Reverse Engineering for Malicious Code Behavior Analysis using Virtual Security Patching

A.EdwinRobert[1], G.Manivasagam[2]
[1]ResearchScholar, [2]Lecturer,
KarpagamUniversity,
Pollachi Main Road,
Coimbatore

N.Sasirekha
ResearchScholar,
KarpagamUniversity,
Pollachi Main Road,
Coimbatore

Dr.M.Hemalatha
Head,Departmentof Software Systems
KarpagamUniversity,
Pollachi Main Road,
Coimbatore

ABSTRACT
Computer hardware and Internet is growing so fast today, security threats of malicious executable code are getting more serious. Basically, malicious executable codes are categorized into three kinds – virus, Spam, Trojan horse, and worm. Current anti-virus products cannot detect all the malicious codes, especially for those unseen, polymorphism malicious executable codes[1]. The newly developed virus will create the damages before it has been found and updated in database. Spy ware is becoming a real concern [2]. The proposed architecture classifies the behavior of the new type virus and it identifies the malicious code through the virtual server, where all the unwanted code executions and dependable are get refined first before it reaches the actual server. This phenomenon is known as virtual engineering. The security features in the virtual server get processed virtually through reverse engineering technique [3]. Here the user or the administrator checks the application first automatically in the virtual server and it analyze the behavior and filters the malicious code and protects the actual server, this process is very fast compare to other architecture which we have noticed in emerging operating systems.

General Terms
Virtual Security, Patching,Virtual Engineering and Virtual Process

Keywords
Function Extraction Technology, Malicious code, Patching, Reverse Engineering, Security threats, Virtual Engineering.

1. INTRODUCTION
Within this paper, security flaws have been identified in the software applications in the industry. In the software market where no specific control is applied to applications submitted by developers. This behavior is one of the consequences of the open source nature of the software applications and is a platform for spy ware spreading like the one discussed in this paper. As software applications in the industry becomes more popular, the number of visitors to the software market is likely to rise, and software checks may become necessary to maintain the platform's trustworthiness. More granularities for application permission might be a solution, for example by not authorizing full Internet access but rather access to specific web site(s). The Anti-virus benchmark pointed out that there is no silver bullet to prevent spy ware, user awareness still remains the best way to prevent spy ware installation.

1.1 Security Flaws
An Error of commission or omission in a Software Application (SA) may allow protection mechanisms to be bypassed[4]. Developer's awareness is also necessary, as several security flaws were discovered recently in banking applications that could take entire control of their devices, as all applications are running with root access.

1.2 Patching and Slow Patching Process
A Patch is a piece of software designed to fix problems with, or update a computer program or its supporting data[5]. This includes fixing security vulnerabilities and other bugs, and improving the usability or performance. Though meant to fix problems, poorly designed patches can sometimes introduce new problems.

Within the industrial chain is the slow patching process. The reason is that so many manufacturers distribute embedded software devices, when a security update is required it is up to the manufacturers to provide it to their customers. So software applications distribution requires more time than for competitors like Apple[6]. This slow patching process leads to a rise of Jailbreak devices, as users want access to the latest releases. As a consequence, the restrictions are completely bypassed and users are more exposed to spyware an application claiming to be the latest version of a famous Twitter client, which actually runs spyware in the background and uploads all private data to the attacker.

1.3 Malicious Web Application Risks & Exploitation
Since the patching process in the web applications in the industry is relatively slow, an attacker could use spyware as a process to escalate privilege on the phone by exploiting new problems by altering or eliminating vulnerabilities by controlling the inputs and outputs to and from the applications. A virtual process can be created to mitigate a given vulnerability through packet manipulation & proxies via brokering the protocols to the application in question. The virtual process itself is created through vulnerabilities discovered in the Linux kernel before the patch release. The Application and the slow patching process are responsible for much of the current threat landscape for software applications. In order to improve its overall security, better controls of the Software industrial Market as well as a better patching system are required.

2. PROPOSED ARCHITECTURE

The purpose of this paper will be to explore a guideline to the spy ware development using reverse engineering techniques and provide real case attack scenarios. The basic idea of the proposed system is, it will analyze the behavior of the malicious codes and based on the behavior signature of the malicious code the algorithm will detect the type of common viruses, worms, spam and Trojans in the virtual server and protect the actual database in the server. The Virtual server has the virtual security mechanism which filters out and protects the system from further execution of the spy wares.

2.1 Function Extraction Process

Function extraction (FX) is a disruptive new technology with potential to improve the economics of software development and increase the dependability of software systems [7]. The behavior of the code is analyzed using the function extraction technology.

3. REVERSE ENGINEERING

The activity of examining and copying a product developed by another company in order to make your own product through a legally sanctioned method of copying a technology which (as opposed to starting from scratch) begins with an existing product and works backward to figure out how it does what it does. When the product's basic principle or core concept is determined, the next step is to reproduce the same results by employing different mechanisms to avoid any (legally forbidden) patent infringement. Reverse engineering will be used, because most users do not check the permissions of the applications loaded onto their software applications. Even security professionals admit they do not often check permissions of their applications [8].

3.1 Virtual Process

A virtual process can be defined as the ability to do "vulnerability mitigation" means to reduce attacks and threats in a separate layer. By incorporating a "Virtual Processing" strategy, an organization can greatly improve efforts to reduce their organizational risk through quick remediation of vulnerabilities in web software[9] . As the web interface has become the ubiquitous interface to software, this paper will provide an overview of virtual processing web applications with a focus on the open source project. Through the adoption of Virtual processing as another tool in the information security arsenal, this article will illustrate how organizations can decrease the risk from software vulnerabilities and provide overall better defenses across their technology environments. In situations where traditional patches are not feasible, a virtual process can be utilized to reduce the likelihood of a successful attack.

3.2 Reducing Risk through Virtual Engineering

Problems in applications without having to touch the applications themselves or as a policy for an intermediary device Web Application Firewall (WAF) that is able to identify/block attempts to exploit a specific Website vulnerability. A virtual process deals with the process or method of fixing the rule language of the packet manipulator. This paper, focus on Virtual Engineering with Virtual Security, a popular and extremely versatile Open Source Web Application Firewall (WAF) used to create and apply custom virtual processes. The range and capabilities in Virtual Security are far reaching and in this paper we will focus specifically on Virtual processing approaches with Virtual Security.

4. FISH EYE SECURITY

The architecture of the environment for which packet manipulation can occur is key and a cornerstone to the virtual process success [10]. Using Fish Eye Security, several network arrangements exists to allow packet manipulation for a virtual engineering to be implemented.

4.1 Embedded Security

The first arrangement is to run Security embedded as a Fish Module on the server which is aimed to defend and apply virtual engineering concepts for the given application. In this case Fish Eye Security runs embedded with the applications which it is defending, as a Fish Module changes are needed to enable defense and virtual Engineering capabilities for the web applications.

5. REVERSE ENGINEERING

A dramatic increase of functionality, quantity, size, and complexity of software-intensive embedded systems in the automotive industry can be observed. In particular, the growing complexity drives current reverse engineering practices to the limits. The second architecture to run Fish Eye Security to enable defense and virtual engineering is to configure Security on a reverse proxy server running Fish [11]. In this case Security can defend and apply virtual Engineering to numerous applications, on numerous servers which it is tasked to oversee.
The purpose is to deduce design decisions from end products with little or no additional knowledge about the procedures involved in the original production. The same techniques are subsequently being researched for application to legacy software systems, not for industrial or defense ends, but rather to replace incorrect, incomplete, or otherwise unavailable documentation

5.1 Method

Reverse engineering may also be necessary if alternative methods of obtaining technical data are more costly than the actual reverse engineering process. Generally, many products are protected by copyrights and patents. Patents are the stronger protection against copying since they protect the ideas behind the functioning of a new product, whereas a copyright protects only its look and shape[12].

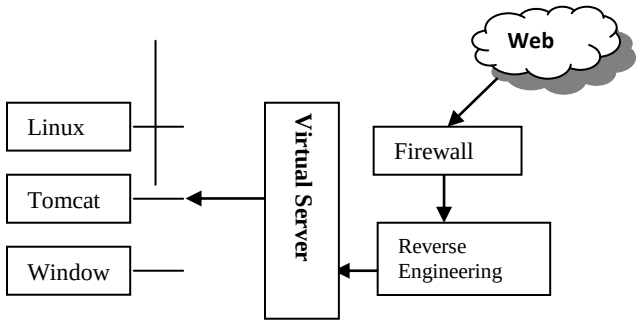

Fig 2. Reverse Engineering with Virtual Server

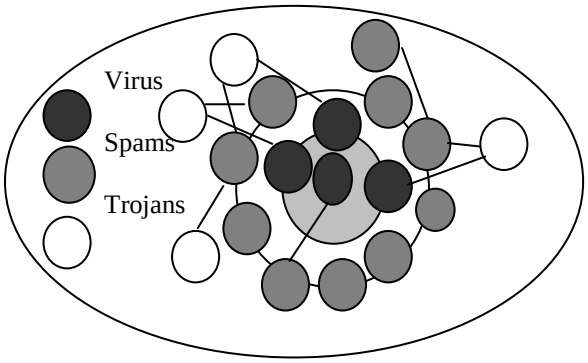

Fig 3. Scope of Fish Eye

6. REVERSE ENGINEERING WITH VIRTUAL SERVER

With this model many advantages exist as security separation layer exists and the defense is truly operating system and programming language diagnostic [13]. Fish Eye Security can be configured to protect various web applications running on a various operating systems, whether they are Linux, Windows, UNIX, Mac OS etc. – it matters not. An added benefit to this model is that administration efforts can occur with greater ease, as you're dealing with one technology stack to administer instead of splitting administration expertise across several O/S environments. A key and fundamental assumption to both architecture models is that your applications and security are tightly coupled. Virtual Security is tasked to maintain the integrity of the application, and proper application request filtering must be maintained with this defense approach.

6.1 Architecture Advantages

Deciding which are the right and the best architecture approach for your organization will truly depend on your business needs. Fish Eye Security on each individual server can be quickly stood up and ready to be brought online without requiring a change to the network, thus reducing the number of overall team members involved to implement. Also with this architecture, no new hardware is really needed to implement an embedded Security install. One of the downsides to running Security embedded is that while the Security application footprint is minimal, it will still share the computing resources of the server hosting the web application. The embedded architecture is the potential administration troubles that can quickly get out of hand in managing multiple servers with rule sets and configuration, as more web servers and web applications are introduced to the company's application lineup. This can be aided through scripted deployment and auto updating, but it does involve a bit more complexity to maintain a growing number of servers which may increase exponentially as the organization grows over the years.

Running Security on a reverse virtual server does have several business advantages as the expertise to maintain and update one instance of Security with virtual Engineering and configuration can be leveraged against many web application and servers which need to be defended and virtually patched. The organization can respond to taking other vulnerable applications "under the wing" fairly quickly, which may not have originally been identified as candidates needing crucial patches. The flexibility of this architecture approach shines since the applications that might need critical patches can quickly be arranged to be covered by Security for protection and virtual processing, even though the applications may be written in a variety of languages and running on different operating systems.

This architecture also works well for closed source proprietary web based vendor software, which your organization may run, but which the vendor may not still be in business or adequately capable of patching the software.

A few drawbacks to the reverse proxy server implementation include introducing a single point of failure if the reverse proxy fails or bottlenecks traffic. With the advances in virtualization and on demand resource balancing this may become less of an issue and drawback to this approach. In all architectures there are advantages and disadvantages. Ultimately, the decision boils down to whether your organization chooses to implement centralized (reverse proxy) virtual processing administration or a decentralized (embedded) virtual processing administration approach.

7. THE VULNERABLE WEB APPLICATION

With either architecture in place, the environment is set to deploy an in response to a reported vulnerability. To get an understanding of how a Virtual process can be applied to mitigate a real application vulnerability, we'll turn to a case where the code handling user logins of a company commerce portal is flawed, leaving the website susceptible to a SQL injection attack [14].

7.1 The Unauthorized Access

This vulnerability could be used by an attacker to gain unauthorized access to information or escalate his privileges in the system, gaining administrator access. Once the attacker has gained administrative privileges, the potential end result of the successful exploitation of this web application vulnerability could translate into significant dollar losses for the business as information stored in the application is up for grabs for those with mal intent. With this security compromise, the potential damage to the company's reputation and the loss of consumer confidence is great, as customer or financial data could be stolen and sold on the black market or company trade secrets lost.

8. FISH EYE STATE CODE BEHAVIOR ALGORITHM

Fish Eye State Code Behavior Algorithm (FECB) is a proactive or table driven filtering algorithm. FECB is based on the traditional link state routing algorithm. Each and every node collects the information about the structure of the vertices from the neighboring nodes and calculates the filtering table. It then disseminates the information locally to the neighboring nodes. The FECB differs from the traditional link state filtering algorithm in the way it disseminates the information across the neighboring nodes [15]. It reduces the overhead associated with updating paths by introducing the notion of multi-level fish eye scope. The scope of the fisheye has been given in figure 3. The frequency of exchanging the filtering information with neighbors depends on the distance between the source and the destination. From the link state entries the node calculates the optimal shortest routes to other nodes. FECB is simple, scalable and efficient in ad hoc communications.

8.1 Representation of Code Behavior in FECB

The network is represented as an undirected graph G= (V, E) where V = number of vertices or nodes in the network and E = number of edges or undirected links in the network[16]. Each node has a unique identifier which represents a host with a device with transmission range R, and an infinite storage space.

A link between two nodes i and j is formed when the distance between i and j becomes less than R. The link (i, j) is moved if distance between i and j exceeds the range R. In FECB, for each node i, one list and three tables are maintained.

(i) A Neighbor list Ai
(ii) A Structure table STi
(iii) A Next host table NEXTi
(iv) A Distance table Dti
Ai stores all the nodes those are neighbors to the node i. The structure table contains the most up to date information about the structure of the vertices from the link state message. The information in the structure table are required while calculating the filtering table. The structure table has three fields; destination address, destination sequence number, link state list[12] Any destination j in STi link state list has two parts STi.LS(j) which denotes the link state information reported by node j and STi.SEQ(j) indicates the time stamp at which j has generated the link state information. For each destination j, NEXTi(j) denotes the next hop to forward packets destined to j. Di(j) denotes the distance of the shortest path from i to j. A weight function can be used measure the distance of a link and is denoted by E-> Z0+ , which returns 1 if there is a direct link between two nodes , else, it returns ∞.

8.2 FECB ALGORITHM

The FECB algorithm is given below

Step 1: Initialize Array List Ai, Neighbour node Ni, Structure of node list STi, Next host NEXTi,Queue Qi,
Distance between two nodes Dti

Step 2 : if (Ai.Qi≠empty)
For Ai= 0 to Ai.Qi
{
Ai=Ai.Qi;
Ni=Ni +Ai.source; [Neighbour node]
STi.LS(j)=STi.LS(j) + Ai.source;
}
For j= 1 to V [Vertices]
{
//J records the transaction time and link state information - time stamp

if (j<>i) && (Ai.STM(j))>STi.STM(j))
{
STi.STM(j)=Ai.STM(j);
STi.LS(j)=Ai.LS(j);
}
}

Step 3 : for j= 1 to Ni
{
If weight (i,j)= ∞
Ni=Ni- j [source]
}

Step 4 : for x =1 to N
{
STi.LS(i) = STi.LS(i) + x [Source];
Spy.fileid =i;
}
for x Є N
{
for ScopeLevel l:= 1 to L do
if ((Clock() mod UpdateIntervall = 0)
^ (Di(x) Є FisheyeScopel))
then spy.STi = spy.STi + STi.LS(x); [source]
}
// Di(x) is calculated using
//Disjkstra's Shortest path algorithm

step 5 : publish(j,spy) to all j Є Ai;

9. FECB - BEHAVIORAL SYNTHESIS

Behavioral synthesis is an automated architecture process that interprets an algorithmic description of a desired behavior and creates hardware that implements that behavior. Starting with an algorithmic description in a high-level language, behavioral synthesis architectures automatically create the cycle-by-cycle detail needed for hardware implementation.

9.1 RTL Implementation

Most behavioral synthesis approaches leverage the existing logic synthesis architecture set by creating a register-transfer level (RTL) implementation from the algorithmic description. The architecture description made possible by a behavioral synthesis architecture flow differs in a number of specific ways from that which is required for traditional logic synthesis. Logic synthesis uses an RTL description of the architecture. Behavioral synthesis uses a high-level un-timed, or partially timed, functional description. These descriptions can contain large portions of algorithm that, after behavioral synthesis, will be spread over many clock cycles. These algorithms can, and typically do, contain loops and array accesses that are not typically seen in register-transfer level (RTL) architectures.

The behavioral synthesis architecture will figure out how best to schedule that over the multiple cycles in order to meet architecture constraints.It will also determine the data path, multiplexing and finite state machine needed to implement the architecture. Behavioral synthesis will interpret this as a Fish eye control-dataflow graph (FCDFG) which represents the dependencies of the various operations as shown in Figure 4.

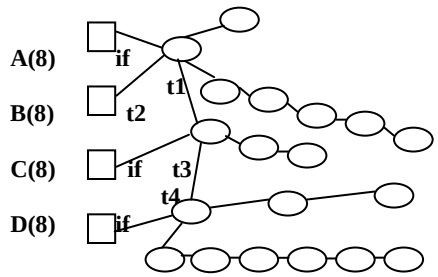

Fig 4 : Control Data Flow Graph

Given that CDFG and the list of available functional units (adders, multipliers, etc.) shown in Table 1, the behavioral synthesis architecture could choose to schedule the operations in several different ways, depending on the design goals, such as performance or area. In the given table,the first row represents four(4),Ten(10) conditions which we need eight (8) and twenty(20) functional units respectively. The execution time delay for the respective conditional statements using the time stamp function STi.LS(j) and it covers the respective functional unit area

Table 1: Functional Units

Functional Unit	*Delay*	*Area*
4.4 = 8	1.34	2433.2
10.10=20	2.34	1770.6

Assuming minimizing area is the design goal, the behavioral synthesis process select the schedule shown in Table 2. using the needed operator per cycle calculating for the Cycle1[source] and Cycle2[linkstate] and based on the control data flow graph[CDFG] it produces the 16,20 operators and it needs 0.5 seconds with source state t1 to the link state in t2 cycles respectively using the function STi.LS

Table 2: Minimum Area Schedule

Operator	*#Needed*	*Cycle 1*	*Cycle 2*
8.8=16	0.5	t1+source	t2. linkstate
10.10=20	0.5	t1+source	t2. linkstate

After determining that schedule, behavioral synthesis will create the (RTL) register-transfer level implementation. Note the multiplexing that has been added in front of the multiplier because it is being shared. Additionally, behavioral synthesis would also determine the FSM (Fish Eye State Module) .

10. CONCLUSION

This paper gives some Guidelines to enhance the security mechanism to improve and to detect malicious code using Fish Eye Code Behavior Algorithm. Compared with content filtering mechanism and slow patching process this one is different in code behavior features. The algorithm is much efficient and reliable that it can detect and filter the latest Viruses, Trojans, Spam and Worms especially in the emerging operating systems and in the current Software industrial scenario**.**

11. REFERENCES

[1] Bright Hub available at: www.brighthub.com

[2] Info Security available at: www.infosecurity-us.com

[3] Computer Security Training, Network Research & Resources available at: www.sans.org

[4] PHYSORG available at : www.physorg.com

[5] TalkTechToMe available at:www.gfi.com/blog/tag/patch-management

[6] HACKER BOSS available at: www.hackerboss **.** com

[7] Function Extraction Technology: Automated Calculation of Computer Progam Behavior available at: www.cert.org/sse/fxmc.html

[9] Web Application Vulnerabilities available at: www.acunetix.com/vulnerabilities

[10] D.Slur, J.Crupi, and D.Malks, "Core J2EE patterns: Best practices and architecture strategies". Sun Micro Systems, 2001.

[11] G.Abowd, R.Allen, and D.Garlan, "Using Style to Give Meaning to Software Architecture", ACM, New York, 1993.

[12] guardian.co.uk available at: www.guardian.co.uk/2011/google-android- patent-lawsuits-batt.

[13] Peter Braun, Manfred Broy, Frank Houdek, Matthias Kirchmayr and Mark Müller,et al.Online First™, 20 October 2010

[14] Securing SQL Server available at: www.securingsqlserver.com/tag/sql-injection

[15] Performance comparison and analysis of mobile ad hoc routing protocols: An International Journal (CSEIJ), Vol.1, No.1, April 2011

[16] Dahl O.-J., Dijkstra E. W., and Hoare C. A.,Structured Programming, Academic Press, 1972

AUTHOR PROFILE:

A.EdwinRobert, MCA., M.Phil., in Computer Science. He is a Doctoral research scholar in Computer Science at the Karpagam University, Coimbatore, Tamilnadu,India.He is currently working as Assistant Professor in Software Systems Department at Karpagam University,Coimbatore,TamilNadu.He had five years of teaching experience. He has presented a paper in National Conference**.** His area of research is Software Engineering.

N.Sasirekha, completed MCA., M.Phil., in Computer Science. She is a Doctoral research scholar in Computer Science at the Karpagam University, Coimbatore, Tamilnadu, India. She is currently working as Assistant Professor in PG Department of Computer Applications at Vidyasagar College of Arts and Science, Udumalpet, Tamilnadu. She has presented eleven papers in various National Conferences and one paper in International Conference. She has published a paper in National and International Journal. Her area of research is Software Engineering.

Dr.M.Hemalatha, completed MCA., MPhil., PhD in Computer Science and Currently working as a Asst Professor and Head , Dept of Software Systems in Karpagam University. She has ten years of experience in teaching and published Twenty Seven papers in International Journals and also presented Seventy papers in various National Conferences and one paper in International Conference. Her areas of research are Data mining, Software Engineering, Bioinformatics and Neural Networks. She is also a reviewer in several National and International Journals.