptographic scheme, a dealer encodes a secret into n shares and gives each participant a share, where each share is a transparency. The secret is visible if k (or more) of participants stack their transparencies together, but none can see the shared secret if fewer than k transparencies are stacked together.

Until the year 1997, although the transparencies could be stacked to recover the secret image without any computation, the revealed secret images (as in [2] [3] [6] [8]) were all black and white. In [9], Verheul and Van Tilborg used the concept of *arcs* to construct a colored visual cryptography scheme, where users could share colored secret images. The key concept for a c-colorful visual cryptography scheme is to transform one pixel to b sub pixels, and each sub-pixel is divided into c color regions. In each sub-pixel, there is exactly one color region colored, and all the other color regions are black. The color of one pixel depends on the interrelations between the stacked sub-pixels. For example, if we want to encrypt a pixel of color c_i, we color region i with color c_i on all sub-pixels. If all sub-pixels are colored in the same way, we sees color c_i, when looking at this pixel; otherwise one sees black.

A major disadvantage of this scheme is that the number of colors and the number of sub-pixels determine the resolution of the recovered secret image. If the number of colors is large, coloring the sub-pixels will become a very difficult task, even though we can use a special *image editing package* to color these sub-pixels. How to stack these transparencies correctly and precisely by human beings is also a difficult problem. Another problem is that when the number of sub-pixels is b, the loss in resolution

from the original secret image to the recovered image becomes b.

In [7], Hwang proposed a new visual cryptography scheme which improved the visual effect of the shares (the shares in their scheme were significant images, while those in the previous scheme were meaningless images). Hwang's scheme is very useful when we need to manage a lot of transparencies; nevertheless, it can only be used in black and white images. For this reason, Chang, Tsai and Chen proposed a new secret color image sharing scheme [5] based on *modified visual cryptography*.

A major disadvantage of this scheme is that the size of the share increases with the number of participants, i.e., the more the participants, the larger the size of the share. The ratio of the size of one share to the size of the secret is called the information rate.

2. Permutation Ordered Binary Number System

We introduce a general number system, called, Permutation Ordered Binary (POB) Number System with two non-negative integral parameters, n and r, where $n \geq r$. The system is denoted by POB(n, r). In this number system, we represent all integers in the range 0, ..., $\binom{n}{r}$ -1, as a binary string, say $B = b_{n-1}b_{n-2}\dots b_0$, of length n, and having exactly r 1s. Each digit of this number, say, b_j is associated with its position value, given by

$$b_j \times \binom{j}{p_j}, \ where \ p_j = \sum_{i=0}^{j} b_i \ ,$$

and the value represented by the POB-number B, denoted by $V(B)$, will be the sum of position values of all of its digits.

$$i.e., \ V(B) = \sum_{j=0}^{n-1} b_j \binom{j}{p_j} \tag{1}$$

It can be proved that, since exactly $\binom{n}{r}$ such binary strings exist, each number will have a distinct representation. In order to emphasize that a binary string, $B = b_{n-1}b_{n-2}\dots b_0$ is a POB-number, we denote the same by using the suffix 'p'. For example, 001101010_p is a POB(9, 4) number with value of 29. However, such a string, regarded as a binary number will have a decimal value of 106.

2.1. POB-representation is unique

We prove that for a given POB-value, the representation as a POB-number is unique.

Theorem 1 (POB-representation is unique): The value of a POB-number, $V(B)$ of $B = b_{n-1}b_{n-2}\dots b_0$ computed by the formula (1) given above, produces distinct values in the range $0, \cdots, \binom{n}{r} - 1$.

Proof: We can easily prove the following:

1) If j is the highest integer with b_j = 1, then
$$\binom{j}{r} \leq V(B) \leq \binom{j+1}{r} - 1. \tag{2}$$
2) If $j = n$, equation (2) gives, $V(B) \leq \binom{n}{r} - 1$.
3) If X and B are two distinct POB-numbers then by equation (2) $V(X) \neq V(B)$.

So, the POB-representation is unique. Hence the theorem.
The equation (1) computes the decimal value : $V(B)$, for a given POB-number : B. In a practical situation, we may need to find the POB-number: B corresponding to a given POB-value: $V(B)$. We give the algorithms for finding the B given $V(B)$ and vice - versa.

2.2. Algorithm for finding POB-number

For a given pair of parameters n and r with $r \leq n$, the algorithm takes three inputs: n, r and $value$ with $0 \leq value \leq \binom{n}{r} - 1$ and produce POB-number corresponding to the $value$.

Algorithm 1: [Generate POB-number corresponding to a given POB-value]
In a POB(n, r) number system, if a POB-value, $value$ is given, the algorithm generates POB(n,r) number :B such that $V(B) = value$.
Input : Three numbers: n, r and $value$ with $r \leq n$ and $0 \leq value \leq \binom{n}{r} - 1$.
Output: The POB-number $B = b_{n-1}b_{n-2}\dots b_0$.

Step 1.Let $j = n$ and $temp = value$.
Step 2.For $k = r$ down to 1 do:
1. Repeat {
2. $j = j - 1$;
3. $p = \binom{j}{k}$;
4. if ($temp \geq p$)
5. $temp = temp - p$;
6. $b_j = 1$;
7. else $b_j = 0$;
8. } Until (b_j = 1);

Step 3.if ($j > 0$)
For $k = j - 1$ down to 0 do:
$b_k = 0$;

Remark: $B = b_{n-1}b_{n-2}\dots b_0$ is the POB-number.

Lemma 1: Algorithm 1 generates the POB-number(n, r) corresponding to the given POB-value.

Proof: At step 2, of the algorithm, a maximum of r b_js will be equal to 1. It may be observed that at any stage of the algorithm, $0 \leq temp$. Further, in any iteration of Step 2, for a k, at $j = k-1$, $p = \binom{k-1}{k} = 0$ and so $temp \geq p$ (in line no. 4 of Step 2) and hence, b_j will be equal to 1, if not so for a higher value of j. Hence, it is clear that, after execution of Step 2, the binary string $B = b_{n-1}b_{n-2}\dots b_0$ will have precisely r 1s and $n-j$ 0s. By Step 3, it will have r 1s and $n-r$ 0s.

Remarks

1) Given two positive integral values n and r such that $n \geq r$, there will be exactly $\binom{n}{r}$ members in POB(n, r).
2) Since $\binom{n}{r}$ is maximum when $r = \lfloor \frac{z}{2} \rfloor$, for most of the applications, POB$(n, \lfloor \frac{z}{2} \rfloor)$ number system is the best.
3) It may be noted that in order to represent a 9 bit POB(9,4) number:B, we need only the 7-bit POB-value:$V(B)$. Whenever we need B, the algorithm 1 can be used to generate it.

3. Secret Sharing Scheme Using POB Number System

In this section we describe the construction details of a 2 out of 2 secret sharing scheme and in the next section, the construction details of an n out of n scheme for $n \geq 3$. The simplest version of the scheme assumes that the secret consists of a sequence of bytes and each byte is handled separately. The construction is based on the following theorem:

Theorem 2: Let T be a binary string of even parity, having length 9. Then we can find two binary strings A and B each having exactly four 1s and 5 0s such that $T = A \oplus B$.

Proof: We can assume, without loss of generality that, the leading $2m, (0 \leq m \leq 4)$ digits of T are 1s and remaining $9-2m$ digits are 0s. Now, let $A = PQ$ be the binary string obtained by concatenating the strings P and Q, where P is a string having exactly m 1s and 0s, and Q is having exactly $4-m$ 1s and $5-m$ 0s. Then the choice $B = \overline{P}Q$, where, $\overline{P}$ is the complement of P, will prove the theorem.

3.1. A 2 out of 2 Construction

Let $K = k_1k_2\dots k_8$ be one byte of the secret information to be shared between two participants. In order to share the byte between two participants, we first extend the byte by inserting a bit at random position, $r, 1 \leq r \leq 9$. The inserted digit will be such that, the resulting extended string T is of even parity. This extended string T is split into two POB(9,4) numbers, according to theorem 2, such that $T = A \oplus B$. The shares S_1 and S_2 are the values $V(A)$ and $V(B)$ represented by thePOB-numbers A and B respectively. The following algorithm (Algorithm 2) describes the construction in detail.

Algorithm 2: [Sharing a byte between two blocks]
Input: A binary string $K = K_1K_2\dots K_8$.
Output : Two blocks S_1 and S_2 of length 7 bits.

Step 1.Let A and B are two 9 bits long integers. Set all the bits of A and B to null, randomly select an integer r in $[1\dots 9]$.

Step 2.The input string K is expanded to T by inserting one bit at position r. Compute the binary string $T = (T_1T_2\dots T_9)$

$$where\ T_i = \begin{cases} K_i, & \text{if } i < r \\ K_{i-1}, & \text{if } i > r \\ 0, & \text{if } i = r \text{ and } K \text{ is even parity} \\ 1, & \text{if } i = r \text{ and } K \text{ is odd parity} \end{cases}$$

Step 3.noOfOne = 0;
For i = 1 to 9 do
if (T_i = 1) then
$noOfOne = noOfOne + 1$;
if ($noOfOne$ is odd) $A_i = 1$;
else A_i = 0;

Step 4.Randomly assign the rest null bits of A to 0 or 1, let A consists of 4 1s and 5 0s.

Step 5.let j = 0.
For i = 1 to 9 do
$B_i = A_i \oplus T_i$

Step 6.Let S_1 and S_2 be the POB-values corresponding to the POB-numbers A and B, respectively.

Algorithm 3: [Recover the secret information]
Input : Two shares S_1 and S_2 of length 7 bits each and the random integer r.
Output: The secret information $K = K_1K_2K_3\dots K_8$.

Step 1.Let A and B be the POB-number corresponding to S_1 and S_2 respectively.

Step 2.
For i = 1 to 8 do
if ($i \geq r$) $j = i + 1$;
else j = i;
$K_i = A_j \oplus B_j$.

Step 3.The recovered secret is $K = K_1K_2K_3\dots K_8$

Lemma 2: The above scheme is a 2 out of 2 secret sharing scheme.

Proof: It may be observed that, in step 2 of Algorithm 2, the extended string T is of even parity. Since the length of T is 9, it can have a maximum of eight 1s. Let T contains $2m, (0 \leq m \leq 4)$ 1s. Then in Step 3, the $2m$ bits of A, corresponding to the 1s in T will be set to 1s and 0s equally. The Step 4 of Algorithm 2, ensures that A contains 4 1s and 5 0s. The string $B = A \oplus T$, computed in Step 5, also consists of 4 1s and 5 0s, as per Theorem 2. So the shares S_1 and S_2, which are POB-values of A and B are 7 bit length each. The condition $B = A \oplus T$ in Step 5, implies $T = A \oplus B$, and if we drop out rth bit of T, we get, K. Thus, the above scheme is a 2 out of 2 secret sharing scheme. Besides, each byte is shared by a seven bit string.

It may be seen that in algorithm 2, the size of shares is only 7 bits, while the size of the original secret message is 8 bits. The new scheme provides a gain of one bit per one byte of secret in its representation.

Example 1:

Let us consider a secret of two bytes, say, K = 11011110 10100001 Let the random numbers generated to share these two bytes be 4, and 3 respectively, so that the extended string T (inserted bits are underlined) is as follows:

Step 2. 110$\underline{0}$11110 10$\underline{1}$100001.

The string A after step 3 and 4 are as follows:

Step 3. 10**$\underline{1}$010* 1*$\underline{0}$1****0.

Step 4. 101010100 100110100

The string $B = A \oplus T$, computed in Step 5 is:

011001010 001010101.

The indices of these codes are 98, 88 and 59, 20.

The final shares are 1100010 1011000 and 0111011 0010100.

Recovery : The codes corresponding to the numbers are as follows: A : 101010100 100110100

B : 011001010 001010101

$T = A \oplus B$ = 110011110 101100001

Deleting the 4th and 3rd bits from the consecutive blocks of T, we get, the secret K = 11011110 10100001.

3.2. A n out of n Construction

Algorithm 4: [Sharing a secret among n blocks]

Input:A single byte string $K = K_1K_2K_3 \ldots K_8$.
Output : n shares $S_1, S_2, \ldots, S_n$ of length 7 bit,

Step 1.Let $A_1, A_2, \ldots A_n$ be n 9 bit strings initialized to null bits.

Step 2.Randomly assign n-2 POB(9,4)-numbers one for each of $A_i, 2 \leq i \leq n-1$. Let r be the quotient obtained, when, dividing $V(A_2)$ by 14.

Step 3.The input string K is expanded to T by inserting one bit at position r.

Compute the binary string $T = (T_1T_2 \ldots T_9)$

$$where\ T_i = \begin{cases} K_i, & \text{if } i < r \\ K_{i-1}, & \text{if } i > r \\ 0, & \text{if i = r and K is even parity} \\ 1, & \text{if i = r and K is odd parity} \end{cases}$$

Step 4.Let $W = T \oplus A_2 \oplus A_3 \oplus \ldots \oplus A_{n-1}$

Step 5.$noOfOne$ = 0;, let $W = W_1W_2 \ldots W_9$

For i = 1 to 9 do
if (W_i = 1) then
$noOfOne = noOfOne + 1$;
if ($noOfOne$ is odd) A_{1i} = 1;
else A_{1i} = 0;

Step 6.Randomly assign the rest null bits of A_1 to 0 or 1, let A_1 consists of 4 1s and 5 0s.

Step 7.Compute $A_n = W \oplus A_1$

Step 8.For i = 1 to n do $S_i = V(A_i)$.

Algorithm 5: [Recover the secret information]
Input : n shares S_1, S_2, ...,S_n of length 7 bits each.
Output: The secret information $K = K_1K_2K_3 \ldots K_8$.

Step 1.Let $A_1, A_2, \ldots A_n$ be the POB-numbers corresponding to S_1, S_2, ...,S_n respectively and r be the the quotient obtained, when, dividing S_2 by 14.
Compute $T = A_1 \oplus A_2 \oplus A_3 \oplus \ldots \oplus A_n$
Let $T = (T_1T_2 \ldots T_9)$

Step 2.
For i = 1 to 8 do
if ($i \geq r$) $j = i + 1$;
else $j = i$;
$K_i = T_j$.

Step 3.The recovered secret is $K = K_1K_2K_3 \ldots K_8$

Lemma 3: The above scheme is an n out of n secret sharing scheme.

Proof: In Step 2, of Algorithm 4, A_is are assigned as random POB(9, 4)-numbers, $V(A_2)$ is a random number in [0, ..., 125] and hence, r is uniformly at random number in [0,...,9]. It may be noted that after Step 3, the expanded string T is of even parity. It is clear that Step 4 of Algorithm 4,

$$W = T \oplus A_2 \oplus A_3 \oplus \ldots \oplus A_{n-1}, \tag{3}$$

from which the following equation holds:

$$T = W \oplus A_2 \oplus A_3 \oplus \ldots \oplus A_{n-1} \tag{4}$$

Further more, since all the A_is are of even parity, W is also an even parity and length 9. The W is written as,

$$W = A_1 \oplus A_n, \tag{5}$$

by using Steps 5, 6, and 7, in the same way as what we have done in the case of Algorithm 2. Substituting (5) in (4), we get,

$$T = A_1 \oplus A_2 \oplus A_3 \oplus \ldots \oplus A_n \tag{6}$$

Finally, the shares, S_is, are POB-values corresponding to the POB-numbers A_is. In order to get the secret K, rth bit of T is dropped out.

Example 2:

For a (5, 5) threshold scheme, secret $K = 10110110$ is taken.

Randomly assign 5 0s and 4 1s to 3 rows $\{A_2, A_3, A_4\}$. Therefore,

$$\begin{aligned} A_2 &= 101100010, \\ A_3 &= 010101001, \; and \\ A_4 &= 110010100. \end{aligned}$$

Let the random number r is the quotient obtained, when, $V(A_2) = 101$ divided by 14. i.e., r = 7.

The expanded string T as per step 3, of Algorithm 4 is $T = 101101110$

Step 4. Computes W = 100110001,
by Step 5., A_1 = 1**01***0, and
by step 6., A_1 becomes = 110010100

By Step 7, $A_5 = 010100101$

The shares are the indices: 113, 101, 48, 113, 46. All the 5 shares are listed below:

$$\begin{aligned} S_1 &= 1110001, \\ S_2 &= 1100101, \\ S_3 &= 0110000, \\ S_4 &= 1110001, \; and \\ S_5 &= 0101110. \end{aligned}$$

Recovery: Compute $T = A_1 \oplus A_2 \oplus A_3 \oplus A_4 \oplus A_5$, and get 101101110. Deleting the 7th bit, we get secret as K = 10110110.

4. Security Analysis

In the construction under the POB(9,4) number system there is a total of 126 shares corresponding to one byte of secret. The probability of a correct guess of a share is $\frac{1}{126}$ per byte of secret. This would mean that for a secret of m-bytes, the probability of correct guess of a share will be as low as $\left(\frac{1}{126}\right)^m$.

5. Conclusion

We have introduced a new number system, called Permutation Ordered Binary Number System. This number system has great potential in Secret Sharing. We have also given an algorithm for generating the shares and recovery of the secret. The proposed scheme is effective, where we have a gain of one bit for every 8 bits of information. The full potential of the newly introduced POB number system is yet to be explored.

References

[1] Adi Shamir, *How to Share a Secret*, Communications of the ACM, Vol. 22, no. 11, pp. 612-613, Nov. 1979.

[2] G. Ateniese, C. Blundo, A. D. Santis, and D. Atinson,*Constructions and Bounds for Visual Cryptography*, Proceedings 23rd International Colloquium on Automata, Languages, and Programming (ICALP '96), 1099, 1996, pp. 416-428.

[3] G. Ateniese, C. Blundo, A. D. Santis, and D. Atinson *Visual Cryptography for General Access Structures*, Information and Computation, vol. 129, no.2, 1996, pp. 86-106.

[4] G. R. Blakley, *Safeguarding Cryptographic keys* Proceeding of AFIPS 1979 National Computer Conference, vol. 48, New York, NY, pp. 313-317, June 1979.

[5] C. Chang, C. Tsait and T. Chen *A New Scheme for Sharing Secret Colour Images in Computer Network*, Proceeding of International Conference on Parallel and Distributed Systems, July 2000, pp 21-27.

[6] S. Droste, *New Results on Visual Cryptography*, Advances in Cryptology-CRYPTO'96, Lecture Notes in Computer Science, vol. 1109, 1996, pp. 401-415.

[7] R. Hwang and C. Chang, *Some Secret Sharing Schemes and their Applications*, Ph. D. dissertation of the Department of Computer Science and Information Engineering, National Chung Cheng University, Chiayi, Taiwan, 1998.

[8] Moni Naor and Adi Shamir, *Visual Cryptography*, Advances in cryptology- EUROCRYPT94, Lecture Notes in Computer Science, vol. 950, pp. 1-12, 1995.

[9] E. Verheul and H. V. Tilborg *Constructions and Propertics of k out of n Visual Secret Sharing Schemes*, Designs, Codes and Cryptography, vol. 11 no.2, 1997, pp. 179-196.

Accessing Files from Public Computers in a Trusted Manner

Salih K A, Abhay Khoje and Rajat Moona
Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
Email: kasali, akhoje, moona@cse.iitk.ac.in

***Abstract*—Governments, military, different private organizations, financial institutions and hospitals have great deal of confidential information which is stored on computers. All these organizations want this information to be secure and accessible from any part of the world. For data security, one can use any of the encrypting file systems like eCryptfs [1], dmCrypt [2]. However these encrypting file systems do not address the problem of accessing files over network from public computers. In this case the public host, the actual FileServer host and the network between them are vulnerable to many attacks. This paper discusses the major problems and proposes a solution for the same using strong cryptographic methods. It also describes how the proposed solution can be implemented on a Linux-based environment.**

***Index Terms*—FileServer (FS), WorkStation (WS), Encrypted File System (EFS), TransCrypt, SmartCard (SC), Distributed File System (DFS).**

I. Introduction

In today's world, data storage has become very common and affordable. Large organizations and institutions generally have centralized storage devices. As a consequence of these, securing confidential data against the data thefts which eventually impose risks of losing important personnel and organization data, is of utmost importance.

An encrypting file system provides the best solution to the above problem. There are many encrypting file systems which provides security by encrypting and decrypting the users data transparent to the user. These encrypting file systems address the problem of data security in different ways, such as per-file encryption, flexible key sharing and inclusion of superuser in the trust model. These encrypting file systems do not address the security issues when the data is accessed over network from a WorkStation (WS). In this paper we discuss the problems that arise when secure data is accessed from public computers through any distributed file system and also propose a solution for the same.

We assume the model shown in Figure 1, in which users access file system volumes hosted on FileServer (FS) from any WS through any distributed file system.

In this scenario, WS, FS and the network channel between them are vulnerable to many attacks. Among them the more severe ones are man-in-the-middle attack, client spoofing [2], and user masquerading attack. In this paper, we design a protocol which makes use of known cryptographic methods like mutual authentication, process credentials, session establishment and so on, to counter the various attacks.

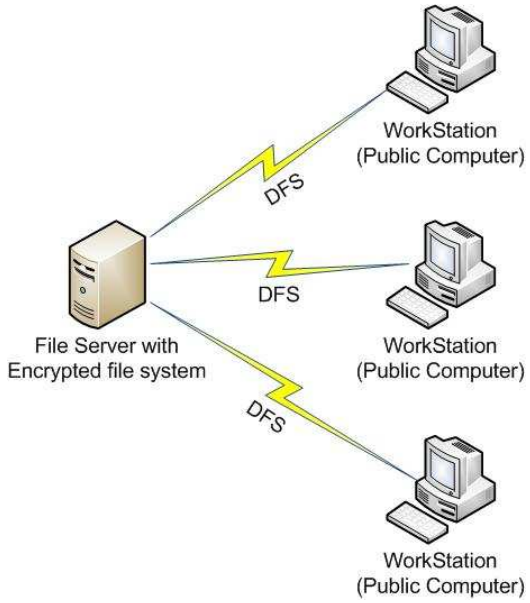


Figure 1. Model of file access in Public Networks

II. Previous Work

The NFS[3] based network provides transparent file sharing between heterogeneous systems. It had placed little emphasis on security concerns. The later versions of NFS tried to add more levels of security to the initial version. But most of them rely on the parameters which the administrators use to configure the system. NFS version 4 [4] specifies a number of sophisticated security mechanisms and mandates their implementation by all conforming clients. These mechanisms include Kerberos 5, SPKM3, IPsec in addition to traditional AUTH_SYS security.

While IPsec is useful for securing NFS, because its security is host based, it does not protect the network from the attacker who logs on to IPsec-protected hosts and assume the identity of other users. The Kerberos V5 [5] system solves the above problem using user credentials. But it needs a Kerberos Distribution Center (KDC), which must be kept safe from attack to preserve the integrity of the system. Also it is very hard to assume such a setup in a public environment.

Using open source tools, such as Secure Shell (ssh) [6] one can create secure connections between FileServer and

WorkStations [7]. But it provides security only between the WorkStations and the FileServer host. So it is vulnerable to attacks on these hosts.

To prevent the attack on FileServer, encrypted file systems such as the Encrypting File System [8], File Vault [9], dmCrypt [2], eCryptfs [1], TransCrypt [10] can be used. EFS implements encryption at the kernel level, but all key-management operations are done at user space. This hybrid feature makes it vulnerable to user-space attacks. File Vault creates encrypted sparse images to store user's home directories and does virtual memory encryption. dmCrypt is a volume based encrypting file system that is part of the standard Linux kernel. But it lacks flexibility due to the use of a common mount-wide key. Also it only addresses a narrow threat model. eCryptfs is a kernel-native cryptographic file system for Linux. It is a stacked file system that encrypts and decrypts the files as they are written to or read from the lower file system. Its hybrid design employing a user-space key management daemon exposes the system to user-space attacks and fails to exclude the superuser account from the trust model. TransCrypt is a secure, usable, transparent, efficient enterprise-class encrypting file system for Linux. It excludes the super user from the trust model. It uses per-file per-user file keys for encryption and decryption. But the present version of TransCrypt is vulnerable to user masquerading attack by any privileged user in a networked environment.

Even though these methods address attacks on FileServer hosts, they do not tackle the attacks arising from the network and WorkStation hosts, which are more vulnerable in general.

III. Issues/Problems to consider

As discussed earlier there are many issues to consider while accessing files using public computers (WS). We discuss the major ones in detail.

A. Masquerading attack

A masquerade takes place when one entity pretends to be a different entity. Authentication sequences can be captured and replayed after a valid authentication sequence has taken place, thus enabling an unauthorized entity to obtain extra privileges by impersonating an entity that has those privileges.

Let us consider that user *usr1* has access to some files on machine *cl1*. Suppose a malicious user *usr2* who wants to access *usr1*'s file, can impersonate as *usr1* if he has super-user privileges on any machine where *usr1* has account. Now, FS cannot distinguish the file operation request from *usr1* and *usr2*, and it will give access to both the users. Figure 2 shows such an attack over network.

B. Man-in-the-middle attack

In this attack, an attacker actively monitors the traffic between two entities and also captures and controls the communication transparently. For example, an attacker can re-route a data exchange. When computers are communicating at low levels of the network layer, the computers might not be able to determine with whom they are exchanging data.

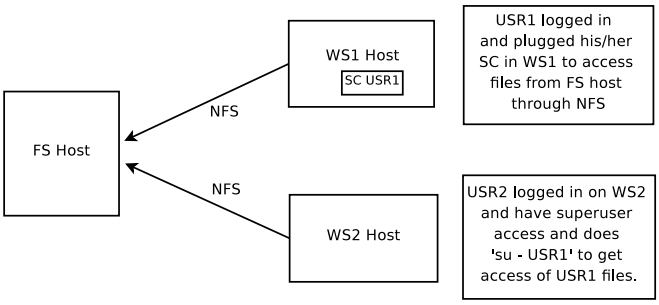


Figure 2. Masquerading attack over Network

Attacker makes independent connections with the two entities and relays messages between them. The attacker makes them to believe that they are talking directly to each other over a private connection when in fact the entire conversation is controlled by the attacker.

Suppose user *usr1* on machine *cl1* wants to access his files from server *cl2* over the network. Meanwhile, user *usr2* on machine *cl3* wishes to eavesdrop on the conversation. To get started, *usr1* sends the file identifier and his user ID to *cl2* over the network. But *usr2* is able to intercept it, a man-in-the-middle attack can begin. *usr2* sends a forged message to *cl2* that claims to be from *usr1* at *cl3*. *cl2*, believing this message from *usr1* at *cl3*, sends the response to *cl3*. Now *usr2* depending on his desire, can send altered data to *cl1*. When *usr1* receives the message, it believes it came from *cl1*. In this case a malicious user gets access to a genuine user's file and is even able to alter his data.

C. Replay Attack

Message replay involves the re-use of captured data at a later time than originally intended in order to repeat some action of benefit to the attacker. In our scenario, the attacker can capture the packet containing file handle information for DFS operations of a genuine user and can replay it later to get access to the files.

D. Offline Attack

Files stored on FS are vulnerable to offline attack, in which the attacker can get hold of the physical device where the files are residing and can access the data which are stored in plain text.

IV. Design

This section gives a comprehensive description of our protocol, designed using known cryptographic methods. It further discusses the effectiveness of this protocol to protect against the earlier mentioned attacks. An EFS can protect data from *offline attack*, hence FS should use an EFS to keep the data secure.

A. Protocol

We assume the following trust model in developing the protocol.

- FS kernel is trusted not to leak file encryption keys and file system keys used in EFS.
- Super user is partially trusted not to substitute the kernel image with a malicious version over a system reboot.
- WS login program is trusted to assign credentials to correct login session only and not to malicious user's login session.
- WS kernel is trusted not to assign credentials to processes created by malicious user and also trusted to send correct credentials assigned during login when the user is accessing EFS volume.

The various steps in the protocol are described briefly below.

1) Authenticate and establish a session between WS kernel and FS kernel: In the mutual authentication, user will be authenticated using his private key store (PKS) which can be any device for example SmartCard, Mobile Phone, USB device, which supports public key infrastructure (PKI). We assume that SmartCard is the PKS in our environment, since it is the most secure tool to carry the secret keys.

There should be a global identifier which is common between WS and FS host. This global identifier will be used by FS to identify which user has logged in on WS host. FS can not use user ID as a global identifier because user ID might be different on FS and WS host. However, common name of user's certificate is same everywhere and can be used as global identifier between WS and FS host.

WS login program waits for user to put his SmartCard (SC). When users puts his SC, login program initiates the authentication process by sending GET_CERT request to SmartCard of the user. SmartCard in response to this command sends user's certificate to login program. Login program extracts common name from the certificate and sends a request containing certificate name, user ID and SmartCard Uniform Resource Identifier *(CERT_NAME, UID, SCURI)* to the FS kernel for registering user's SmartCard. FS kernel authenticates the user who is logged in on WS host (corresponding to UID) by sending a challenge (r_fs) and part of session key (k_fs) to user's SmartCard through WS. SmartCard decrypts the challenge and then sends the response of r_fs, another challenge (r_sc) (to authenticate the FS kernel) and the other part of session key (k_sc) back to FS. FS decrypts the request from SC, verifies the challenge and calculates the session key for future use. It also prepares the response to the challenge r_sc sent by the SC.

Figure 3 explains in detail how authentication protocol works. FS generates credentials (SID) and passes this, along with the response to r_sc, to already authenticated SC. SC verifies the response r_sc. Before passing the credentials to WS host from SmartCard, SmartCard authenticates WS by sending a challenge encrypted with WS host's public key and verifies the response that came from WS. After authenticating WS, SC passes these credentials to WS login program encrypted with WS host's public key.

FS also maintains this information (credentials, UID, SCURI) in a table shown in Figure 4 so that when any file request comes for the EFS volume from WS kernel, it can

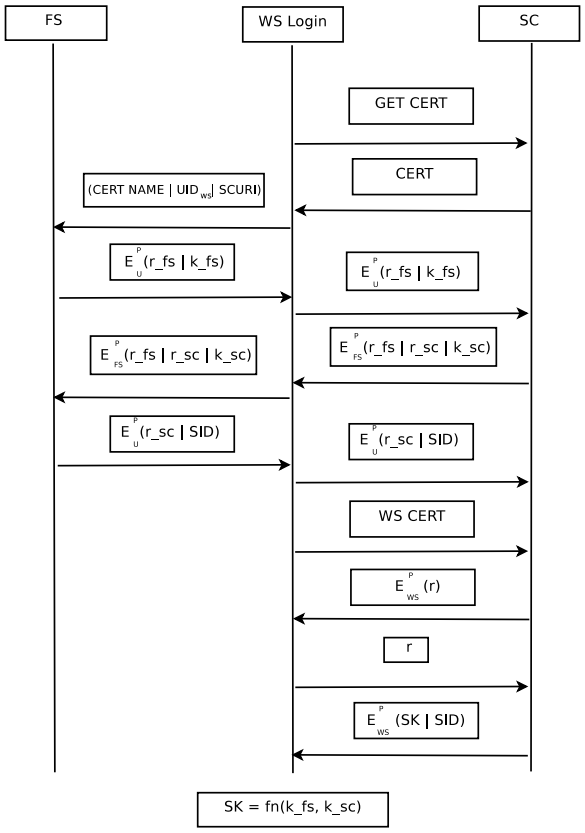


Figure 3. Authentication Protocol

check the incoming credentials and SCURI with the ones stored in its table. And only if this information matches, it will further process the file request. Otherwise it will return an error since this request is not from a genuine user session. FS uses SCURI to send the request to SmartCard during processing the file request.

Figure 4. FS Table

2) Assign credentials to WS kernel: Login process needs to pass the credentials (SID, which it got from user's SC) to WS kernel so that WS kernel can pass this with every file operation to FS kernel. By assigning the credentials to login process, processes which are descendants (children) of login process will inherit these credentials and will able to get access to EFS volume.

B. Distributed File System Operations

We now look at how open, read and write request from DFS client to DFS server works in our protocol using the established *Credentials* and *Session Keys*. Suppose a user on WS wants to access EFS volume located on FS. Then user has to mount the corresponding EFS volume on WS host. User can

use any DFS to do so. In the following discussion WS will act as DFS client and FS as DFS server.

1) Open: When user opens a file on WS host which is actually mounted from an EFS volume on FS host, DFS client sends the file handle and the credentials, both encrypted with the session key to the FS. After receiving this, FS decrypts the received data using the session key maintained in the table and also verifies the credentials and SCURI. If verification failed, FS will not process the request further and sends the *Permission Denied* error to WorkStation. Figure 5 shows how DFS open works in this protocol.

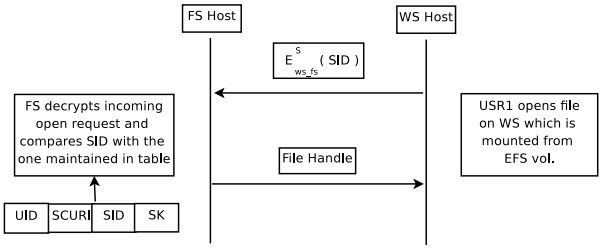


Figure 5. DFS open request

2) Read: All the parameters of this call are encrypted using the session key. FS decrypts the incoming parameters of the read call and verifies these with the one stored in the table. The data blocks, output of the read call, from FS can also be secured by using IPsec or the session key established between the WorkStation and FileServer. Figure 6 shows the steps in DFS read.

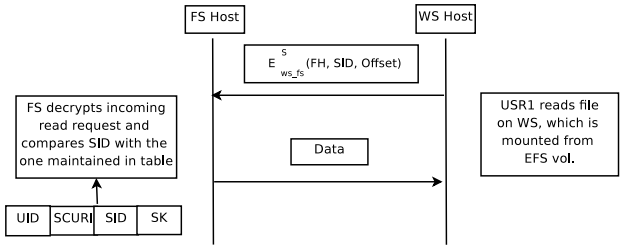


Figure 6. DFS read request

3) Write : All the parameters of this call are encrypted using the session key. By using IPsec or the session key even the data blocks to be written can be secured from network attacks. This is illustrated in Figure 7.

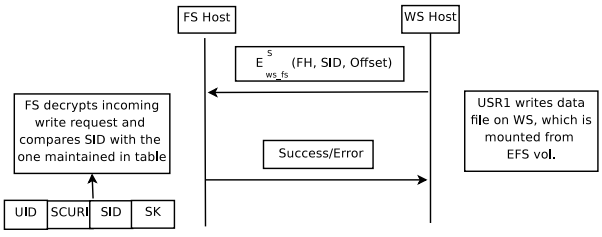


Figure 7. DFS write request

The designed protocol prevents the *Offline attack* on FS side by keeping the information on the physical device encrypted using an encrypted file system. *Man-in-*the-*middle* attacks are effectively handled by mutually authenticating user WorkStation and FileServer prior to any file operations. Along with the authentication, a session key is also established between the hosts so that the responses from FS will be encrypted with this key. Only the genuine WS can decrypt this response since it is the only one who has the session key. *User masquerading attack* is solved by assigning credentials to the genuine user's processes in WS by the FS during the user's login. In every file operation by the user from the WS host, these credentials are passed to the FS kernel. FS kernel verifies these incoming credentials with the stored ones to check whether this request is from a genuine user process.

V. Implementation

We now describes a scalable implementation of our design. The basic steps involved are described in brief in following subsections.

A. Choosing the platform for implementation

FileServer should use an encrypted file system to store the user information to prevent the attacks against the user data. We can use any enterprise encrypted file system like dmCrypt, eCryptfs or TransCrypt. The one which matches the most with our assumed trust model is TransCrypt. The major security hole in TransCrypt is the masquerading attack from any privileged user. However by integrating the user credentials approach (discussed in section IV) in every file operation, the attack can be prevented.

To access files on the FileServer from WorkStations, one can make use of any distributed file systems like NFS, CODA and so on. The NFS was developed to allow machines to mount a remote file system as if it were a local disk. It allows fast and secure sharing of files over the network. Samba provides file services to Windows clients. The Andrew file system provides a file sharing mechanism with some additional security and performance features. The CODA file system combines file sharing with a special consideration for disconnected clients. Many of the features of the Andrew and CODA file systems are slated for inclusion in the next version of NFS [4]. The advantage of NFS today is that it is mature, standard, well understood, and supported robustly across a variety of platforms. Hence we use NFS for remote file access.

B. Implementation using TransCrypt and NFS

Having defined the protocol and the platform, the implementation involves changes in the user WorkStation side and TransCrypt FileServer kernel. We will discuss which components from both sides need changes.

1) User WorkStation Side : Here we need to do changes in the user space as well as in kernel space. In userspace we need login program changes, new TransCrypt daemon and a SmartCard daemon. In kernel space, we need changes to support acquiring and setting user credentials and to secure NFS operations. Each component is described in brief below.

- *Login Patch.* This has to take care of the mutual authentication between FS and WS. This can be done by the (PAM) pluggable authentication module[11] when login program sets the session ID i.e. in the session management. This module have two methods. One will perform tasks associated with session set-up and other will perform tasks associated with session tear-down. The first method is used to communicate with User's SmartCard and FS kernel for mutual authentication and session key establishment. Apart from above, it needs to pass the per user session credential from FS to the WS Kernel. It will use an ioctl system call to pass this information to the kernel. Second method will be used to inform FS that user has logged out from WS. So that it can remove the corresponding state information.

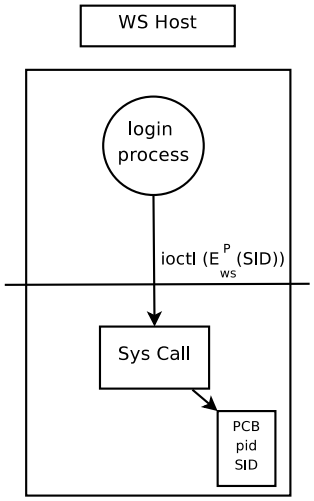


Figure 8. Credential Assignment

- *SmartCard Daemon*. It has to take care of the full duplex communication between Work Station, login process and user's SmartCard. So it needs SmartCard APIs to communicate to SC and socket support to communicate with other entities.
- *Work Station-TransCrypt daemon*. It has to communicate between Work Station kernel and File System kernel during any TransCrypt volume file operations from the user. It also needs to forward packets intended to user's SmartCard to the SmartCard daemon. So basically it integrates socket support for communication.
- *Device Layer in kernel*. When we call *ioctl [12]* system call from PAM module in the open session, WS kernel should accept these parameters passed by ioctl call and set these as process credentials of login process. These things can be done by registering miscellaneous device for routing ioctl system calls. This part of the kernel will be instantiated when any application calling ioctl for particular file handle. This will assign the credential to the 'task_struct' structure of the login process so that every subsequent children will inherit it. It will store the session keys communicated in a kernel data structure (in a table as discussed in previous section) so that every communication between WS kernel and other entities will be secured.
- *NFS Client Patch*. During any file operation from the WS side, NFS client has to pass the credentials (SID) corresponding to the user process to the NFS server (to prevent user masquerading attack). This along with other parameters will be encrypted with session key (established earlier b/w WS and FS) before passing to Server. We can use 'ipsec' between NFS client and NFS server for secure data transmission. And this work is achieved by patching the NFS client code in the Linux kernel.

2) FileServer Side: Here also we need changes in the user space as well as in kernel space. In userspace we need TransCrypt daemon changes to support initial authentication protocol. In kernel space we need changes to keep the state information corresponding to every authenticated user. Also need changes in the NFS server to support secure communications. Each of these changes are explained in brief below.

- *TransCrypt Daemon*. All the communication between FileServer and other entities (login process, WS-TransCrypt daemon) has to be routed through this. So it basically integrates socket support.
- *Netlink Communication Manager Change*. In the current version of TransCrypt, the communication manager subsystem supports only kernel initiated packets. It is not capable for handling User initiated requests like those in the Work Station authentication protocol. So a 'direction' field has to be added to the messages so that messages related to different protocols can be handled correctly.
- *User Authentication Subsystem*. This subsystem actually has to generate the messages that has to be generated as part of the initial authentication protocol and then will communicate to the Work Station through the communication manager. It also store the credentials, WorkStation details and the session keys established as part of the protocol already discussed, in a shared data structure.
- *NFS Server Changes*. When any NFS client tries to mount a TransCrypt enabled volume, we need to check whether the user WorkStation is already registered with the FileServer kernel. If not it has to reject the connection. If registered, during other/open() file operations it has to verify whether the credential passed are correct. Apart from that before granting access, it has to make sure that it got the user token decrypted back.

C. Usability and Performance

Atmost care has been taken for modularity for easy integration with current Linux Kernel and Login program. Linux Security Module (LSM) is a framework of kernel hooks that would allow many security models to work as loadable kernel modules. The kernel vfs layer changes are moved to LSM hooks so that one can easily load it, when required and can easily port TransCrypt to newer version kernels. By the use of

Pluggable Authentication Module, the login program can be easily patched in WS side to work with the solution.

A significant performance improvement for file creation and open operations, as well as all ACL manipulations in TransCrypt can be derived by caching user certificates in memory instead of sending a request to *transcryptd* every time a certificate is required. Caching feature is added to TransCrypt for the same and a user-interface is provided to invalidate the cache at any point in time using configfs [13].

VI. Conclusions

This paper discussed the security vulnerabilities, when a user tries to access his files over an untrusted network. We also proposed a secure and scalable protocol to solve the same. The strong trust model we assumed (even administrators are not trusted) distinguishes it from all the other existing solutions proposed so far. We have a scalable implementation of the solution in Linux environment using the open source encrypted file system, TransCrypt [10]. The modularized implementation helps anyone to easily integrate the solution to their environment. The source code and binaries can be obtained from [14].

VII. Acknowledgments

We would like to thank Prabhu Goel Research Center for supporting us and providing the facilities to carry out this research work. We would also like to thank Prof. Arnab Bhattacharya and Prof. Piyush Kurur for their help and suggestions through out our work. We also thank Satyam Sharma for guiding us in the right path.

References

[1] M. A. Halcrow, "eCryptfs: An Enterprise-class Encrypted Filesystem for Linux," in *Proceedings of the Linux Symposium*, Ottawa, Canada, Jul. 2005, pp. 201–218.

[2] "dm-crypt: a device-mapper crypto target for Linux," Website, http://www.saout.de/misc/dm-crypt/.

[3] S. D. R.Sandberg, D.Goddberg and B-Lyon, "Design and Implementation of the Sun Network File system ," in *Usenix Conference Proceedings*, June 1985.

[4] D. R. R. T. S. Shepler, B. Callaghan, "Network File System (NFS) version 4 Protocol," United States, 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3530.txt

[5] "Kerberos Resource Page," Website, http://web.mit.edu/Kerberos/.

[6] T.Ylonen, "The SSH (Secure Shell) Remote Login Protocol, SSH-1 Specification," 1995.

[7] "SSH Tunneling," Website, http://www.ccs.neu.edu/groups/howto/howtosshtunnel.html.

[8] "How Encrypting File System Works," Website, http://technet2.microsoft.com/WindowsServer/en/Library/997fdd99-73ec-4041-9cf4-1370739a59201033.mspx.

[9] "Apple Mac OS X FileVault," Website, http://www.apple.com/macosx/features/filevault/.

[10] S. Sharma, R. Moona, and D. Sanghi, "TransCrypt: A Secure and Transparent Encrypting File System for Enterprises," in *8th International Symposium on System and Information Security*, 2006.

[11] "Linux PAM: Pluggable Authentication Modules for Linux," Website, http://www.kernel.org/pub/linux/libs/pam/.

[12] "ioctl - control device," Manual Page, `ioctl(2)`.

[13] J. Becker, "[PATCH] configfs, a filesystem for userspace-driven kernel object configuration," Linux Kernel Mailing List, April 2005, http://lkml.org/lkml/2005/4/3/112.

[14] "TransCrypt Filesystem Homepage," Website, http://www2.cse.iitk.ac.in/transcrypt/.

[15] "Arms dealers got Navy plans and deployment details," Website, http://www.indianexpress.com/story/8028.html.

[16] "Symantec: Average Laptop Contents Are Worth Half A Million Quid," Website, http://www.digital-lifestyles.info/display_page.asp?section=cm&id=2960.

[17] P. s. C. S. D. L. D. H. Brean Pawloski, Chet Jaszczak, "NFS version 3 design and implementation," in *Usenix Conference Proceedings*, June 1994.

[18] P. S. B. Callaghan, B. Pawlowski, "NFS Version 3 Protocol Specification," United States, 1995. [Online]. Available: http://www.ietf.org/rfc/rfc1813.txt

[19] "EncFS: Virtual Encrypted Filesystem for Linux," Website, http://encfs.sourceforge.net/.

SCTP-Sec: A secure Transmission Control Protocol

Rahul Choudhari
Indian Institute of Information Technology & Management, Gwalior, INDIA
Email: *rahul.choudhari@iiitm.ac.in*

Somanath Tripathy
Indian Institute of Technology Patna, Bihar, INDIA
Email:*som@iitp.ac.in*

Abstract – *The Stream Control Transmission Protocol (SCTP) does not retain state information at the server side to avoid the traditional DoS(denial of service) attack in STCP. Unfortunately, SCTP is not secure against verification-tag guessing -attack which leads to association-hijacking and forces that victim clients to starve out of services. Therefore, we propose a secure SCTP mechanism called SCTP-Sec that includes Cookie mechanism as base to make the server a stateless, while it uses cryptographic hash operation to resist against the verification tag and hijacking attacks.*

Keywords*: Denial of service, SCTP, SYN flood, SYN cookies,*

I. INTRODUCTION

A communication network is susceptible to denial-of-service (DoS) attack, unless specific care has been taken. The primary objective of DoS attack is to degrade or interrupt the services offered by the network, or the end systems connected to it. Sometimes, DoS attack causes incorrect behavior and crashes the system to interrupt the services.

DoS attacks may be carried out on any layer of the network protocol stack. In the network layer for example, a simple attack could jam part of a network by flowing useless data packets (e.g., ICMP echo request "*ping*" messages in an IP network) to overload the router that results in delaying or dropping the worthy packets. On the transport layer, an attacker could send specific requests to exhaust resources at the target node. Obviously, the attack request should be hard or impossible to distinguish the forged request from legitimate requests. Most often, attackers exploit the architecture of a transport layer protocol like three-way handshake in TCP is vulnerable to DoS attacks [4]. **SYN flood** is a powerful denial-of-service attack against TCP in which, an attacker sends a succession of SYN requests to a target's system. It works if a server allocates resources after receiving a SYN and before receiving the ACK.

SYN flooding attack takes advantage of the state retention of TCP after receiving a SYN segment to a port that has been put into the LISTEN state. The basic idea of SYN-flood attack is to exploit the state retention behavior causing the host to retain enough states for bogus half-connections. Thus all the resources get consumed by the bogus half connections and therefore, no resource left to establish a new legitimate connection. There are two methods to involve the server to stay without ACK. A simple method could be that a malicious client can skip sending the last ACK message. Alternatively, attacker spoofs the source IP address in the SYN making the server to send the SYN-ACK to the falsified IP address, so that the server never receives the ACK. In both the cases server waits for the acknowledgement.

The cookie mechanism was proposed back in 1996 by D. J. Bernstein [2] to prevent the SYN flood attack. The SYN Cookies allow a server to avoid dropping connections even if the SYN queue fills up. The major advantage of use of SYN Cookie mechanism is that it does not need to break any protocol. On the contrary, the major limitation would be that the server is limited to only 8 unique maximum segment size (MSS) value. Another drawback is that the server must reject all TCP options (such as large windows), because the server discards the SYN queue entry where that information would otherwise be stored [9].

Though the cookie mechanism prevents SYN-flood attack, it is not resistant against blind connection forgery [2]. If an attacker guesses a valid sequence number sent to someone else's host then the attacker could be able to forge a connection from that host. SCTP [8] has been designed from scratch keeping the cookie mechanism in mind. SCTP is a 4-way handshake mechanism without the state retention at the server side (till the client's acknowledgement arrives) and therefore, avoids the traditional security flaws like TCP half open connections. Still it is observed that the attacker can prevent communication between honest endpoints by blind connection forgery, stealing/camping their addresses, hijacking an association being on the route between the endpoints, trick a server to flood a target address with data and creating denial of service to the client in an unexpected way [1]. In this paper, we discuss the attacks against SCTP that leads to deny a legitimate client from obtaining services. Further, we propose a mechanism called SCTP-Sec to resist against such attacks

The remainder of this paper is organized as follows. SCTP protocol and its vulnerabilities are discussed respectively in Section II and Section III. The proposed mechanism called SCTP-Sec is discussed in Section IV. Section V contains the security analysis of SCTP-Sec. The effectiveness of SCTP-sec is discussed in Section VI and the work is concluded in Section VII.

II. OVERVIEW OF SCTP

The "Stream Control Transmission Protocol" (SCTP) [8] is a relatively new transport layer protocol for the IP protocol stack. It has been developed by the IETF (Internet

Engineering Task Force) SIGTRAN (SIGnalling TRANsport) working group [5] to be the base of architecture for the transport of SS7 signalling data over IP networks. The "Signalling System No. 7" (SS7) is a packet-oriented network used for controlling the operation of the (connection-oriented) Public Switched Telephone Network (PSTN). Despite this objective, SCTP is designed as a generic transport layer protocol such as TCP and UDP. This general approach is emphasized by the fact that further development of SCTP has been turned over to the IETF TSVWG (Transport Area Working Group) in the meantime.

A. Protocol Basics

SCTP association is a relationship between two SCTP endpoints. An end point is a set of transport addresses and a transport address consists of a network- layer address and a port number. In SCTP, all transport addresses of an endpoint must share the same port number. Thus in practice, an SCTP endpoint is identified with a non-empty set of IP addresses and a single port number. Each transport address can belong to only one endpoint at a time. This means that no special endpoint identifiers are needed. The receiver of an SCTP packet identifies the source and destination endpoints and the association to which the packet belongs based on the source and destination IP addresses and port numbers.

An SCTP packet comprises of a common header and zero or more chunks. The chunks may carry either SCTP signaling information or user data (DATA chunk). Multiple chunks, such as data and acknowledgements, may be bundled into one packet. SCTP provides ordered and reliable multi-stream transport. Figure 1 shows the messages sent during a typical SCTP association setup, which is a 4-way handshake. The messages in the figure are identified by the names of the signaling chunks that they contain. Although not shown in the figure, user data can be bundled into the third and fourth messages of the handshake to save one round trip.

B. SCTP 4-way handshake with cryptographic cookies

As depicted in figure 1, SCTP uses a 4-way handshake with a cryptographic cookie for an association establishment. The approach uses similar to TCP SYN cookies. However unlike TCP (where the cookie mechanism has been added later without changing the original protocol specification), the SCTP protocol has been designed and specified to use the cookie mechanism from start. SCTP uses a type indicator field in its message ("chunks" in SCTP's terminology), which makes it possible to distinguish all four messages those are used for the handshake (INIT, INIT ACK, COOKIE ECHO and COOKIE ACK) and simplifies the packet-filtering.

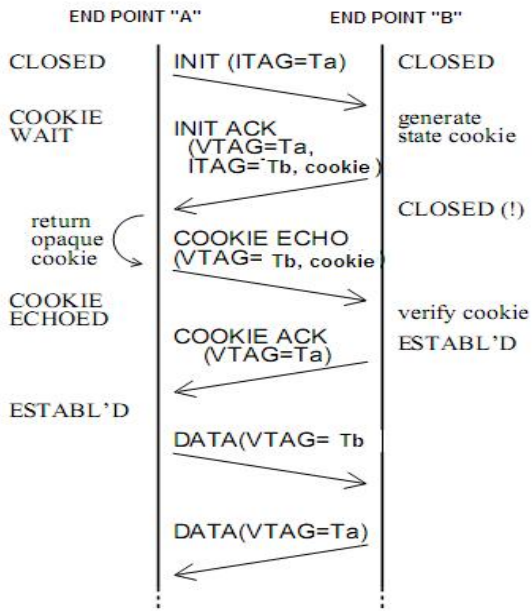


Fig.1.SCTP Handshake

For faster association setup, the first DATA chunks may be bundled with the COOKIE ECHO and COOKIE ACK chunks. An important feature of SCTP handshake is that the respondent (endpoint B) remains stateless between sending the second-message and receiving the third-message. The respondent encodes the protocol state including the contents of the INIT into a state cookie to be sent to the initiator (endpoint A) in the INIT-ACK. The initiator returns the cookie to the respondent in the COOKIE ECHO. This prevents state-exhaustion attacks similar to the TCP SYN flooding [4].

The SCTP specification provides own data field for storing the state cookie in the respective messages. The only size limit for the cookie is that the cookie together with all the other parameters has to fit into the INIT ACK or COOKIE ECHO message, respectively, which must not exceed 64 KB each. Therefore, all state information and all options may be stored in the cookie including the auxiliary parameters such as timers. Note that the SCTP standard does not specify the data format of the cookie nor the cryptographic algorithm to be used for signing the cookie. Instead, these are "implementation specific" and issues are left to the implementer's choice.

C. Verification tags

In the first two messages of the handshake, the endpoints exchange random nonce, a 32-bit sequence number used internally by SCTP. Receiver of this packet uses the Verification Tag to validate the sender of the SCTP packet. Value of the Verification Tag (VTAG) must be set to the value of the Initiate Tag (ITAG) received from the peer endpoint during the association initialization.

Initiate Tag values should be selected randomly from the range of 1 to $(2)^{32}$- 1 to protect against "man in the middle" and "sequence number" attacks. Careful selection of Initiate Tags is also necessary to prevent duplicate packets from previous associations being mistakenly processed as belong to the current association. Moreover, the Verification Tag value used by either of the endpoints in a given association must not change during the life time of an association. A new Verification Tag value must be used on each time the endpoint tears down and then re-establishes an association to the same peer. This tag serves the same security purpose as the randomly initialized sequence numbers in TCP. Obviously, the verification tags are not a very strong security mechanism. Any node that sees packets belonging to the association learns the verification tag values and can consequently spoof packets for that association.

D. Chunk Fields

SCTP packets have a simpler basic structure than that of TCP or UDP packets. As depicted in Figure 2, each packet consists of two basic sections: the *common header* which occupies the first 12 bytes and the *data chunks* which occupy the remaining portion of the packet. Each chunk has a type identifier that is one byte long yielding at most 255 different chunk types. The list of chunk-types is defined in [7]. The remainder of a chunk is a two byte length (maximum size of 65,535 bytes). Chunk Flag depends on the Chunk type as given by the Chunk Type field. Unless specified, these are set to 0 on transmit and are ignored on receipt. Chunk Length represents the size of the chunk in bytes, including the Chunk Type, Chunk Flags, Chunk Length, and Chunk Value fields. Therefore, if the Chunk Value field is zero-length, the Length field is 4. The Chunk Length field does not count any chunk padding. If the chunk does not form a multiple of 4 bytes (i.e., the length is not a multiple of 4) then it is implicitly padded with zeros which are not included in the chunk length. The receiver must ignore the padding. A robust implementation should accept the chunk whether or not the final padding has been included in the Chunk Length. The Chunk Value field contains the actual information to be transferred in the chunk. The usage and format of this field is dependent on the Chunk Type.

0	Common Header		
96	Chunk1	Chunk1 Flags	Chunk1
128	Chunk 1 Value		
…	…….		
…	Chunk N Type	Chunk N	Chunk N
…	Chunk N Value		

Fig.2. SCTP chunk fields

III. VULNERABILITIES OF SCTP

This section discusses the vulnerability issue of SCTP arises due to the weakness of the protocol. The attacker can disrupt communication between honest endpoints by stealing their addresses, blind connection forgery, hijack the cookie, tricks a server to flood a target address with data, and forward associations in an unexpected way.

A. Address camping and stealing

This attack exploits the SCTP's multi-homing to craft a form of denial of service attack. In effect, an illegitimate client connects to a server and steals or "holds up" a valid peer's address to prevent the legitimate peer from communicating with the server.

B. Blind connection forgery

This attack can be executed in various forms. If an attacker guesses a valid verification tag which was sent to someone else's host, he can forge a connection from that host. The attacker can try to cryptanalyze the server-selected secret function: inspect a series of valid cookies and then intelligently guess a new cookie.

C. Cookie hijacking attacks

In this attack, the attacker sends an INIT to the server using same IP address and port number as in the existing association. The server automatically responds with an INIT-ACK containing a state cookie. Inside the state cookie, attacker can find both the verification tags. Moreover, the locations of each tag in the cookie depend on the implementation which is not hard to guess. The worst part is that the attacker usually does not need to know the correct port number, the peer's address, even that the association exists. This is because the arrival of any packet belonging to the association alerts the new address owner to the opportunity for misuse. A single packet even provides all the necessary information for the above attacks.

D. Bombing Attack (Amplification)

Here, an attacker sends packets using the victim's address as the source address containing an INIT chunk to the SCTP Server. The server then sends a packet containing an INIT-ACK chunk to the victim, which is most likely larger than the packet containing the INIT.

IV. THE PROPOSED MECHANISM SCTP-SEC

In this section, we describe our proposed mechanism called SCTP-Sec for tackling, tag spoofing and hijacking in SCTP. This mechanism prevents DoS (denial of service) against client as well as server. SCTP-Sec uses hashing of verification tags used at the client end to prevent the attacks that exploits the sequence number in SCTP four way handshake. The mechanism also provides authentication of the clients. We will outline the basic operation of this system, emphasizing some parameters that affect its performance.

A. Assumptions

SCTP-Sec uses an extended SCTP chunk in the header. It is clearly defined in [7] that only 15 different chunk types are used currently while rest other Chunk Types are reserved for future use by IETF. We define a new chunk type (may be one of the reserved chunk types), and refer this extra chunk as "pseudoINIT" chunk. This chunk contains the value of initial Verification Tag of the client (end point A) as its chunk value. The Chunk Type is set to the extended chunk-id value. The flag value is set as 1 on transmission of the chunk, so that the chunk is processed on receipt. This extended chunk is sent along with the cookie echo process. After processing the pseudoINIT chunk value it is hashed at the server (end point B) and compared with the initial hashed value of the INIT of the client.

B. The SCTP-Sec operation

The model for our system goes a step further from a normal SCTP connection establishment. It allocates state only after the server receives the pseudoINIT Chunk Value (**Ta**) along with the cookie echo. The detailed description is given in the Figure 3.

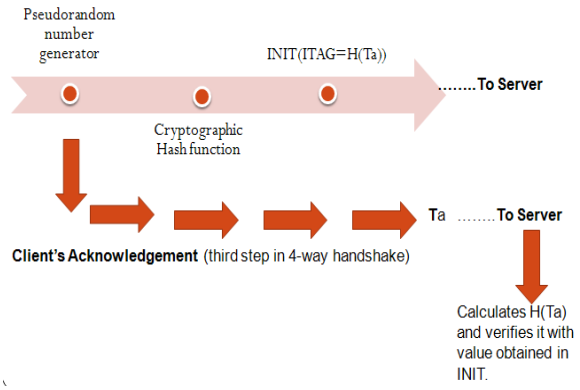


Fig.3. Client's role during connection establishment

The client generates a pseudorandom number (**Ta**). This value is hashed by using any secure hash function (agreed between the client and server) such that the hashed value will contain 32 bits. This 32 bit value is sent along INIT as the initial Verification Tag of client for the four-way handshake during connection establishment. The client during its acknowledgement sends the pseudorandom number (Ta generated in the first step) in the pseudo-INIT Chunk. The server performs hash operation on the value (Ta) in pseudo-INIT Chunk and compares it with the value obtained during INIT of the handshake. The four-way handshake for SCTP-Sec is explained in Figure 4.

In order to initiate a connection, the client (end point A) generates a random number so as to make harder for an attacker to guess the Tag. Then the system performs a cryptographic hash function on the chosen random number to get a hashed value, ie, H(**Ta**) where, H is a cryptographic one way hash function. This H(**Ta**) is sent to the server as INIT the initial Verification Tag. The server sents a cleartext cookie containing all the parameters involved in a normal SCTP cookie, and an ACK corresponding to client's INIT. The client sends the cookie echo, verification tag of server (**Tb**) along with the pseudorandom number in the pseudoINIT Chunk Value (**Ta**). The value (**Ta**) is received by the server through the pseudoINIT Chunk depicted in Figure 4. The server first computes H(**Ta**) on Ta and compares it with the hash value received during INIT; if both match then verify the cookie, otherwise drop the connection. In SCTP-Sec, the cookie mechanism remains the same for verification.

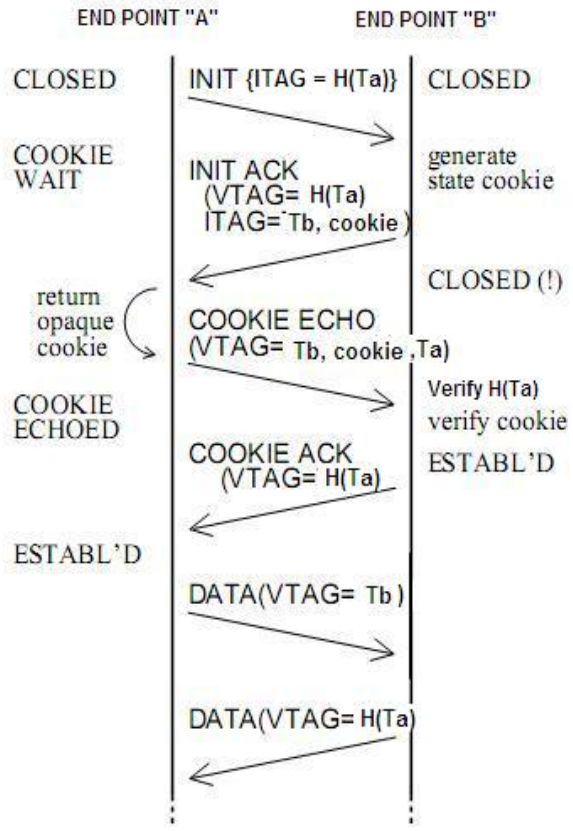


Fig.4. SCTP-Sec 4-way Handshake

V. SECURITY ANALYSIS OF SCTP-SEC

SCTP-Sec makes the spoofing of address infeasible. It is because the mechanism is more client-oriented and the system assures that the only legitimate clients can request to the server. Legitimacy of the client is verified through a cryptographic one-way hash function. The Verification Tags are reasonable tradeoffs between security and cost. However, it is possible to improve the robustness of the acknowledgements without resorting to expensive cryptography. In SCTP-Sec even if the attacker somehow obtains the Verification Tag **H(Ta)**, then it is not feasible for the attacker to guess the original pseudorandom number generated. This is owing to the pre-image resistance property of the cryptographic hash function [5].

As client has generated **Ta** and sent H(Ta) keeping Ta part secret, only he can authenticate itself by sending that **Ta** to the server during the third step of the handshake. This authentication acts as a secure end-to-end mechanism between the client and the server. This prevents, *guessing of verification tags, address spoofing and blind connection forgery* attacks.

SCTP-Sec does not affect the cookie mechanism and provides helping hand in authentication of client. It is difficult to exploit SCTP-Sec even if an attacker guesses the port number of a legitimate source.

A. Address camping or stealing

The proposed system forces the host to send back its authentication during the client's acknowledgement. The client needs to verify itself to the server by transmitting the pseudorandom number (**Ta**) generated for the INIT process. In this way, the attacker can't spoof address of the legitimate client.

B. Blind connection forgery

Attacker succeeds to guess a verification tag requires 2^{31} brute-force attempts. If we assume that the attacker has exponentially high computational power than client, attacker can inspect a series of valid cookies and then intelligently guess a new cookie. But this works only if the attacker has the pseudorandom number. If the attacker failed to guess the pseudorandom number (**Ta**) then the connection will be rejected before the verification of cookie. But, the attacker could be authenticated only when it has both (**Ta**) and H(**Ta**), which is computationally infeasible.

C. Cookie hijacking/replay attacks

The cookie hijacking is possible only if the attacker finds both the Verification Tags. The Verification Tags can be found in INIT-ACK that contains a state cookie. The location of the tags in the cookie depends on the implementation but it is not hard to guess. Since the attacker does not need to know the correct port number, the peer's address, even if the association exists. A single packet also provides all the necessary information. To resist this issue we have used a 32 bit hash digest as INIT. If the attacker guesses this hashed value then it is of no use till the attacker knows the secret pseudorandom number, meanwhile guessing this is computationally infeasible because of the one-way property of hash function. Thus it works like a secure key mechanism between server and client.

D. Bombing attack

Bombing attack uses an SCTP server to send a larger packet to a victim, than it sends to the SCTP server. The attacker sends packets using the victim's address as the source address containing an INIT chunk to an SCTP Server. In SCTP-Sec, double checking of client before establishing a connection assures the client is legitimate one if the connection is established and therefore, bombing attack is avoided.

VI. DISCUSSION

SCTP-Sec is resistant against SYN floods, guessing and blind connection forgery attacks as described in earlier section (summarized in Table 1). This is due the proposed mechanism SCTP-Sec to double check the authentication of the client. There is no computational delay from the client side in SCTP-Sec, as the hash value of pseudorandom number can be pre-computed at the client's end. The client only requires transmitting an extra chunk for sending **Ta** but this does not provide any significant effect on the 4-way handshake.

In SCTP-Sec, we can construct the encryption on cookie as client is sending its own verification value in the form of **Ta**. The cookie can be transferred in clear-text rather than encrypted. This will save the processing time. The server has to perform an extra hash computation for verification of the **Ta**. Thus there is a performance gain in SCTP –Sec over SCTP, since computing hash value is less computational intensive then an encryption/ decryption operation.

Table 1

s. no.	Parameters	SCTP-Sec	SCTP
1.	Security with Cleartext cookie	Yes	No
2.	Address Spoofing/ Camping	Resistant	Not Resistant
3.	Cookie-replay attacks	Resistant	Not Resistant
4.	Bombing attacks	Resistant	Not Resistant
5.	Blind connection fogery	Resistant	Not Resistant

VII. CONCLUSION

This paper discussed the vulnerability issues of SCTP against verification-tag guessing-attack which leads to relinquish the legitimate client from obtaining requested services. Also, we have proposed a secure transport protocol SCTP-Sec to avoid the similar attacks. In SCTP-Sec the client machine performs a pre-computation (hash-operation) before sending the Verification Tags during connection initiation to make the spoofing and guessing attacks more difficult. Further, the similar approach can be used in TCP SYN cookie mechanism to make it secure against sequence guessing attacks and hijacking attacks.

REFERENCES

[l] T. Aura, P. Nikander, G. Camarillo: *Effects of Mobility and Multi-homing on Transport-Protocol Security*, Proceedings 2004 IEEE Symposium Security and Privacy, 2004.

[2] D. Bernstein, "SYN cookies", visited in December 2005, <http://cr.yp.to/syncookies.html>.

[3] S. Bellovin, *Defending Against Sequence Number Attacks*, *RFC* 1948, May 1996.

[4] L. Christoph Schuba, et.al.: *Analysis of a denial of service attack on TCP*. In Proc. 1997 IEEE Symposium on Security and Privacy, pages 208-223, Oakland, CA USA, May 1997. IEEE Computer Society Press.

[5] D.R. Stinson, cryptography theory and practice, second edition, CRC Press, 2002.

[6] R. Fox: *TCP Big Window and Nak Options*. RFC 1106, June 1989.

[7] R. Stewart, Ed.: Stream Control Transmission Protocol. RFC 4960, September 2007.

[8] Stewart, et al.: *Stream Control Transmission Protocol.* RFC 2960, October 2000.

[9] M. Wesley, *TCP SYN Flooding Attacks and Common Mitigations*, *RFC 2960*, IETF, August 2007.

[10] Information Sciences Institute, university of Southern California, *Transmission Control Protocol*, RFC 793 September 1981.

Technical Session III

Tools and State-of-the-Art

A Hardware Authentication Architecture for Pervasive Devices

Abhishek Chanda
Microsoft India Pvt Ltd
Hyderabad, India
achanda@microsoft.com

Abhik Mukherjee
Bengal Engineering and Science University, Shibpur
Howrah, India
abhik@cs.becs.ac.in

Abstract

We present a hardware implementation for the HB++ authentication protocol. The design is efficient enough to be used in resource constrained pervasive devices like RFID tags or sensor motes. The architecture has been developed targeting the RFID environment, it has 8 bit data path along with a control unit and has been implemented on Altera's FPGA and CPLD platforms (Cyclone and MAX II) using the Quartus software. A detailed analysis of the implementation in terms of resource usage is also discussed.

1 Introduction

This paper presents the design and analysis of a hardware implementation of the HB++ authentication protocol for resource constrained pervasive environments. We have focused on the RFID environment as a potential candidate for the deployment of an authentication mechanism based on the HB++ protocol. This implementation can be easily ported to other similar environments like sensor network, wireless network etc. where authentication is necessary.

The paper is organized as follows: First we present an overview of the RFID environment and related security concerns, in Section 2 we describe the HB++ protocol. Section 3 describes our proposed architecture and the control units, Section 4 presents some experimental results related to the implementation. Section 5 and 6 presents some issues related to the implementation and some discussions about the proposed system. Section 7 concludes the paper.

1.1 Introduction to RFID

Radio Frequency Identification (RFID) technology is a promising alternative to manual identification systems. A number of unique challenges of this environment has attracted recent academic research in this area. A typical RFID system has three components:

- A tag, that bears an unique identification number. When interrogated by the reader, the tag will transmit that ID number to the reader.
- A reader, that interrogates tags in its range for their ID number. When the reader gets a response, it will relay the data to a backend server.
- A backend server, that maintains a database of valid tag ID against other parameters. When it receives data from the reader, it searches its database for the tag ID.

The tags must be extremely small and cheap since they are to be used in a large quantity. This means that it is not quite feasible to have a processor or even a microcontroller on the tag. Everything on the tag must be hardwired. This introduces serious resource constraints on the tag side in terms of processing power and available amount of energy.

The reader side,however, is not so constrained and is generally implemented on small microcontrollers.

1.2 Security issues in RFID

Like many other pervasive applications, one major security issue in RFID is authentication, where a tag and a reader must authenticate each other before transmitting potentially sensitive information. If a malicious reader interrogates a valid tag and the tag leaks out its ID, there may be a serious security breach. Similarly, the tag can be malicious also, where it supplies wrong information to a valid reader. This issue becomes more alarming due to the pervasiveness of these devices, it is easy to place a counterfeited tag in a region where only valid tags are to be present .

The resource constraints in RFID tags makes the implementation of common authentication protocol infeasible in them. Recently, a number of low cost protocols have been proposed that are more suitable for such pervasive applications. A number of low cost protocols have been proposed for the tag-reader authentication problem (in 2003 [1, 2], in 2004 [3], in 2005 [4, 5, 6] etc). Notably, at Crypto'05, HB+, a lightweight cryptographic authentication scheme

very well suited for low-cost hardware implementation, was introduced by Juels and Weis [6]. It provides a symmetric-key protocol allowing tags to identify themselves to the reader. HB+ is presented as an improvement of the HB protocol, which had been introduced in [7]. The security of the HB protocol does not rely on classical symmetric key cryptography solutions, but rather on the hardness of the computational Learning Parity with Noise (LPN) problem [8]. While the HB protocol is made to be secure against passive attacks only, the aim of HB+ is to be resistant to some active attacks. A proof of security is provided but at the same time, Gilbert, Robshaw and Sibert [5] describe a man-in-the-middle attack on HB+ not covered by the corresponding security model.

We focus on the HB++ protocol introduced in [9]. Our contribution is a hardware implementation of the protocol. This design can be used to implement an authentication layer in resource constrained pervasive applications like RFID. The design maintains the lightweightedness of the protocol to make it suitable for pervasive environments.

2 Overview of the HB++ protocol

We briefly describe and evaluate HB++ in this section. After providing a brief introduction to the protocol, we consider some security violations that may occur. The notations used are:

- a, b: random k-bit binary vectors.
- x, $\acute{x}$, y, $\acute{y}$: k-bit secret key vectors.
- $\nu, \acute{\nu}$: noise bits (=1 with probability $\eta \in [0, 1/2]$).
- r: total number of rounds for which the protocol runs.
- ρ: index of the current round, $\rho \in \{1, 2, \ldots, r\}$.

An authentication round is initiated by the tag and proceeds as follows (see Fig 1):

1. The tag generates a k bit random blinding vector b and sends it to the reader.
2. In response, the reader generates a k bit random challenge a and sends to the tag.
3. The tag computes the vectors z and $\acute{z}$ where
   $z = a \cdot x \oplus b \cdot y \oplus \nu$
   $\acute{z} = rot(f(a), \rho) \cdot \acute{x} \oplus rot(f(b), \rho) \cdot \acute{y} \oplus \acute{\nu}$ and sends back to the reader.

The tag is authenticated if the check on the reader side fails at most $r\eta$ times ([9]).

The security of the HB++ protocol is based on the Learning Parity with Noise (LPN) problem and not on

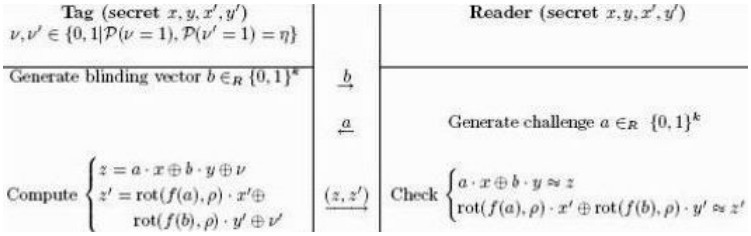


Figure 1. A round of the HB++ protocol

classical symmetric key cryptographic schemes. The LPN problem has been proved to be computationally hard ([8]); this fact provides the security to HB++. HB++ is based on the same security model on which HB is based, this implies that it is at least as secure as HB is. But HB++ takes into account a wider adversarial model compared to HB and HB+, it encompasses the possibility of an active attack and a man-in-the-middle attack. Thus it is widely more secure than HB and HB+ while maintaining the simplicity and low-cost design.

3 Description of the proposed architecture

3.1 Description of the hardware

The block diagrams of the designed system are presented, both on the reader side and the tag side. Fig 2 shows the unit that computes the expressions $rot(f(a), \rho)$ and $rot(f(b), \rho)$. They are stored in the registers a_temp and b_temp respectively. This unit is common to both the tag and the reader since both of the needs to calculate the given expressions. However, the units will be separately implemented on each side. The same hardware computes $rot(f(a), \rho)$ and $rot(f(b), \rho)$ by using a mux and a demux. A select line ($select\_a\_b$) indicates whether the unit is working with a or b by manipulating the mux and the demux. This line also routes the computed values to the appropriate registers.

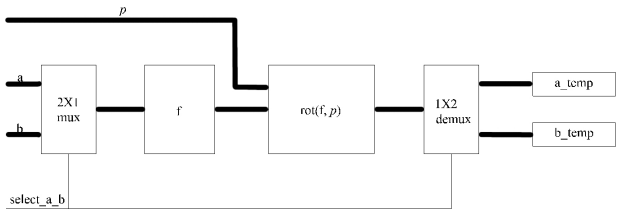


Figure 2. Functional unit common to tag and reader

We have followed the construction of the function f and

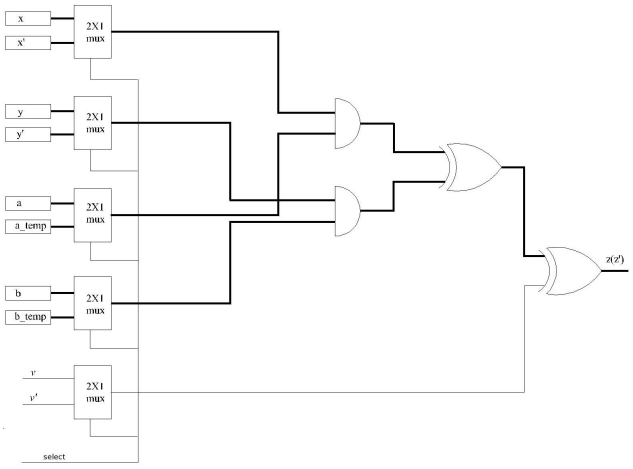


Figure 3. Tag authentication unit

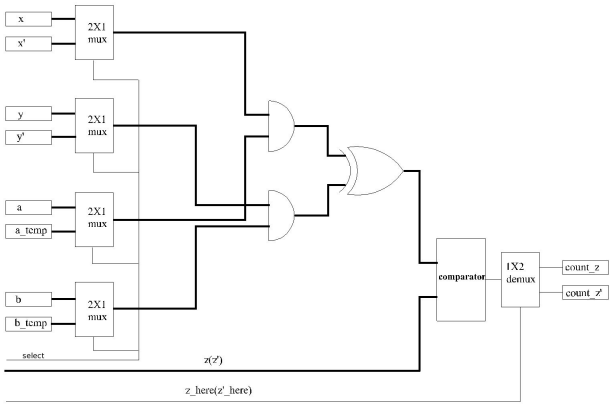


Figure 4. Reader authentication unit

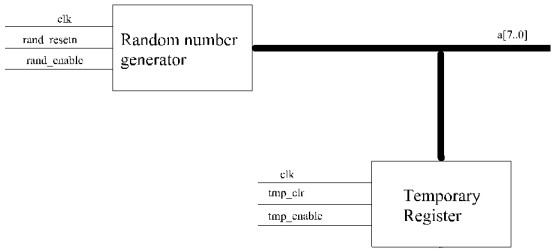


Figure 5. Random number generation unit

the function rot given in [9]. The function f transforms five low order bits of the input and leaves the rest unchanged. The function rot is simulated using LPM megafunction package available in Quartus's library [10]. At each round, the given bit vector is rotated right by two bits (there is no loss of generality in assuming that the rotation is always right rotation).

Fig. 3 shows the tag side unit that computes z and $\acute{z}$. We have four secret key registers $x, y, \acute{x}$ and $\acute{y}$ each having a read/write line and an input data line (not shown in the figure). Here also, the same unit is used to compute z and $\acute{z}$ to save hardware resources. A number of mux are used. ν and $\acute{\nu}$ are the noise bits from the noise generation unit. The $select$ signal indicates whether z is computed or $\acute{z}$. When $select = 0$ the output is z and when $select = 1$ output is $\acute{z}$.

Fig. 4 shows the reader side authentication unit. Here we do not have the noise bits, instead we have a comparator that compares the received value of z (or $\acute{z}$) with $a \cdot x \oplus b \cdot y$ (or $rot(f(a), \rho) \cdot \acute{x} \oplus rot(f(b), \rho) \cdot \acute{y}$). The corresponding comparison results are routed through a demux to counters that counts the number of times the comparison succeeds.

Figure 5 shows the random number generator unit along with the temporary register to store values of the corresponding variables that will be used during computation of z and $\acute{z}$ and also during the comparison. Both the tag and the reader has this unit implemented.

An important tradeoff here is between hardware cost and the operation time of the unit. An alternative design could have been a parallel architecture to calculate z and $\acute{z}$ in parallel, that would save time but would result in almost double the hardware resources. We have chosen this approach since HB++ provides a good amount of security, so even if an authentication round takes longer time we do not have any serious problem. This approach saves hardware cost.

The whole system operates as follows:

1. Four secret keys $x, y, \acute{x}$ and $\acute{y}$ are written to the secret key registers of both the tag and the reader. The key in both of them must be the same for authentication to succeed.

2. Then the authentication round is initiated by the tag. It generates a random blinding vector b, sends it to the reader and also saves a copy of it into its temporary register. During this time the reader waits.

3. The reader responds by sending back a random vector a, it also saves a copy in its temporary register.

4. The tag computes z and $\acute{z}$ and sends back to the reader.

5. The reader checks whether the received values of z and $\acute{z}$ are equal to $a \cdot x \oplus b \cdot y$ and $rot(f(a), \rho) \cdot \acute{x} \oplus rot(f(b), \rho) \cdot \acute{y}$ or not and stores the comparison results in counters. The tag is authenticated if the check fails at most $r\eta$ times.

All the functional units are integrated as shown in Figure 6.

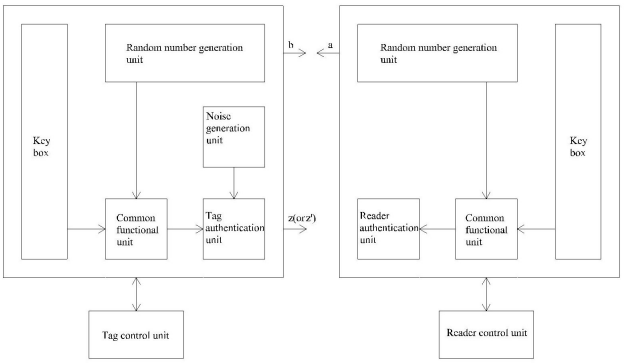


Figure 6. The whole system

3.2 Description of the control units

The operations of the functional units will be synchronized by the control units. Here we present the state tables for the control units, one for tag side and another for reader side. The actual implementation of the units can be derived easily form the state tables through standard sequential circuit design methods. The impossible input combinations have been marked with a D. Table 1 shows the state table for the tag side control unit and Table 2 shows that for the reader side control unit. We have used some extra signals to indicate presence or absence of a variable. When z_here is 0, this means that the reader has received z and when z_here is 1, the reader has received $\acute{z}$. Similarly, a_here indicates the presence (when it is 1) or absense (when it is 0) of a. The signal g indicates that an authentication round should proceed, thus when it goes low, an ongoing round must stop.

3.3 Description of the random number generation unit

The random number generator that we have used is Linear Feedback Shift Register (LFSR) based. The feedback polynomial used is $x^8 + x^6 + x^5 + x^4 + 1$ It produces pseudorandom sequence of width 8 bits and has a period of $2^n - 1 = 255$ since here $n = 8$.

4 Implementation details and results

We implemented the proposed architecture using Alteras Quartus II software [10]. All the system components were described using the structural style. The whole design was synthesized, placed and routed using Altera FPGA and CPLD devices namely Cyclone [11] and MAX II [12]. The Cyclone device used is EP1C6Q240C8 and the MAX II device used is EPM240T100C5. The total amount of resources available on each device is as follows: 5980 logic elements, 185 pins and 2 PLL for the FPGA device; 240 logic elements, 80 pins and 1 UFM block for the CPLD device. No PLL or UFM block has been used. The implementation results for the authentication units are shown in the following tables. These results do not include the control units.

Table 1. State table for tag side control unit

State	Inputs (g and a_here)		
	0X	10	11
S0 (Begin)	S5	S1	D
S1 (Generate b)	S5	S2	D
S2 (Wait for a)	S5	S2	S3
S3 (Generate z)	S5	S4	S4
S4 (Generate $\acute{z}$)	S5	S5	S5
S5 (End/Abort)	S5	S5	S5

Table 2. State table for reader side control unit

State	Inputs (g, a_here and z_here)			
	0XX	100	101	111
S0 (Begin)	S7	S1	D	D
S1 (Wait for b)	S7	S1	D	S2
S2 (Generate a)	S7	S3	S3	S3
S3 (Wait for z)	S7	S4	D	D
S4 (Check z)	S7	S5	S5	S5
S5 (Wait for $\acute{z}$)	S7	D	S6	S6
S6 (Check $\acute{z}$)	S7	S7	S7	S7
S7 (End/Abort)	S7	S7	S7	S7

Table 3. Resource usage summary of the proposed system

Component name	Logic elements	Registers	I/O pins
Reader authentication unit	95	80	94
Tag authentication unit	77	72	76
Rotation unit	0	0	35
Random number generation unit	16	16	19

5 Issues related to the implementation

A conventional tag ID is 96 bits long. To get a fair amount of security from HB++, at least 40 rounds are

required [9] with 4 bytes of raw data communication per round, assuming that all data is one byte long. This means, for an authenticated communication, the reader must communicate with the tag for about 10 times more than the duration of an un-authenticated communication. HB++ is algorithmically more resistant to side channel attacks due to the presence of the functions f and rot. If the communication snaps during an authentication round, both the tag and the reader must abort the current round and start the authentication afresh to ensure that no man-in-the-middle attack can compromise the system. However, this may elongate the authentication process if the contact between the tag and the reader is not steady as in the case of passive tags.

We consider the number of clock cycles required for each uninterrupted round of HB++. During the first clock, both the tag and the reader are initialized, in the second clock the tag generates b and saves it in the temporary register. During that time the reader waits. In the third clock, the tag waits while the reader generates and saves a. In the fourth clock, the tag generates z while the reader waits. Then the tag takes two clocks to generate $\acute{z}$ since it has to rotate vector two positions to the right. In the first of the two clocks the reader checks z and in the second it waits. In the seventh clock the tag ends its state transition for one round while the reader checks $\acute{z}$. Then both of them halt. Thus for an uninterrupted communication we need at least 8 clock cycles plus additional delays. This is large compared to 4 clock cycles for an implementation of HB (5 in case of HB+) as seen in [13].

6 Discussions

Input or output buffers are used in the implementation. Buffers are more necessary if the clock frequency of the tag and the reader are different. In that case, the time required to compute a particular function on each device will be different. This may lead to discrepancy in the functioning of the hardware. Again, clock synchronization between the tag and the reader should not be a serious problem if a suitable encoding scheme is used that supports clock recovery. If such a scheme is implemented at the data link layer, buffers may be omitted. This would save hardware cost.

We had to include two extra signals in the implementation. When a device is waiting for a signal, it is in the corresponding wait state, an extra signal is needed to change its state. That signal denotes that it has received the signal it was waiting for. Some examples are a_here, z_here etc. These extra signals do not change the basic structure of the protocol.

We have assumed the existence of a separate active high signal g, that actually begins and continues the authentication round. On the tag, this signal may be asserted by a proximity detector that detects the presence of an interrogating reader.

We expect that in most cases the protocols will be implemented on the reader in software since there is no serious resource constraint there. The final value of the counter after a number of rounds will indicate overall success or failure of the authentication.

The random number generators we have used are LFSR based. The random noise bit is generated by extracting the lowest significant bit from a random number.

As stated before, an important design tradeoff is speed versus hardware cost. We could have parallelized the architecture to a certain extent. That approach would require multiple units to do the same operation parallely. While that will make the system fast, will also increase hardware cost. Our approach, on the other hand, uses the same unit to compute functions sequentially by using multiplexers. Thus, while the time required to compute a function increases, hardware cost is decreased.

7 Conclusion and future research directions

We have proposed a scalable hardware implementation of the HB++ authentication protocol. The implementation has been simulated using Altera FPGA and CPLD platforms. The implementation results show that our design is efficient in terms of resources and so is suitable for use in constrained RFID systems, especially on tags. The protocol provides a reasonable amount of security in practical systems, the probability that an attack of the type [5] will succeed is smaller than $(2^{-4})^r$. Our architecture reflects this aspect of the protocol and also its light weighted design.

Future research directions in this field includes, above everything else, design of better low cost protocols. The protocols must be hardware efficient and provide enough security. Other NP-hard problems may be used to design such protocols. Another important factor is, our design can be improved by pipelining the different functional stages. This would improve the overall performance of the system by partially parallelizing the computations.

References

[1] A. Jules, R. Rivest and M. Szydlo, *The blocker tag: Selective blocking of RFID tags for consumer privacy*, Conference on Computer and Communications Security ACM CCS, pp. 103-111, ACM Press, 2003.

[2] S. Weis, S. Sarma, R. Rivest and D. Engels, *Security and privacy aspects of low-cost radio frequency identification systems*, International Conference on Security in Pervasive Computing SPC 2003, vol. 2802 of LNCS, pp. 454-469, Springer-Verlag, 2003.

[3] M. Feldhofer, S. Dominikus and J. Wolkerstorfer, *Strong authentication for RFID systems using the AES algorithm*, Workshop on Cryptographic Hardware and Embedded Systems CHES 2004, volume 3156 of LNCS, pages 357-370, Boston, Massachusetts, USA, August, IACR, Springer-Verlag, 2004.

[4] S. Dominikus, E. Oswald and M. Feldhofer, *Symmetric authentication for RFID systems in practice*, Handout of the Ecrypt Workshop on RFID and Lightweight Crypto, July 2005.

[5] G. Gilbert, M. Robshaw and H. Sibert, *An active attack against HB+ - a probably secure lightweight authentication protocol*, Cryptology ePrint Archive, Report 2005/237, 2005.

[6] A. Juels and S. Weis, *Authenticating Pervasive Devices with Human Protocols*, Advanced in Cryptology - CRYPTO'05, Volume 3126, LNCS, pp. 293–308, Springer-Verlag, 2005.

[7] N. Hopper and M. Blum, *Secure human identification protocols*, Advances in Cryptology ASIACRYPT 2001, volume 2248 of LNCS, pages 52-66. Springer-Verlag, 2001.

[8] J. Hastad, *Some optimal inapproximability results* STOC 1997, pages 1-10, 1997.

[9] J. Bringer, H. Chabanne, and E. Dottax, *HB++: a Lightweight Authentication Protocol Secure against Some Attacks*, Proceedings of the Second International Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing - SecPerU'06, 2006.

[10] Altera corporation, Quartus II development software handbook, `http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf`

[11] Altera corporation, Cyclone device handbook, `http://www.altera.com/literature/hb/cyc/cyc_c5v1.pdf`

[12] Altera corporation, MAX II device handbook, `http://www.altera.com/literature/hb/max2/max2_mii5v1.pdf`

[13] A. Chanda and A. Mukherjee, *On Issues Related to the Hardware Implementation of RFID Authentication Protocols*, Proceedings of CSI Research and Development of Hardware and Systems - CSI RDHS, Kolkata, India, June 2008

Suraksha: A Security Designers' Workbench

Santhosh Babu G., Vineet Kumar Maurya, Ebenezer Jangam , Muni Sekhar V.,
Asoke K. Talukder, Alwyn Roshan Pais
Information Security Lab
Dept. of Computer Engineering, National Institute of Technology Karnataka, Surathkal
santhosh003@gmail.com, vineet.nitks@gmail.com, ebenezer.jangam@gmail.com,
munisek@gmail.com, asoke.talukder@saharanext.com, alwyn.pais@gmail.com

***Abstract* — To design a secure software system, a security designer needs a workbench so that security of the system can be embedded into the system from the very early stages of Software Development Life Cycle (SDLC). Among the proposed approaches to elicit security aspects in the early stages of SDLC, researchers have suggested different techniques. However, a workbench that will facilitate a designer to build security right from the requirement analysis is not available. This paper presents a Security Designers' Workbench to achieve this.**

Keywords —Security Designers' Workbench; Attack tree; Misuse case; Security requirement analysis; STRIDE; DREAD; Security Patterns; Security Development Lifecycle

I. INTRODUCTION

In 1968 October NATO Science Committee organized a conference on Software Engineering [1]. Delegates of this conference discussed the Software Crisis. They also discussed how this crisis could be addressed through Software Engineering. In last forty years different techniques in Software Engineering have been proposed to address this crisis. Today Software Engineering is a well established discipline that includes engineering practices that includes requirement elicitation, design, construction, testing, and maintenance of software.

Combined with the growth of Internet and Telecommunication, world is today experiencing a different type of crisis – this is the Software Security Crisis. Due to lack of security awareness, software that are being developed lack proper security considerations and quite often they are compromised. As a result the assets behind these software are not always secure and safe. To address this crisis we need Secure Software Engineering.

Security has been considered to be a nonfunctional requirement. The nonfunctional requirement is defined as – "A requirement definition that includes all requirements that are not functional". It can also be defined as how the software will behave for a function that is not defined or known. The functional requirement is defined as – "A system or software requirement that specifies a function that a system or software system or its component must be capable of performing. These are requirements that define behavior of the system or the software".

Till sometime ago, security for a software system was managed through perimetric security. It was not mandatory that applications should be security aware. That is why security has always been an afterthought. Now-a-days, it is obligatory to develop security aware applications. Security needs to be embedded into the system from the early stages of Software Development Life Cycle to develop a security aware application. Hence, security should be considered as functional requirement to avert the present Software Security Crisis.

There are several benefits if security is considered as functional requirement and embedded into the Software Development Life Cycle right from requirement analysis phase. The earlier is the focus on security, the more is the insight obtained into the security aspects of the system. Various advantages of identifying security aspects in early development stages are outlined in [2]. The early focus on security gives the security designer an opportunity to find out countermeasures also in the earlier phases of lifecycle. Developer also can be aware of security aspects of the system before developing the system. These factors pave way to build a secure system against different kinds of attacks. Unfortunately, there are no tools to support security designer to identify and implant security aspects from the very early phases of development.

In this paper we present a system named *Suraksha*, which is a Security Designer's Workbench that comprises of interfaces that help designing secure software system. *Suraksha* in Sanskrit means security. This tool helps a security designer to elicit security requirement followed by security design.

In functional requirement analysis, we start with use case; however, in security related non-functional requirement we start with misuse cases. Misuse cases specify all the actions the system should not do as mentioned in [3, 4].

The organization of the paper is as follows. Section 2 explains the related work done in this area and available tool support. Section 3 gives the outline of the holistic approach recommended for a Security Designer. Section 4 provides information about various features of Security Designers' Workbench *Suraksha*. We conclude in section 5.

II. Related Work

Use cases are not sufficient to elicitate, communicate and document security requirements of a system. Hence, misuse cases are proposed by Sindre and Opdahl [3] to represent the security requirements of a system. Textual representation of misuse cases was proposed and explained in [3, 5]. They gave a five step procedure for misuse case development [5]. Misuse cases can be either lightweight or extensive. If a misuse case is lightweight, it does not aid developers; and, if it is extensive, it will be useful for later development stages [6].

According to Ian Alexander, an approach combining use and misuse cases can be applied at any level in systems engineering [7]. Moreover, it was stated that sometimes it is worth documenting misuse case scenarios in detail and other times just the name of misuse case is enough [8].

As said by Donald G. Firesmith [9], Security Use cases should be used to specify requirements that the system shall successfully protect itself from its relevant security threats while misuse cases are suitable for analyzing and specifying security threats.

While introducing attack trees, Bruce Schneier [10] stated that attack trees provide a formal methodical way of describing the security of systems, based on varying attacks. Attack trees or threat trees provide insight into the attack or threat mentioned at root node. Child nodes of a node are connected using either AND or OR component.

Misuse case helps to identify the cases that need to be further analyzed. Attack tree provides the detailed information about different ways of performing an attack [10, 11]. Attack Trees and misuse Cases provide complementary information. In the one hand, attack Trees provide a systematic way to symbolize attacks. On the other hand, a misuse case gives additional information that is more accessible and useful to a developer. The combination of attack Tree and misuse Case gives more information to designers as mentioned in [12].

STRIDE is a methodology for identifying possible threats [13, 14, 15]. It is used by Microsoft for threat modeling of their systems. Threats are identified by exploring the possibilities of Spoofing Identity, Tampering with Data, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege in the given case.

The DREAD methodology [15, 16] is another tool to determine possible threats and their impact. This acronym is also formed from the first letter of each category. DREAD modeling not only tries to identify a threat, but it also influences the thinking behind setting the risk rating, and is also used directly to mitigate the risks. While calculating the risk of a threat, its Damage Potential, Reproducibility, Exploitability, Affected Users and Discoverability of the threat are considered [16].

A design pattern is a formal way of documenting successful solutions to problems. The idea of design patterns was introduced by Christopher Alexander and has been adapted for various other disciplines. The goal of patterns in software engineering is to create a body of literature to help software community to map recurring problems into patterns through the software development lifecycle [17].

Thus researchers gave various kinds of approaches that are explained in various papers. But, the absence of holistic approach brings confusion and ambiguity in combining different approaches.

As explained in [17] by Asoke K Talukder, all of the above techniques when effectively combined together in security requirement analysis phase, can be used to reduce the attack surface. This literature also describes how to do the construction of the software in a secure fashion followed by security testing.

Regarding the available tool support, the present situation is not at all encouraging. Even though the combination of all the approaches is promising for security requirement analysis, hardly any tools support all the approaches. Even tools supporting individual techniques are rarely found. Misuse case modeling tool is not available and a few tools only support attack tree. Open source tools are hardly ever available for attack trees and misuse cases. Absence of holistic approach and lack of open source tools are genuine matters of concern for a software designer.

III. Our Approach

Mamadou, Jose, Susan and Debra anticipated that the combination of misuse cases and attack trees will provide more information to the designer [12]. Moreover Sindre and Opdahl generalized the concept of misuse cases [18]. They also suggested that combination of misuse cases and attack trees is beneficial [6]. This combination forms an alternative to common criteria for specifying security requirements [12]. While the Common Criteria is comprehensive and thorough, it does not provide any tool to quantify and implement it. Furthermore, it does not provide a way to specify detailed

information such as preconditions in security objectives that might prove useful for the designers [12]. Considering STRIDE to identify threats and DREAD to calculate the risk of the threat, our approach reduces attack surface by effectively combining all the techniques together. Also concept of security patterns is used for formal documentation of recurring problems. Moreover, this is a open source tool that allows the community to improvise it.

Step 1: At the very first step, system objectives are identified. Assets are also identified along with their associated risks. An example of asset identification and prioritization is presented in Fig. 1. We followed the procedure explained in [19] to identify and prioritize assets. As a first step, a brainstorming session is conducted and all the valuable assets are listed. Next step is to examine various existing documents for other important assets. Once all the assets are listed, the assets are categorized and prioritized with respect to security. To perform this, an asset is taken and viewed from different perspectives i.e. customer, administrator and attacker. From each perspective, each asset gets assigned a number indicating the importance of confidentiality, integrity or availability for this asset. All the priorities of each asset are added and the asset with lowest sum is ranked as highest priority asset.

Step 2: Functional requirements of the system are analyzed using Use case and UML tools. Use case diagram specifying the functional requirements of the system is developed.

Step 3: For each actor in the above Use case, a misactor (one or more) is assumed and STRIDE [13, 14, 15] concept is applied in connection with each action and assets related to it. Here we use the same terminology and same graphical notations as suggested by Sindre[3].The possibilities for Spoofing Identity, Tampering with Data, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege in the given case by misactor are explored. This yields a list of possible abstract threats. A light weight misuse case is developed. This is explained in detail in Section IV.

Step 4: Now, a list of abstract threats is available. Each threat needs to be explored further for getting a clear understanding about different ways of accomplishing the corresponding attack. Hence, each abstract threat in the above misuse case diagram is considered as a root node and corresponding attack tree is developed to understand what are the AND and OR components in the threat path. This is shown in Fig. 4. *Suraksha* has many libraries of threat models that help model the threat tree. User can use these trees as it is or enhance to create the threat forest.

Once different ways to accomplish threat are explored by drawing an attack tree, DREAD is used to rate a threat. This is done by assigning values to each node beginning from leaf nodes. The values are percolated up after calculation at each node. This concept is illustrated in Fig. 5

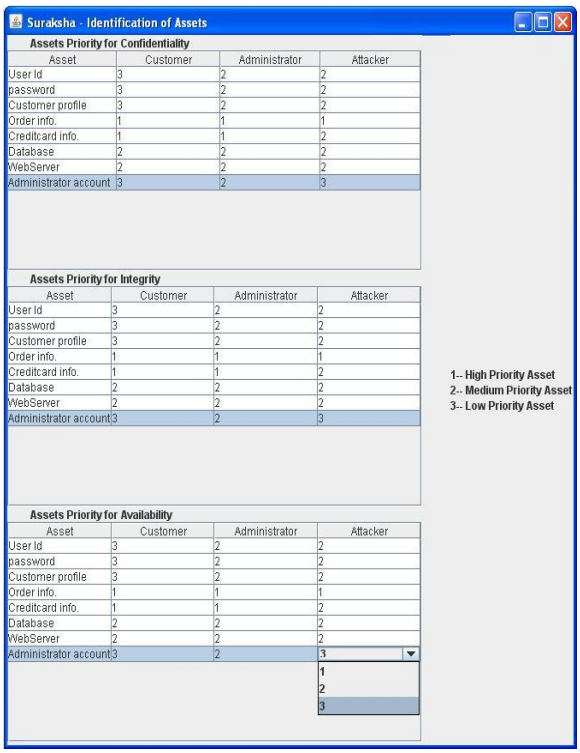


Fig. 1 Assets identification for e-commerce Application

Step 5: Finally, depending on DREAD rating suitable value is assigned to these threats and then compared with cost of assets. If it is too expensive to secure an asset compared to the cost of the asset, those threats need not be considered. DREAD tool is also used to prioritize the threat mitigation plan.

After careful examination of each threat using DREAD, the threats are classified as 'considered' and 'neglected'. For each threat considered, identification is given as a high risk or moderate risk or low risk.

Step 6: Detailed information about each threat is now available in the form of corresponding attack tree. Using the available information in the form of a forest (group of attack trees), misuse case diagram in step 3 is modified and now extensive misuse case is developed.

Step 7: The security requirement is finalized using DREAD. Here all these threats that filter through the

DREAD rating are considered as functional requirement for a security aware system. Protection of assets through security that was considered a non-functional requirement then becomes a functional requirement. Use case diagram and misuse case diagram are represented together with these misuse cases converted into a use case and the misactor converted into an actor. Possible mitigation techniques are suggested in the form of security use cases.

Step 8: In next step all the Security Patterns are considered and ensured that these patterns are included in design. If there are patterns but no use-case, the iteration starts from Step 1.

All the security patterns are listed below. Joseph Yoder and Jeffrey Barcalow were first to adapt security design patterns for information security. It is easy to document what the system is required to do. However, it is quite difficult and sometime impossible to define what a system is not supposed to do. The Yoder and Barcalow paper presented seven patterns in 1998 for security design [20]. They are:

1) *Single Access Point*: Providing a security module and a way to log into the system. This pattern suggests that keep only one way to enter into the system.
2) *Check Point*: Organizing security checks and their repercussions. Authentication and authorization are two basic entity of this pattern.
3) *Roles*: Organizing users with similar security privileges.
4) *Session*: Localizing global information in a multi-user environment.
5) *Full View with Errors*: Provide a full view to users, showing exceptions when needed.
6) *Limited View*: Allowing users to only see what they have access to.
7) *Secure Access Layer*: Integrating application security with low-level security.

To manage the security challenges of networked computers today, we need to look at many more design patterns. However following patterns must be included in any security system [17].

8) *Least Privilege*: Privilege state should be shortest lived state.
9) *Journaling*: Keep a complete record of usage of resource.
10) *Exit Gracefully*: Designing systems to fail in a secure manner.

Finally the attack surface is analyzed and above process is repeated until the attack surface is reduced to the required level.

The modified misuse case diagram serves as an input to threat modeling in the next phases of software development life cycle. Thus, this misuse case diagram enhances the process of threat modeling [21, 22]. Moreover, this misuse case diagram along with different attack trees will be available for a security designer.

As an example, we have presented the e-commerce application that was used by Sindre and Opdahl [3] throughout this paper. Two actors considered in this example are operator and customer. Operator or administrator has responsibility to maintain the system. Customer is authorized person to browse catalog and order goods. Crook is a malicious user who always tries to exploit the vulnerabilities of the system. Various misactions that can be performed by a crook are Denial of service, Spoofing identity and Information disclosure.

IV. SURAKSHA: OPEN SOURCE TOOL

At *National Institute of Technology Karnataka Surathkal*, we have developed the Open Source Security Designers' Workbench tool named *Suraksha*. The tool is freely downloadable from http://isea.nitk.ac.in/suraksha/. *Suraksha* supports various important features like Assets identification and prioritization, Textual representation of misuse cases using misuse case template, Co-representation of use and misuse Cases, attack tree development, DREAD Rating, Support for security patterns etc.

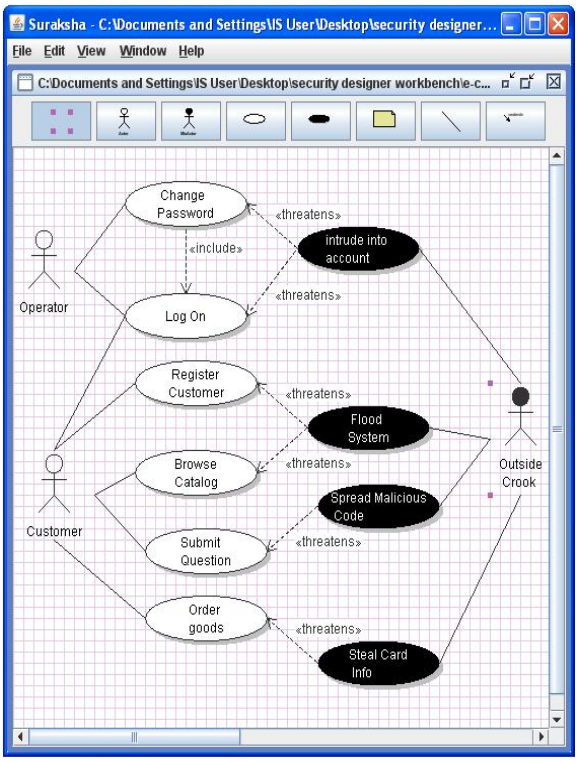


Fig. 2 Misuse-Case diagram for e-commerce Application

Suraksha offers support for asset identification and prioritization as represented in Fig. 1. Assets are classified into two categories based on the mobility. They are static assets and assets in transit [17].

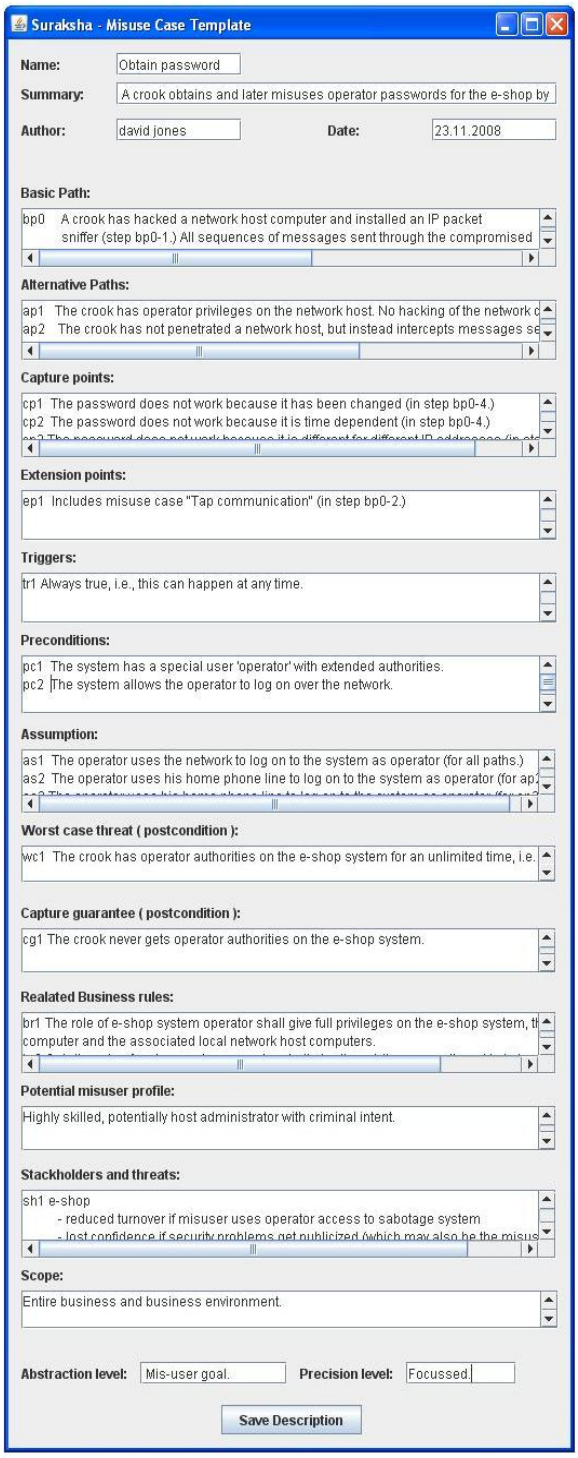


Fig. 3 Textual representation of misuse cases

Suraksha facilitates user to list all the valuable assets. It promotes not only listing valuable assets but also prioritizing assets based on the values assigned by the user. User needs to enter values for Confidentiality, Integrity and Availability from the perspective of stakeholder and attacker. User need to choose one from three numbers 1, 2 and 3 provided using drop down box. Value 1 denotes high priority indicating that the importance of confidentiality, integrity or availability for the corresponding asset is more. If user didn't select any value, default value is taken as 4. Once the assignment of all values is finished, the sum of all values for each asset is calculated. The lesser is the value of sum for an asset, the higher is the priority given to the asset.

Suraksha provide a simple and efficient GUI to draw misuse case diagram as shown in Fig. 2. User can easily add actor node, misactor node, use case node, misuse case node and can easily draw various relationship between them like extend, mitigate, threaten etc by selecting suitable item from the panel. To represent a system behavior, we should include both functional and non-functional requirements together. In order to represent this in graphical fashion, use case and misuse case should be combined to define the system. To represent use cases and misuse cases together, they need to be differentiated. Therefore, use case is black in white and misuse case is shown in an inverted format – white in black. *Suraksha* provides attractive GUI to draw use cases and misuse cases and to co-represent them. Within a short time interval, user can draw misuse cases and use cases easily using this tool. The size of the diagram can be adjusted as required. In the example presented in Fig. 2, operator and customer are actors and their corresponding permitted actions are shown. Crook is also represented along with some of the possible malicious actions.

Suraksha offers user friendly GUI for the user to document the textual representation of misuse cases. Sindre and Opdahl focused on templates for misuse cases in [23]. *Suraksha* uses the misuse case template suggested by them. We considered the example used by them in [23]. The textual representation of the example Misuse case is shown in Fig. 3.User can enter corresponding information against each field in the provided text boxes. After completion of giving all the necessary information, user can save the textual representation.

Suraksha provides attractive GUI for attack tree development as depicted in Fig. 4. For each abstract threat mentioned in misuse case, detailed information about the threat can be obtained by drawing an attack tree corresponding to the threat. User can draw an attack tree easily using this tool starting with abstract threat as root node. Various paths possible to achieve

the goal (root node) are explored. User can easily draw all possibilities by creating children to a node and connect these children using AND or OR component. The AND component is represented by straight line and OR component is represented using double line arc. User just need to select the required items from panel and can place the items in required position. To facilitate the designer, there are some standard threat models available in the library and can be used by the user. These threat models help to identify various attacks and their relationship. In real system these threats need to be mitigated. Also, the impact of these threats needs to be measured.

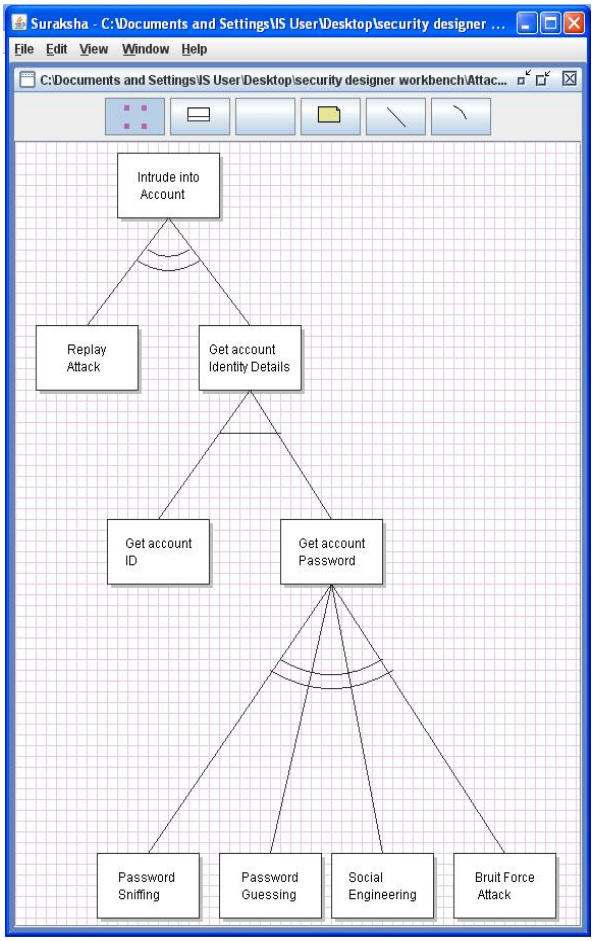


Fig. 4 Attack tree for Intrude into Customer Account in e-commerce application

To measure the impact of each threat, *Suraksha* uses DREAD methodology. When a node in an attack tree is selected, there is provision for the user to enter suitable values for Damage Potential, Reproducibility, Exploitability, Affected Users and Discoverability. By selecting proper radio button, user can choose either 0 or 5 or 10.After completion of giving information, risk associated with corresponding node is calculated automatically based on the formula mentioned below.

Risk DREAD = (D+ R + E +A + D) / 5

Where,

D=Damage Potential,

R=Reproducibility,

E=Exploitability,

A=Affected Users,

D=Discoverability.

The calculation always produces a number between 0 and 10; the higher the number, the more serious the risk. The corresponding snapshot with DREAD rating for the attack tree is shown in Fig. 5.

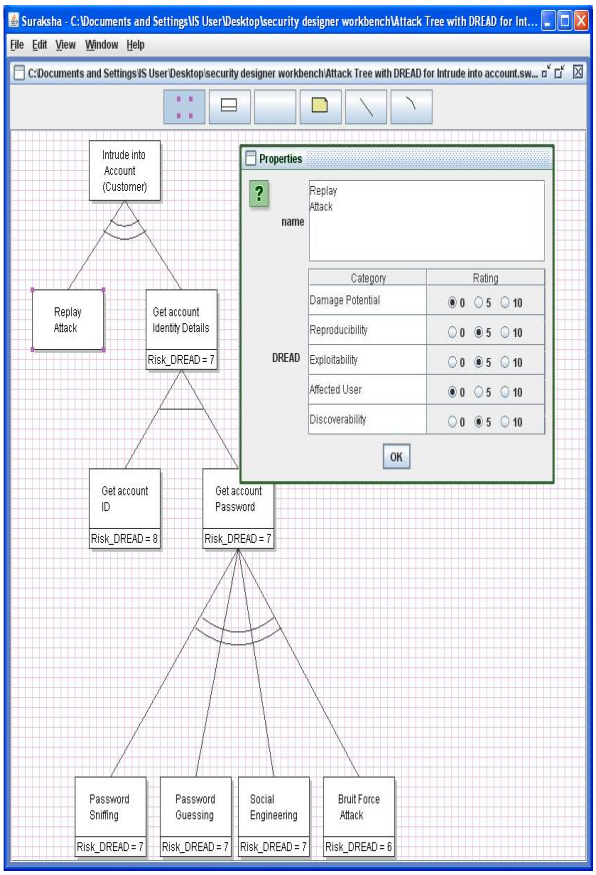


Fig. 5 Attack tree with DREAD Rating for Intrude into Customer Account in e-commerce Application

Suraksha provides unique provision to work with misuse cases and attack trees. For each misaction in lightweight misuse case, an attack tree with misaction as root node can be created. It facilitates the user to think of various ways of performing the attack in the perspective of a hacker. Depending on rating and values obtained for different threats using attack trees, threats are filtered and corresponding extensive misuse case diagram can be generated. Thus this tool reaps the benefits of using attack trees and Misuse cases together. This extensive misuse case diagram

can be saved and forms an excellent input for threat modeling.

Finally, it is very important to provide formal documentation for various problems along with their solutions. The purpose of this documentation is to use these solutions again when the same problems are encountered in future. For recurring problems this approach is very useful. *Suraksha* offers vital support for security designers to solve recurring problems using security patterns. A documented pattern contains details about the problem that the pattern addresses, the context in which this pattern should be used, the solution etc.

V. CONCLUSION

This paper focuses on a security designers' workbench named *Suraksha*. This Open Source workbench tool is developed at the *Information Security Lab, Department of Computer Engineering, National Institute of Technology Karnataka, Surathkal*. The workbench allows a security professional to design security aware applications. The workbench allows user to analyze and capture security requirements through misuse case, followed by identification of threats through attack tree. Once the attack paths are known, the tool allows the user to rate different threats and to prioritize them. This results in a list of possible attacks. These attacks along with their corresponding countermeasures are included as part of functional requirement. This is then used to design the secure system through security patterns. Thus, this open source tool supports security designer to elicit security requirements in the early phases of Software Development Lifecycle (SDLC). As an example we analyzed a part of e-commerce application using this tool. Also, a holistic approach to use various techniques proposed by researchers is outlined.

ACKNOWLEDGEMENTS

We would like to thank the Dept. of Computer Engineering , NITK Surathkal and the ISEA –Project, MCIT, NewDelhi for providing facilities to carry out this research work. We would also like to thank the anonymous reviewers for their constructive feedback, which has enhanced the quality of this paper. Finally we would like to acknowledge the co-operation of all Information Security Lab users.

REFERENCES

[1] P. Naur and B. Randell, (Eds.). Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO (1969) 231pp.

[2] Takao Okubo and Hidehiko Tanaka, "Identifying Security Aspects in Early Development Stages," in Proc. *3rd International Conference on Availability, Reliability and Security (ARES'08)*, Barcelona , Spain, Mar. 2008, pp. 1148–1155.

[3] Guttorm Sindre and Andreas L Opdahl, "Capturing Security Requirements by Misuse Cases," in Proc. *14th Norwegian Informatics Conference (NIK'2001)*,Troms, Norway, Nov. 2001.

[4] Chandramohan Muniraman and Meledath Damodaran, "A practical approach to include security in software development," *Issues in Information Systems*, Volume 2, No. 2, pp. 193-199, 2007.

[5] G. Sindre and A.L. Opdahl, "Eliciting Security Requirements by Misuse Cases," in Proc. *37th Conf. Techniques of Object-Oriented Languages and Systems, TOOLS Pacific 2000*, 2000, pp. 120–131.

[6] G. Sindre and A.L. Opdahl,"Eliciting security requirements with misuse cases," *Requirements Engineering*, Vol. 10, No. 1, pp. 34-44, Jan.2005.

[7] Ian Alexander, "Misuse cases help to elicit non-functional requirements," *Computing & Control Engineering Journal*, vol. 14, no. 1, pp. 40–45, Feb. 2003.

[8] Ian Alexander, "Misuse Cases: Use Cases with Hostile Intent," *IEEE Software*, vol. 20, no. 1, pp. 58-66, Jan. 2003

[9] Donald Firesmith: "Security Use Cases," in *Journal of Object Technology*, vol. 2, no. 3, pp. 53-64, May-June 2003.[Online]. Available:http://www.jot.fm/issues/issue_2003_05/column6

[10] Bruce Schneir, "Modeling Security Threats", December 1999. [Online]. Available: http://www.schneier.com/paper-attacktrees-ddj-ft.html [Accessed: Aug. 17, 2008].

[11] Fredrik Moberg, "Security Analysis of an Information System using an Attack tree-based Methodology," Master's Thesis, Chalmers University of Technology, Goteborg, Sweden, 2000.

[12] Mamadou H. Diallo, Jose Romero-Mariona, Susan Elliott Sim and Debra J. Richardson, "A Comparative Evaluation of Three Approaches to Specifying Security Requirements," presented at 12th Working Conference on Requirements Engineering: Foundation for Software Quality, Luxembourg, 2006.

[13] Shawn Hernan, Scott Lambert, Tomasz Ostwald, Adam Shostack, " Uncover Security Design Flaws using The STRIDE Approach" msdn.microsoft.com, Nov. 2006. [Online]. Available: http://msdn.microsoft.com/en-us/magazine/cc163519.aspx. [Accessed: July 21, 2008].

[14] "The STRIDE Threat Model". [Online]. Available:http://msdn.microsoft.com/en-us/library/ms954176.aspx.[Accessed: July 21, 2008].

[15] F. Swiderski and W. Snyder, *Threat Modeling.* Washington: Microsoft Press, 2004.

[16] J.D. Meier, Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla and Anandha Murukan, "Improving Web Application Security: Threats and Countermeasures," *msdn.microsoft.com*, June 2003.[Online].Available:www.msdn.microsoft.com/en-us/library/aa302419.aspx [Accessed: July.29, 2008].

[17] Asoke K Talukder and Manish Chaitanya, *Architecting Secure Software Systems*, Auerbach Publications, 2008.

[18] Guttorm Sindre, Andreas L. Opdahl, Gøran F. Brevik, "Generalization/specialization as a structuring mechanism for misuse cases," in Proceedings of *2nd Symposium on Requirements Engineering for Information Security*, 2002.

[19] Martin Gilje Jaatun and Inger Anne Tondel,"Covering Your Assets in Software Engineering," in *Proc. 3rd International Conference on Availability, Reliability and Security (ARES'08)*, Barcelona , Spain, Mar. 2008, pp.1172–1179.

[20] Joseph W. Yoder and Jeffrey Barcalow, "Architectural Patterns for Enabling Application Security," in Proc.*4th*

Conference on Patterns Languages of Programs (PLoP '97) Monticello, Illinois, Sept.1997.

[21] John Steven and Gunnar Peterson, "Defining Misuse within the Development process," *IEEE Security& Privacy*, Nov/Dec. 2006.

[22] Anurag Agarwal, "Threat modeling enhanced with misuse cases," s*earchsoftwarequality.techtarget.com*, April 2006.[Online].Available:http://searchsoftwarequality.techtarget.com/tip/0,289483,sid92_gci1186821,00.html [Accessed: Aug.2,2008].

[23] Guttorm Sindre and Andreas L. Opdahl, "Templates for Misuse Case Description," in Proceedings *of the 7 th International Workshop on Requirements Engineering, Foundation for Software Quality (REFSQ'2001),* Interlaken, Switzerland, June 2001.

Proxy Traffic Control: Exploring Solutions

Meetanshu Gupta, Vijay Laxmi, and M.S. Gaur
Department of Computer Engineering
Malaviya National Institute of Technology,Jaipur
meetanshu.gupta@gmail.com, {vlaxmi|gaurms)@mnit.ac.in

Abstract

Internet monitoring and filtering has its proponents and opponents. In recent times, it is become imperative for controlling the flow of information for regulation purposes as well. Of late, various solutions, in form of websites and software have burgeoned to access the filtered/censored content, causing troubles alike to individuals and organizations. Thus, over past couple of years, efforts to curb these means of accessing blocked contents have increased significantly. Effective ways to curb some of these methods are already in use but variants of proxies still baffle the researchers, and information security community is struggling to find neat and full proof solutions. In this paper we are having an insight look into the working of Proxy traffic. Through this work in progress we present some of the possible solutions for the control of Proxy traffic at varying levels. We expect that by the time of the workshop we will have the implementation results of our proposed methods.

I. INTRODUCTION

Internet has evolved as a means of storing and sharing information efficiently, globally, and quickly. In the past decade with the rapid advent of Internet as an information medium, organizations and governments blocked varying proportion of Internet content. The term 'Internet filtering' was then coined [1]. Soon with the availability of porn and other undesired materials on Internet, parents also began to use some software to prevent children from accessing such sites. But with more and more nations banning internet content, people felt that 'Internet filtering' is turning into 'Internet Censorship', with their right to liberty of accessing information being violated. Hence many individuals devised many ways to share and access the blocked content. Other than restoring the right to access information freely, these smart tools have given access to porn and other undesirable material to children, unsocial and dangerous material to common people, and accessing material that cause IPR violations etc. Recently, global terrorism has added another dangerous dimension for sharing information clandestinely.

This paper presents a survey of methods to tackle Proxy traffic, some of them exploiting the present weakness of the proxy traffic, hence fully efficient but temporal while some others not so efficient, but capable of modifications as future forms of proxy traffic may require. In general, we propose that a solution which might be able to work with all the forms of proxy traffic and framework which can address generic mutation of proxy traffic.

II. PROXY TRAFFIC

Proxy traffic is a collection of internet packets that are being sent to or from a blocked IP address, with the help of a third-party server, which acts as an intermediate between the client and blocked IP address. This server has access to the blocked IP address. Thus, though the packets actually are interchanged between client and blocked sever, yet because of the intermediate server the true exchange is concealed. Hence the exchanged packets comprise Proxy traffic and the intermediate server is called the Proxy server. This is different from the proxy server, used as a cache, to reduce the access time of previously visited web pages [2]. The websites that provide proxy server mechanism are called Proxy sites and the software that help in do so are called Proxy.

Graphically, Fig. 1, depicts the mechanism of Proxy Server in a real-world scenario. The simple underlying mechanism can be broken in two steps:-

- Client wants to send a request to blocked server : The packet to be sent to the blocked server is instead sent to the Proxy server. Thus when the packet passes through filtering software, it does not have the IP of banned server but of Proxy server and hence is not blocked. The Proxy server then forwards the packet to the required server, on its behalf.
- Blocked server sends the response to client: The response packet is sent to the proxy server, since proxy server sent it on its behalf. Proxy server then sends this packet to the client. Now since the source address is of Proxy server and not the blocked server, the filtering software lets it pass through to the client.

A. Classification

The response packet from the blocked server, containing the filtered material, when comes from Proxy server can be used to classify Proxy Server:-

- Transparent Proxy-The data in the packet has been encrypted by a standard/non-standard encryption protocol mutually agreed by client and proxy server.

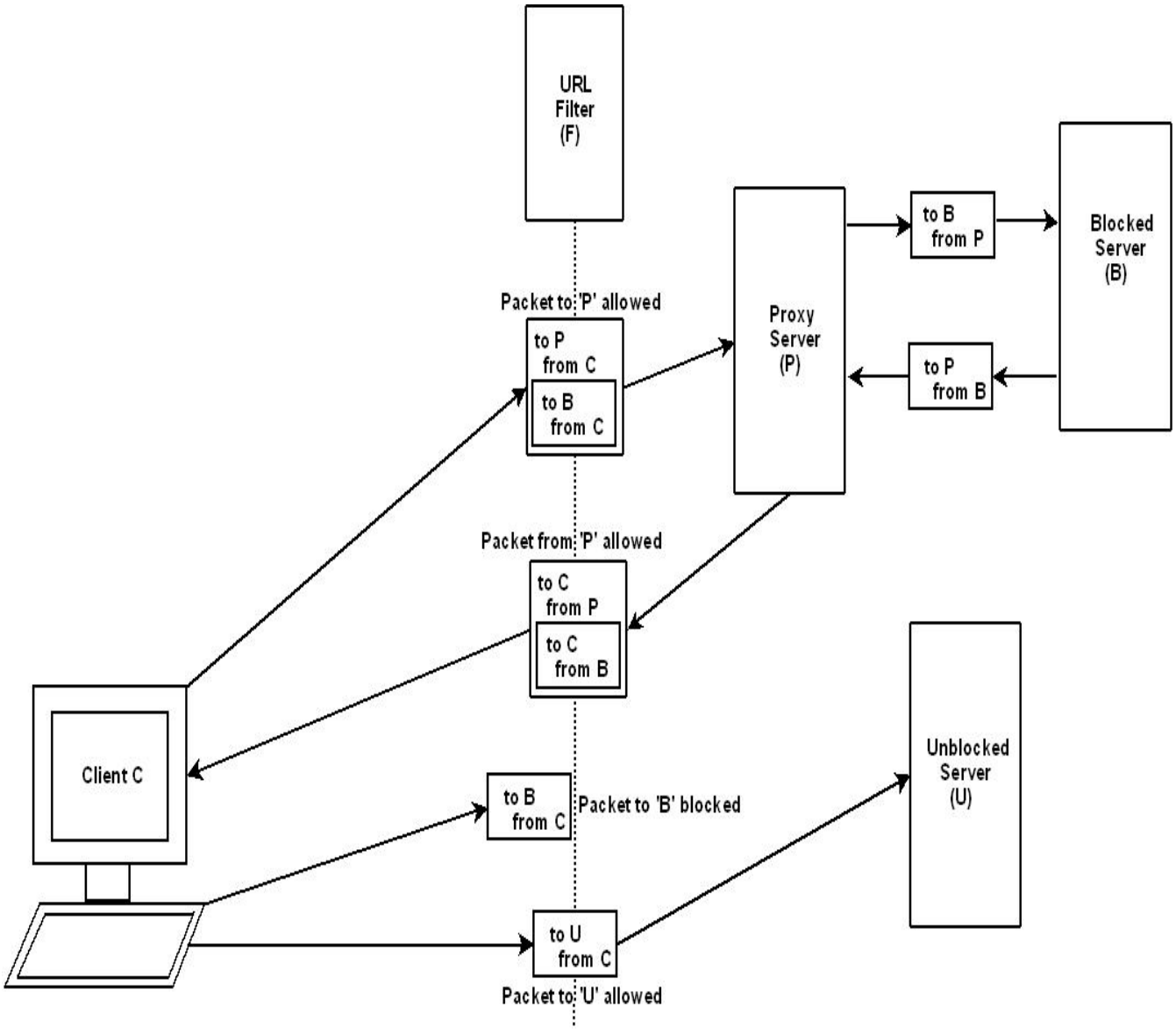


Fig. 1 : Working of Proxy Server

- Non-Transparent Proxy - The data in the packet has not been encrypted and even contains the original source address, that is the address of blocked server. Example- ultrasurf.

The Proxy server is of the central importance in this set-up and based on its IP address, it can be classified into:-

- Fixed IP address – This happens mostly in case of websites with fixed URL's like www.kproxy.com.
- Variable IP address – This requires client to run a certain executable on its machine to be able to access blocked websites. The client's working is transparent to user. It finds the available Proxy Server(s), establishes and maintains the communication with it. It automatically manipulates the sent and received packets.

B. Potential Threats

Other than making the undesired and harmful material available to children and other people, following risks have been identified in accessing the Proxy servers:-

- The Proxy traffic requires client to drop their defenses against the incoming Proxy traffic. The trust can be violated accessing private information from the client computer.
- The executable needed to be run makes it authorized to access anything including the computer itself and turn it into phatbots .This mainly can cause your machine to

be used in some sort of virus spread or Denial-of-Service attacks without your permission.

III. Stopping PROxy Traffic

For an individual with an access to internet, stopping Proxy Traffic, is just a resolution to stop using the above mentioned proxies. This may not be the case with organizations, universities and other groups as its members may want to visit the restricted sites.

The following solutions for the same assume that there is a central gateway of the organization, through which all the computers inside it access the internet. This is needed to ensure that there is one location from which all the inbound and outbound traffic can be monitored. Further a basic firewall is put in it.

A. URL Filter

A URL filter is the way to stop packets to some fixed URL's and IP addresses. This put as an add-on to the firewall can prevent access to Proxy websites. This is the foolproof way of stopping such websites and the basic step to stop proxy traffic. An example of such URL filter is URLfilterDB. http://www.urlfilterdb.com/
A free version of it, to be used with Squid Proxy is available for download at
http://www.urlfilterdb.com/url_filter_sales/index.shtml

B. Stopping Non-Transparent proxy

It needs a Squid Proxy server [3] to be configured for the corporation. Squid has an inbuilt feature by which such non-transparent proxies can be blocked by simple just switching on the same feature in its configuration file.

Problems

Mentioned above are the two solutions that have been worked on and are widely accepted to stop Proxy Websites and Non-Transparent proxies. The above solutions while easy to implement are incapable to stop the Transparent proxy with changing IP address. Further a client side executable, run by the client, searches the servers dynamically, hence make it very hard to track down Proxy servers and stop. Example – freegate. This is the case for which no established solutions exist.

We present three solutions here having a different combination of effectiveness against the proxy traffic and capability to be extended further. These are to be used in conjunction with the above two methods to provide a complete solution to Proxy traffic. That means provided above two namely, URL filter and Organization's Proxy Server are installed, any of the following three solutions can be put along to curb the proxy traffic. Advantages and disadvantages are mentioned along with.

C. Packet Sniffing

This approach is to primarily detect if the proxy traffic exists on the network. In this approach we sniff the packets being exchanged by a client. If they satisfy a certain criteria we assume that they are exchanging the packets with a Proxy Server.

The approach demands a rigorous study of all packet exchange by various proxies and identifies certain traits that are peculiar to the proxy traffic. The traits may not be same across various proxies. Hence a heuristic based formulation needs to be driven for the same. The approach can be outlined as follows.

All the packets or packets at regular interval of frequency are copied. More the number of packets examined in a set of packets exchanged, earlier is the detection of Proxy traffic. In other words, greater the sampling rate, earlier we find out if Proxy traffic exists on our network. These packets are, then, processed by the proposed software. The software checks the packet's characteristics against a set of properties verified to be found in a proxy traffic packet. Depending upon the peculiarity of the property that is matched, a numeric value would be assigned to the client. Higher the uniqueness of the property as being associated with the Proxy traffic, higher is the numeric value assigned. As soon as this numeric value exceeds a particular threshold, it can be concluded that client is using Proxies and appropriate action should be taken. The numeric value of client may be reset after the appropriate action has been taken. Further, the numeric value can be made to be dependent on the past behavior of the client.

Advantages

1. This approach is favored by numerous sniffing tools, with easy to use API, being available.
2. Further it is capable of modifications as a change in some of heuristics involved would be capable of dealing with new ramifications of Proxy traffic.
3. Coping with the change in Proxy traffic's characteristics is as simple as addition of new heuristic to the set of heuristics present in an underlying framework.
4. Customization of the heuristics and thus the implementation is an unique advantage.

Disadvantages

1. People can object to the data sent over the internet being examined, thus infringing their privacy.
2. Despite the hardware being so cheap, the set-up will incur a high cost, due to the huge requirement of hardware to examine the packets. There would be a trade-off between the hardware and the response-time for detection of Proxy Traffic.

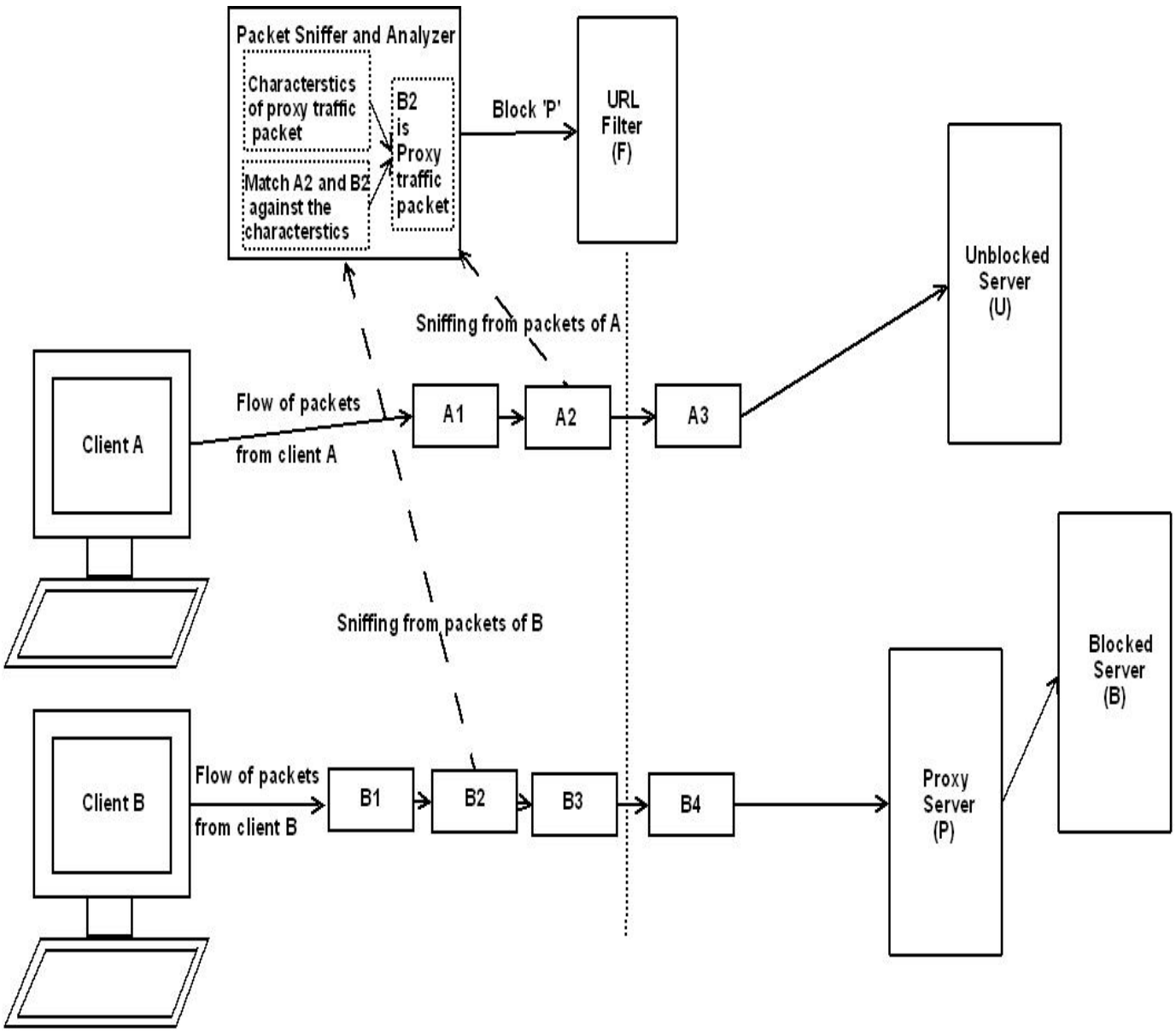


Fig. 2 : Packet sniffing to stop Proxy traffic

3. There would always be a gap between commencement and detection of Proxy Traffic.
4. Further there would always be a hit-ratio of the number of scenarios in which proxy traffic is being detected.
5. A change in heuristic would be necessary to accommodate any change in the characteristics of certain segment of Proxy traffic.
6. Custom proxy applications developed would pass unnoticed since those would not be able for examination.

D. Finding and Blocking the server IP addresses

The approach assumes the existence of Proxy traffic and takes measure to curb it. This application is based on current flaw of the client side executable proxies, to communicate with a number of Proxy servers as soon as they are executed. The exchange of the packets in this phase can be monitored and corresponding IP addresses can be blocked using 'URL filter'. The implementation is as follows. Fig.3 depicts the method.

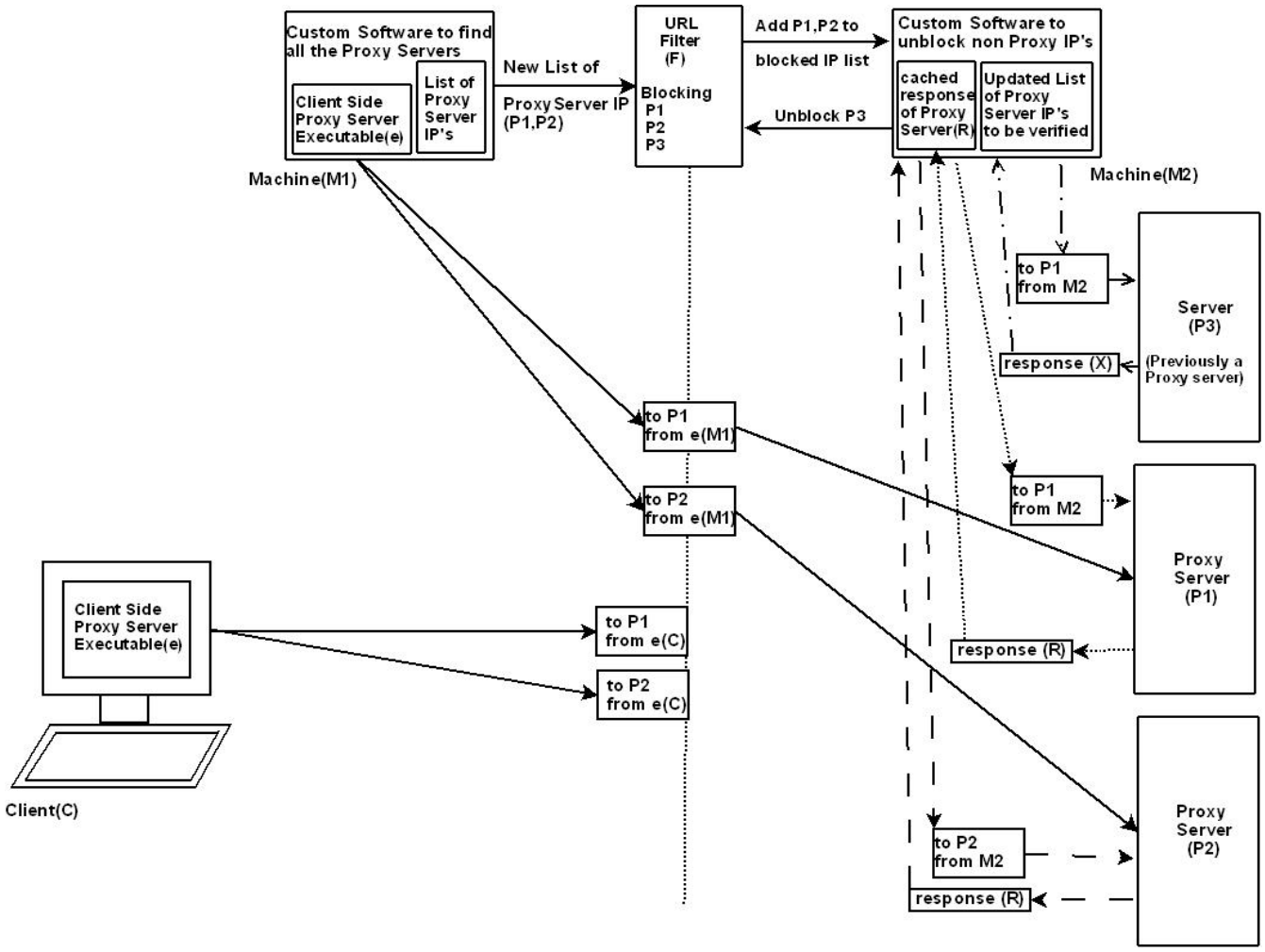


Fig. 3 : Finding and Blocking Proxy Servers IP addresses

We have one machine running each available client side proxy executable separately. When the executable is launched all the corresponding network packets are scanned for the destination address. This demands that the executables should be run one at a time and there should be no other application communicating over internet from the machine, so that packets do not get mixed. We can extract all the destination addresses from the packets using software like Wireshark, Ettercap. This list of IP addresses can be communicated to URL filter which then blocks any packet to these addresses. Again the executable is launched to see if it can still communicate and the process repeated till we get a list of all IP addresses and the executable stops working. This process needs to be carried out periodically since dynamic IP proxies keep on changing server IP's.

To ensure that an IP address that has been identified as Proxy server IP address is unblocked after server switches from it a small addition to the set-up is needed. This solution is based on the observation that when an IP address is being used by Proxy server than accessing that IP directly gives a similar webpage in return, else 'IP not found' or other error or even a different web page comes. We now need, in addition to the above mentioned, a machine external to the URL filter but in the organization's gateway. This machine will have the list of all IP addresses that have been sent to URL filter and are still blocked. It will try to send a 'HTTP get request', identical to the one a web browser sends. If it does not get a similar web page, as put in its cache as to be returned by that Proxy's Server, in return but some error status message or other web page, it unblocks the IP address and removes it from the above mentioned list. This process too needs to be done periodically. Lesser the length of period, earlier is the unblocking of that IP address. The period can be made as long as someone complains of being not able to access a website, that is not supposed to have been blocked.

Advantages

1. The idea, approach is based on, is simple and setting up the required machines is easy.
2. The creation of software which can extract IP addresses from sniffing software is easy.
3. Further most URL filter can take the list of IP addresses to be blocked in known formats like flat files. Thus communicating the list of IP addresses would be very easy.
4. Because whenever a proxy would be launched it will need to communicate with its Proxy servers initially this approach would be foolproof.
5. Further even if proxies try to misguide the application by communicating with lot of IP addresses, the additional set-up mentioned will overcome the problem of blocking permitted sites. The cost incurred is very less.

Disadvantages

1. All the existing client side executable proxy need to be monitored. Thus we need to add any proxy to the server as soon it is being used. This may not be feasible in the long run.
2. A custom proxy that is created by a group of individuals for custom use may not be available for examination and hence that group can continue accessing blocked sites.

E. Client Server Application

This is the most foolproof approach. This is based on the fact that for client side proxy executables to work they must run as a separate process in the machine and that their integration with any browsers or other internet accessing software like IM's will still make machine depict them as a separate executable in action. This can only be avoided by creating a new executable of the existing software, which in turn can be identified. Both of the cases and present client side proxy executables have a similar weak point – At the client level this executable can be identified. Thus we can determine what set of executables may run on a client that can access the internet. Any other application outside the list will not be allowed to access the internet. Based on this notion following set up was devised. Fig. 4 can be looked up for seeing the approach in action.

It requires that each client runs an application in order to access the internet and a server application on the central gateway to ensure the same.

It is only on client machine that we can come to know which executables are sending packet over the internet. So we make client run an application which will monitor what applications are sending the packets to the internet. This can be easily done by monitoring the network stack of the operating system. The application will have a list of executables that are allowed to access the internet. This list can be modified only by an administrator resulting in a new version of application, which the client will need to download to access the additional set of executables. The list can also be made to be downloaded on internet as the application is communicating over internet. So far we ensure that no proxy client side executable may be running as long as our application is being run by the client.

To ensure that the client is running the application the central gateway, which administers all the network connections from within the organization, will also have a complementary application that will check for a token being sent from every client after a fixed time or number of packets. This token would be sent by the application installed on the client machine to confirm that it is in operation. As soon as the gateway finds that a client has defaulted on the token it will terminate all its internet connection. The client will need to start the application. When the client will start the application there would be handshake between the application and server, which again can be exchange of variable number of tokens. The token would be removed from the packet stream by the gateway. The token could be a packet with a random number that can be changed periodically by mutual agreement of gateway and application.

Advantages

1. The set up is completely fool proof as long as client side executable can be disguised as a process of some allowed executables like web browsers.
2. The proxy traffic can be detected as rapid, as more is the frequency of the tokens being sent by the client side application.
3. Any custom proxy server application will also be prevented from execution. This is the unique and most important advantage to it.

Disadvantages

1. The set up requires a central server capable of processing a huge volume of data. Hence the cost of hardware.
2. Further insertion and withdrawal of tokens from the packet stream will slow down the internet speed.

IV. CONCLUSION

In this paper we introduced the blocking of Proxy Server used to access censored contents on Internet. We have given the underlying concept of its working. In the paper we provided a classification for the Proxy servers. Further the potential threats in the usage of such proxies have been discussed on the basis of the classification.

We presented two well known standard solutions to stop proxy websites and non-transparent proxies. On the basis of our understating of the available proxies, three custom solutions to stop the proxy traffic have been proposed. These solutions provide trade-offs between extensibility and effectiveness. When either of them is used with the two well known solutions, the combination can curb the Proxy traffic more effectively. The first solution namely packet sniffing, is widely under research in recent times to detect the Proxy traffic. This method has capability for extension and can be customized as the Proxy traffic evolves over time but at the same time has issues of accuracy in detecting Proxy traffic and privacy concerns. The second solution of blocking the server IP addresses came up during our effort on

understating of the working of proxies. We successfully tracked down all the IP addresses and concluded that we

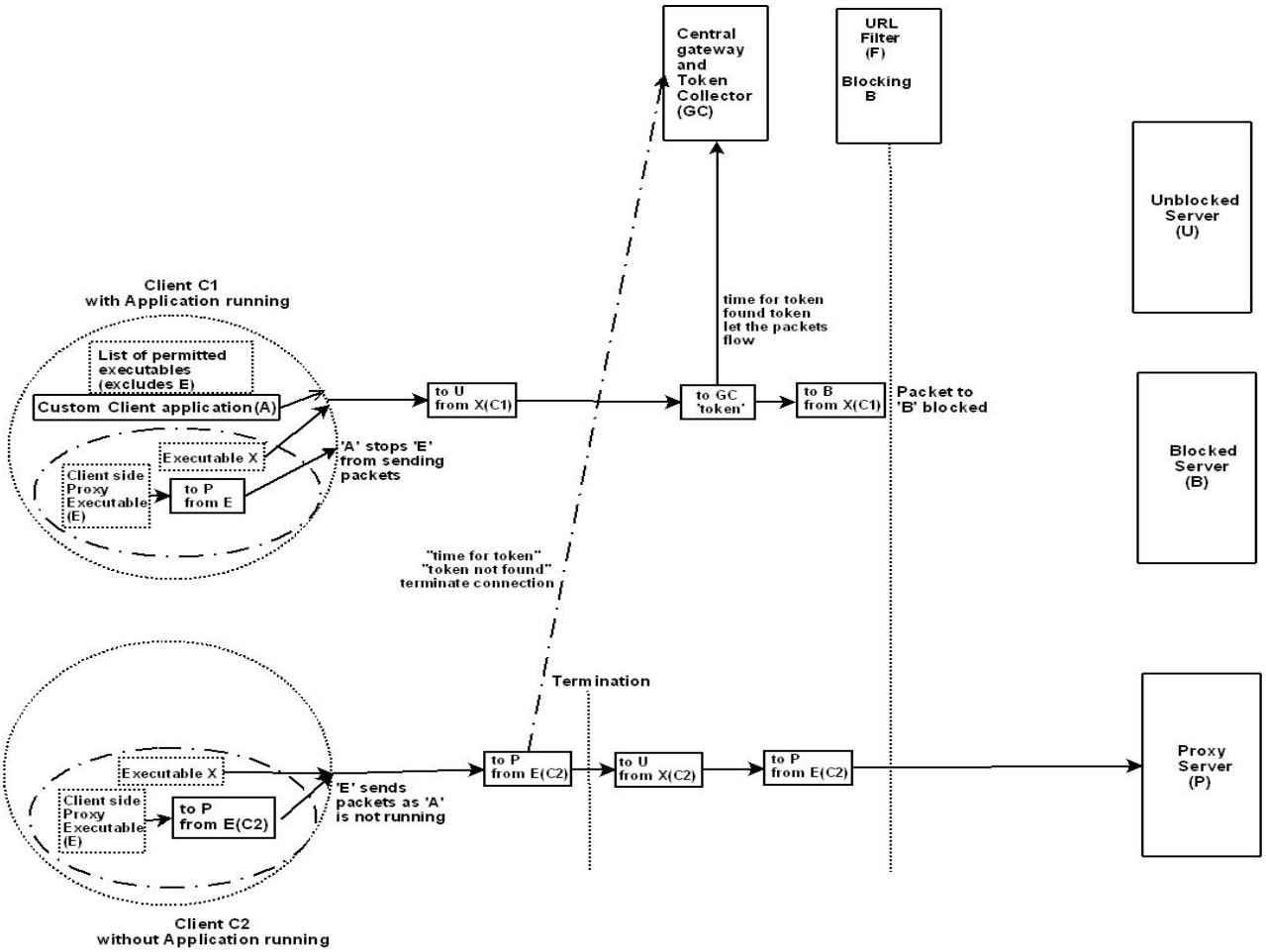


Fig. 4 : Client Server Application

were able to stop proxy by blocking those IP addresses. Thus the second solution is foolproof in present scenario but a change in approach of proxies may render it useless.

Both these solutions are incapable of dealing with private Proxy servers, the means of using which are not publicly available. Thus they rendered ineffective. The final proposed solution address all the problems with a little server overhead. It allows only a selective set of applications to connect to the Internet. It is one of the most foolproof of the three proposals. The present paper is largely work in progress, and we expect that by the time of the workshop we will have implementation results about the comparative data study of our proposed methods.

REFERENCES

[1] Hamade, S.N., "Internet Filtering and Censorship, " *Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference on,* vol., no., pp.1081-1086, 7-9April 2008
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4492629&isnumber=4492438

[2] Abiona, O.O.; Anjali, T.; Onime, C.E.; Kehinde, L.O., "Proxy server experiment and the changing nature of the web," *Electro/Information Technology, 2008. EIT 2008. IEEE International Conference on* , vol., no., pp.242-245, 18-20 May 2008
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4554305&isnumber=4554253

Survey on Malware Detection Methods

Vinod P.
Department of Computer Engineering,
Malaviya National Institute of Technology,
Jaipur, Rajasthan
e-mail: vinod_p22@yahoo.com

V.Laxmi,M.S.Gaur
Department of Computer Engineering,
Malaviya National Institute of Technology,
Jaipur, Rajasthan
e-mail: {vlaxmi|gaurms}@mnit.ac.in

***Abstract*— Malwares are malignant software's .It is designed to damage computer systems without the knowledge of the owner using the system. Software's from reputable vendors also contain malicious code that affects the system or leaks information's to remote servers.Malware's includes computer viruses, spyware, dihonest ad-ware,rootkits,Trojans,dialers etc. The paper focuses on various Malware detection methods like signature based detection, reverse engineering of obfuscated code, to detect malicious nature.**

***Keywords*—** Malware, obfuscation, Malware normalizer, reverse engineering

I. INTRODUCTION

Malware is a collective term for any malicious software which enters system without authorization of user of the system. The term is created from merging the words 'malicious' and 'software'. Malware is a very big threat in today's computing world. It continues to grow in volume and evolve in complexity. As more and more organizations try to address the problem, the number of websites distributing the malware is increasing at an alarming rate and is getting out of control. Most of the malware enters the system while downloading files over Internet. Once the malicious software finds its way into the system, it scans for vulnerabilities of operating system and perform unintended actions on the system finally slowing down the performance of the system.

Malware has ability to infect other executable code, data/system files, boot partitions of drives, and create excessive traffic on network leading to denial of service. When user executes the infected file; it becomes resident in memory and infect any other file executed afterwards. If operating system has a vulnerability, malware can also take control of system and infect other systems on network. Such malicious programs (virus is more popular term) are also known as parasites and adversely affect the performance of machine generally resulting in slow-down.

Some malware are very easy to detect and remove through antivirus software. These antivirus software maintains a repository of virus signatures i.e., binary pattern characteristic of malicious code. Files suspected to be infected are checked for presence of any virus signatures. This method of detection worked well until the malware writer started writing polymorphic and metamorphic malware. These variant of malware avoid detection through use of encryption techniques to thwart signature based detection.

Security products such as virus scanners look for characteristics byte sequence (signature) to identify malicious code. The quality of the detector is determined by the techniques employed for detection. A good malware detection technique must be able to identify malicious code that is hidden or embedded in the original program and should have some capability for detection of yet unknown malware. Commercial virus scanners have very low resilience to new attacks because malware writers continuously make use of new obfuscation methods so that the malware could evade detections.

This paper is organised as follows. Section II briefly describes various types of malware. Section III is a review of malware detector .Section IV reviews malware detection methods explains structural and functional aspects of polymorphic and metamorphic malware. Section V explains obfuscation and transformation techniques used by malware to avoid detection. Section VI explains malware similarity analysis method .Section VII reviews malware normalization with concluding remarks in Section VIII.

II. Malware Types

Malware can be broadly classified into following categories.

A. Viruses

Computer virus refers to a small program with harmful intent and has ability to replicate self. Mode of operation is through appending virus code to an executable file. When file is run, virus code gets executed. The original virus may evolve into new variants by modifying itself as in case of metamorphic viruses. A virus may spread from an infected computer to other through network or corrupted media such as floppy disks, USB drives. Viruses have targeted binary executable file (such as .COM and .EXE files in MSDOS , PE files in Windows etc.), boot records and/or partition table of floppy disks and hard disk, general purpose script files, documents that contains macros, registry entries in Windows, buffer overflow, format string etc.

B. Worms

Worms are self replicating programs. It uses network to send copies of itself to other systems invisibly without user authorization. Worms may cause harm to network by consuming the bandwidth. Unlike virus the worms do not need the support of any file. It might delete files, encrypt files in as crypto viral extortion attack or send junk email. Example Sasser, My Doom, Blaster, Melissa etc.

C. Spyware

Spyware is a collective term for software which monitors and gathers personal information about the user like the pages frequently visited, email address, credit card number, key pressed by user etc. It generally enters a system when free or trial software is downloaded.

D. Adware

Adware or advertising-supported software automatically plays, displays, or downloads advertisements to a computer after malicious software is installed or application is used. This piece of code is generally embedded into free software. The problem is, many developers abuse ad – supported software by monitoring Internet users' activities .The most common adware programs are free games, peer-to-peer clients like KaZaa, BearShare etc.

E. Trojans

Trojan horses emulate behavior of an authentic program such as login shell and hijacks user password to gain control of system remotely. Other malicious activities may include monitoring of system, damages system resources such as files or disk data, denies specific services.

F. Botnet

A botnet is remotely controlled software – collection of autonomous software robots. It is usually a zombie program (Worms, Trojans) under common control on public and private network infrastructure. Botnets are usually used to send spam /spyware remotely. Bot doesn't sit around on machine (infected machine) waiting for the instruction from a third party instead it looks for the communication with similar instances of bots awaiting instructions. Simplest bot configuration is where the bots are connected to single central hub. This configuration does not scale much because maintenance of various connections over single server is difficult. The next configuration is hierarchical structure where bot master connects to hundreds of bots which in turn is connected to many bots. Thus this configuration would scale much larger extent.

III.Malware Detector

A Malware detector 'D' is defined as a function whose domain and range are the set of executable program 'P' and the set {malicious, bening} [1].In other words malware detector can be defined as shown below.

$$D(p) = \begin{cases} \text{malicious} & \text{if p contains malicious code} \\ \text{benign} & \text{otherwise.} \end{cases}$$

The detector scans the program 'p' ε P to test whether a program is benign program or malicious program. The goal of testing is to find out false positive, false negative, hit ratio. The malware detector detects the malware based on signatures of malware.The binary pattern of the machine code of a particular virus is called as signature. Antivirus programs compare their database of virus signatures with the files on the hard disk and removable media (including the boot sectors of the disks) as well as within RAM. The antivirus vendor updates the signatures frequently and makes them available to customers via the Web.

a) False positive:

A false positive occurs when a virus scanner erroneously detects a 'virus' in a non-infected file. False positives result when the signature used to detect a particular virus is not unique to the virus - i.e. the same signature appears in legitimate, non-infected software.

b) False negative:

A false negative occurs when a virus scanner fails to detect a virus in an infected file. The antivirus scanner may fail to detect the virus because the virus is new and no signature is yet available, or it may fail to detect because of configuration settings or even faulty signatures.

c) Hit ratio:

A hit ratio occurs when a malware detector detects the malware.This happen because the signature of malware matches with the signatures stored in the signature databases.

IV. Malware Detection Techniques

Techniques used for malware detection can be broadly classified into two categories: anomaly-based detection and signature-based detection. An anomaly based detection techniques uses the knowledge of what is considered as normal to find out what actually is malicious .A special type of anomaly based detection is specification-based detection .Specification based detection makes use of certain rule set of what is considered as normal in order to decide the maliciousness of the program violating the predefined rule set. Thus programs violating the rule set are considered as malicious program. Signature based detection uses the knowledge of what is considered as malicious to fins out the maliciousness of the program under inspection.

IV (A) Signature-Based Malware Detection Techniques

Commercial antivirus scanners look for signatures which are typically a sequence of bytes within the malware code to declare that the program scanned is malicious in nature. Basically there are three kinds of malwares: basic, polymorphic, metamorphic malwares. In basic malware the program entry point is changed such that the control is transferred to malicious payload. Detection is relatively if the signature can be found for the viral code. Figure 1 show basic malware.

Fig.1 Basic kind of virus

Polymorphic viruses mutates while keeping the original code intact. A polymorphic malware consists of encrypted malicious code along with the decryption module. To enable the polymorphic virus the virus has got polymorphic engine somewhere in the virus body. The polymorphic engine generates new mutants each time it is executed. Signature based detection for such a virus is difficult because each variant new signature is generated which makes signatures based detection difficult. Strong static analysis based on API sequencing is used for polymorphic virus detection [9].Figure 2 shows polymorphic malware's.

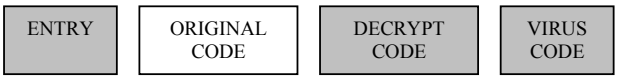


Fig.2 Polymorphic virus

Metamorphic malware can reprogram itself using certain obfuscation techniques so that the children never look like the parent [4]. Such malwares evade the detections from the malware detector since each new variant generated will have different signature, hence it is impossible to store the signatures of multiple variants of same malware sample. In order to thwart detection a metamorphic engine has to be implemented with some sort of disassembler in order to parse the input code. After disassembly, the engine will transform the program code and will produce new code that will retain its functionality and would still look different from the original code Figure 3 shows metamorphic malware and multiple signatures for multiple variants.

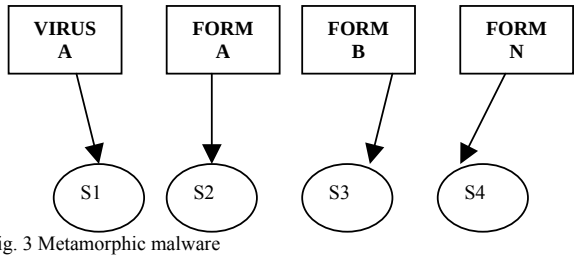


Fig. 3 Metamorphic malware

Let 'S' be the set of malware signatures. For the above figure - 3, $S_i \in S$ are signatures of metamorphic variant belonging to single metamorphic sample 'M'.

The main problems with the signature based detection method is as follows:

- Signature extraction and distribution is a complex task.
- The signature generation involves manual intervention and requires strict code analysis.
- The signatures can be easily bypassed as and when new signatures are created.
- The size of signature repository keeps on growing at an alarming rate.

IV (B) Specification-based Detection

Specification-based detection is the derivate of anomaly-based detection. Instead of approximating the implementation of a system or application, specification-based detection approximates the requirements of application or system. In specification-based system there exists a training phase which attempts to learn the all valid behaviour of a program or system which needs to inspected. The main limitation of specification based system is that it if very difficult to accurately specify the behaviour the system or program. One such tool is Panorama which captures the system wide information flow of the program under inspection over a system, and checks the behaviour against a valid set of rule to detect malicious activity [2, 3].

IV (C) Behaviour -based Detection

Behaviour based detection differs from the surface scanning method in that it identifies the action performed malware rather than the binary pattern. The programs with dissimilar syntax's but having same behaviour are collected, thus this single behaviour signature can identify various samples of malware. This types of detection mechanisms helps in detecting the malwares which keeps on generating new mutants since they will always use the system resources and services in the similar manner. The behaviour detector basically consists of following components which are as follows:

- *Data Collection:* This component collects the dynamic / static information's are captured.
- *Interpretation:* This component converts the raw information collected by data collection module into intermediate representations.
- *Matching Algorithm:* It is used to compare the representation with the behaviour signature.

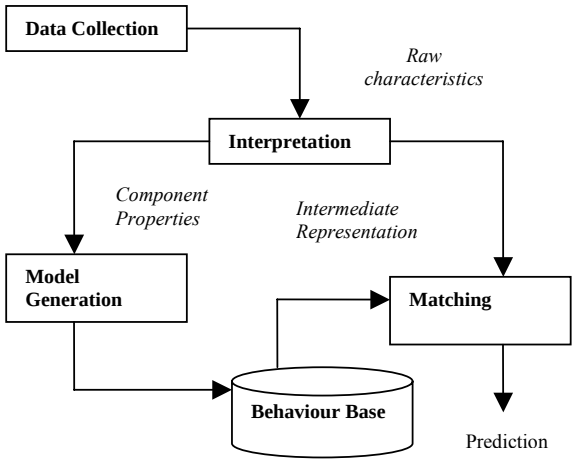


Fig. 4 Behaviour Detector [14]

V. Obfuscation

Obfuscation is to hide the information such that others cannot find the true meaning. Software vendors make use of obfuscation so that the software would be difficult to reverse-engineer. Malware writers take it as advantage and obfuscate the malicious program using various obfuscation transformations so that the Malware is difficult to reverse-engineer and hence its malicious intent cannot be learned.

V (A) Obfuscation theory

Given a program *P* and a transformation function *T* generates program *P'* such that the following properties holds true:

- *P'* is difficult to reverse engineer.
- *P'* holds the functionality of *P*.
- *P'* performs comparable to *P*

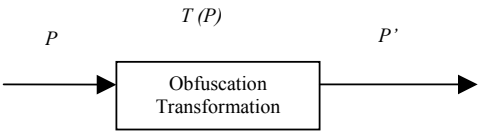


Fig. 5. Obfuscation

Many metamorphic and polymorphic make use obfuscation techniques so that they can defeat the signature based detection .Obfuscation techniques can easily change the signatures of Malware. Let us first look at some example of obfuscation technique modifying the signature of the code given below.

Original Code

Hex Opcodes	**Assembly**
51	push ecx
50	push eax
5B	pop ebx
8D 4B 38	lea ecx,[ebx+38h]
50	push eax
E8 00000000	call 0h
5B	pop ebx
83 C3 1C	add ebx,1Ch
.	
.	

.

Signature

5150 5B8D 4B38 50E8 0000 0000 5B83 C31C

Now suppose the original code is obfuscated by inserting a bunch of junk instruction like *nops.* Then the obfuscated code and the new signature is as follows:

Original Code

Hex Opcodes	**Assembly**
51	push ecx
90	**nop**
50	push eax
5B	pop ebx
8D 4B 38	lea ecx,[ebx+38h]
50	push eax
90	**nop**
E8 00000000	call 0h
5B	pop ebx
83 C3 1C	add ebx,1Ch

Signature

5190 505B 8D4B 3850 90E8 0000 0000 5B83 C31C

Thus the change in signature is not detected by Malware scanner and the false negative rate will increase enormously. Common obfuscation techniques fall into following main categories:

a) Dead-code-insertion
b) Code transportation
c) Register Renaming
d) Instruction Substitution

V (B) Dead-code-insertion

The example shown above falls into the first category, i.e. dead-code-insertion. This is can be done by either inserting bunch of *nops* (that does not accomplish anything), or inserting some number of *push x* followed by *pop x*, where *x* refers to register.

V(C) Code Transportation

Code transportation is done by inserting jump instruction or unconditional branch instructions so that the original control flow of the program is maintained. Other techniques such that swapping instructions which are not interdependent also exist.

Using the same example in section 4, this technique would yield a code that would look as follows:

```
        push ecx
        push eax
        jmp A
A:      pop ebx
        lea ecx,[ebx+38h]
        jmp B
C:      pop ebx
        add ebx,1Ch
B:      push eax
        call 0h
        jmp C
```

V (D) Register Renaming

This technique replaces the use of a register in an instruction with another unused instruction. Register replacement requires that no register dependencies in control flow are affected.

Original Code

```
Mov eax,32
Mov ecx,1024
Push eax
Sub eax,eax
Push eax
Mov eax,ecx
Add eax,2
Mov [eax+2],ebx
Pop eax
```

Transformed Code

```
Mov eax,32
Mov ebx,1024
Push eax
Sub eax,eax
Push eax
Mov eax,ebx
Add eax,2
Mov [eax+2],ebx
Pop eax
```

V (E) Instruction Substitution

A sequence of instructions is associated to a set of alternative sequence of instructions which are semantically equivalent to the original one. Every sequence of original instructions can be replaced by some arbitrary instructions. For example the following code is equivalent.

Original	**Transformed**

MOV reg,imm	a)MOV reg,Random
	ADD reg,imm-Random
	b)MOV reg,Random
	SUB reg,-(imm-Random)

MOV eax,eax	XOR eax,eax

Obfuscated Malware can be detected by collecting multiple variants of same malware sample and performing similarity analysis between the variants or generating the normalized code.

VI. Similarity Analysis

The known sample of malware such as Win32.Evol [4] is taken and multiple files of the same sample is generated by obfuscating the known sample. Similarities between the files are checked to find whether the variant is the child of the sample under inspection. Similarity analysis using Euclidean normal form [12] can be used to find the distance between some vectors *x* and *y* as:

$$D = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

A program is represented as some number of functions *f,* and each function contains some number of statements which are termed as vectors *x* and *y*. The total number of vectors for the same program *P* and for all function *f* is kept same. Similarity analysis can be performed by using cosine similarity measure [6] which is primarily used in text mining. SAVE tool uses the average of various similarity measures to estimate maliciousness of the program [7].

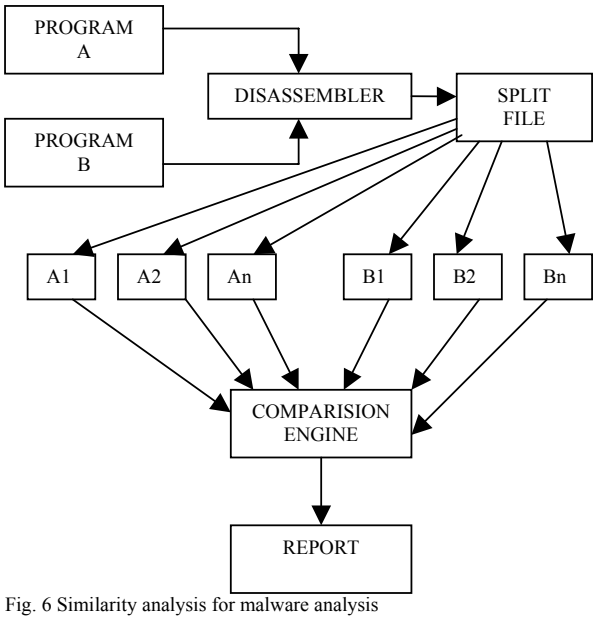


Fig. 6 Similarity analysis for malware analysis

Following steps are adopted to detect malicious nature of program.

Step1: Program executable is decompressed (optional) if the program is compressed.

Step2: Decompressed program is disassembled using the disassembler like IDA PRO [10] or OllyDbg[11].

Step3: Each disassembled program is represented as a vector of functions. Each function is represented as array of equal length.

Step 4: The similarities between the functions of program *P'* and *P''* is computed using Euclidean normal form or cosine similarity measure or Pearson's similarity measure or Jaccard similarity measure.

Step 5: The value of the similarity is compared with the threshold value; if the value is very less than the threshold value then the program under inspection is benign otherwise malicious.

Note the choice of similarity is crucial. A high value of threshold increases the risk of false negative and low value increases the risk of false positiveness.

VII. Malware Normalization

A Malware normalizer accepts the obfuscated version of Malware and eliminates the obfuscation carried on the program and produces the normalized executables .Thus it can be said that the Malware normalizer increases the detection rate of the detector. Malware *M* is taken and passed through the normalizer. After normalization the signature of this Malware is extracted and compared with the signatures of canonical form [8]. Maximum length of matching signature of canonical form with the Malware is considered and the new signature of the canonical form is stored in the signature database for future comparisons. Let us consider a Malware *'M'* and $c_i \varepsilon C$ is canonical form of the malware M.

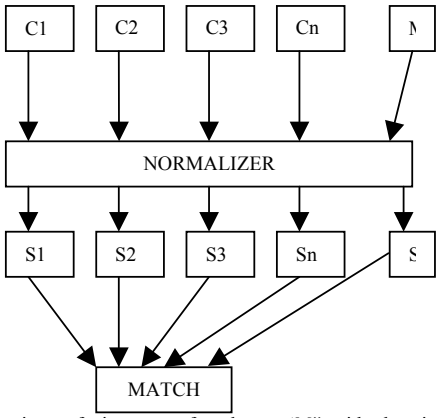


Fig. 7 Comparison of signature of malware *'M"* with the signatures of canonical forms c_i *'s*.

Following are steps involved in malware detection using malware normalizer:

Step 1: Malware PE binary code is decompressed (optional) using decompression software.

Step 2: The decompressed code obtained from above step is disassembled using the standard disassemblers.

Step 3: The disassembled code is then passed through the normalizer. The normalizer check for obfuscation performed eliminates obfuscation and produces normalized code.

Step 4: The normalized code is passed to the malware detector which extracts the signature of the normalized code, compares it with the signature stored in signature repository.

Step 5: The comparison is based on maximum match of the signature stored in the repository with the signature normalized code. For matching any sequence alignment algorithm can be adopted. Finally the signature of normalized code is stored in database for future signature comparisons with other variants.

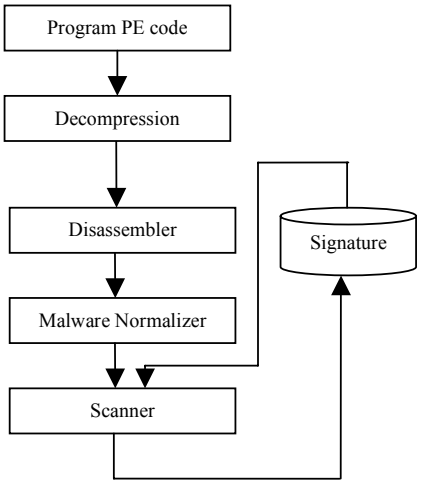


Fig.8. Malware Normalization and signature comparison [13]

Malware signatures are long and similarity analysis between signatures of different samples takes more number of comparisons. Thus there exists need of shorter signatures. Similarity analysis based on API sequence is used to detect polymorphic variants [9].The suspicious PE file is optionally

decompressed. Then it is passed through the PE parser. The output of PE binary parser is a sequence of API calls. This API sequence is 32 bit id. The first 16bit represents a API module and last 16 bit represents specific API within an API module. The API sequence call is compared with the sequence of calls of known malware. Then a similarity between these API's sequence is performed, which is compared against threshold value .If the value is lesser than threshold value then the sample under inspection is reported as benign otherwise malicious.

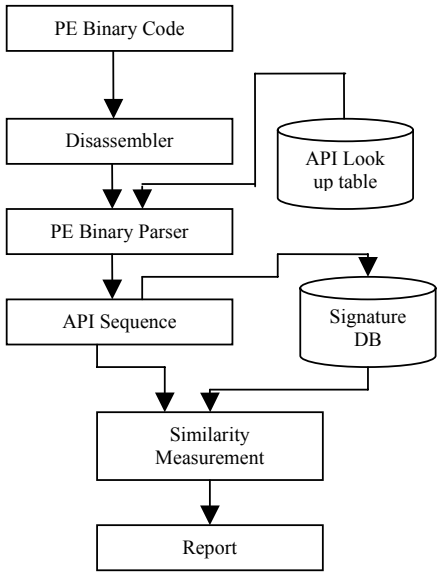


Fig.9 Similarity measure based on API sequence

VIII. Conclusion

In this survey a series of malware detection techniques have been presented. The problems related to traditional signature based detection method is also highlighted. Rate of new malware's causing destructions to systems world wide is increasing at alarming rate. Detection of malware's changing their signatures frequently has posed many open research issues. Challenge lies in the development of good disassembler , similarity analysis algorithm so that the variants of malware's can be detected in shorter time there by reducing the computation overhead.

References

[1] Mihai Christodorescu and Somesh Jha ," Testing Malware Detectors", in Proc. ISSTA'04, July 11 - 14, 2004.pages 33-44, Boston, MA USA, ACM Press.

[2] Heng Yin ,Dawn Song ,Manuel Egele,Christopher Krugel , and Engin Kirda , "Panorama: Capturing System – wide Information Flow for Malware Detection and Analysis", in Proc CCS'07, October 29 – November 2, 2007, Alexandria, Virginia, USA,ACM Press.

[3] Andreas Moser , Christopher Krugel , and Engin Kirda, "Exploring Multiple Execution Paths for Malware Analysis", Secure Systems Lab, Technical University Vienna.

[4] Arun Lakhotia , Aditya Kapoor , Eric Uday , "Are Metamorphic Viruses Really Invincible ? Part 2" , Virus Bulletin ,January 2005.

[5] Abhishek Karnik , Suchandra Goswami and Ratan Guha," Detecting Obfuscated Viruses Using Cosine Similarity Analysis", *in The Proceeding of IEEE Symposium First International Conference on Modelling & Simulation (ASM '07)*

[6] A.H.Sung, J.Xu, P.Chavez, S.Mukkamala,"Static Analyzer of Vicious Executables (SAVE)", Proceeding of the 20th Annual Computer Security Applications Conference (ACSAC 2004), 2004.

[7] Mihai Christodorescu , Johannes Kinder , Somesh Jha , Stefan Katzenbeisser , Helmut Veith , "Malware Normalization ", University of Wisconsin and Madison Technische Universit at M unchen , 2005.

[8] J.Xu, A.H.Sung, P.Chavez, S.Mukkamala,"Polymorphic Malicious Executable Scanner by API Sequence Analysis", Proceeding of 4th IEEE Symposium of International Conference on Hybrid Intelligent Systems (HIS '04).

[9] DataRescue, Inc. The ida pro disassembler www.datarescue.com/idabase,2006

[10] O.Yuschuk. 80x86 32 bit disassembler and assembler.www.ollydbg.de/srcdescr.htm,2006

[11] Mohamed R.Chouchane and Arun Lakhotia,"Using Engine Signature to Detect Metamorphic Malware", In Proc. Worm06 ,November3,2006, Alexandria, Virgina, USA, ACM Press.

[12] Mohamed R.Chouchane, Andrew Walenstein and Arun Lakhotia," Statistical Signature for Fast Filtering of Instruction-substituting Metamorphic Malware", In Proc. Worm07, November 2, 2007", Alexandria, Virgina, USA, ACM Press.

[13] Ran Jin,Qiang Wei,Pei Yang and Qingxian Wang ,"Normalization towards Instruction Substitution Metamorphism Based on Standard Instruction Set", In Proc. IEEE symposium on 2007 International Conference on Computational Intelligence and Security Workshops.

[14] Greoigre Jacob,Herve Debar,Eric Fillol,"Behavioral detection of malware:from a survey towards an established taxonomy",Springer-Verlag France 2008

[15] Gerard Wagener,Radu State,Alexandre Dulaunoy,"Malware Behavior Analysis",Springer-Verlag France 2007.