CHAPTER TWO: LITERATURE REVIEW

This chapter describe the foundation of autonomic computing which is essential comprehend in this work, we as well present an important state of the art and research that is related to this subject.

Network systems are consist of three entities in order to perform communication. First is sending information and receiving information, additionally the information itself which they need to share it, second is the channel or the medium, ant third is the agreement between the tow entities which is called protocol.

The entities mentioned above is now a mixed of hardware and software, which is called network or computer network. The element of hardware is composing of several components such as: hosts, hubs, bridges, routers, gateways…ect.

The element of the network may own resources individually, that is locally, or globally. Network software consists of all application programs and network protocols that are used to synchronize, coordinate, and bring about the sharing and exchange of data among the network elements. Network software also makes the sharing of expensive resources in all networks possible. Network elements, network software, and user all work together so that individual user gets to exchange message and share resources on the system that are not readily available locally. The network element together with their resources may be of various hardware technologies and the software may be as diverse as possible, but the whole combination must work together in union.

The progression of computing research has gone through many generations, beginning from a single process running on a single computer system to multiple processes running on geographically dispersed heterogeneous computers that could extent several continents.

A number of systems were mostly concerned about performance, and focused intensive research on parallel processing and high performance computer architectures and applications to address this condition. As the scale and distribution of computer systems and applications evolved to cover critical areas where failures can be catastrophic, the reliability and availability of the systems and applications become a major concern. This, in turn has lead to a separate research activities that focus on reliability and fault tolerance computing. In the same way, the research in computing security has also evolved to address the needs to protect the integrity and security of computing systems and their services without consideration to other important system attributes such as performance, reliability, and configurability because security and system assurance are the main objective of such systems and services.

The emerging and promising systems and applications are complex, dynamic, and their requirements change continuously within one application or a class of applications. Consequently, their high performance, fault tolerance, security, availability, configurability requirements might change dynamically at runtime. Hence, it is very critical for future computing systems and/or software architecture to be adaptive in all its attributes and functionalities (performance, security, fault tolerance, configurability, maintainability, etc.). There has been research to integrate all these techniques into one computing system and is mainly being characterized as ad hoc. However, the integration of these isolated techniques into one system produces a system that is complex, unpredictable, unmanageable and insecure; the actions performed by the security technique might cancel the actions taken by the high performance computing technique and so on.

It is considerably challenging research problem to integrate all these different techniques such that it is practicable to maintain simultaneously and efficiently in real-time all system functionality and attributes such as performance, fault, and security. Fundamentally, a new paradigm is needed to design and develop large-scale complex and adaptive systems and applications.

In our approaches we develop a Model that is based on autonomic computing to self-regenerate the system component.

2.1 Autonomic Computing

Autonomic computing systems are those systems that repeatedly administer or control themselves by performing tasks that have been usually performed by computer specialists.

Autonomic Computing is an initiative started by IBM (Horn, 2001) in 2001. Its crucial aim is to create self-managing computer systems to overcome their rapidly growing complexity and to enable their further growth. The term autonomic is derived from human biology.

The autonomic nervous system monitors your heartbeat, checks your blood sugar level and keeps your body temperature close to 98.6°F without any conscious effort on your part. In much the same way, self-managing autonomic capabilities anticipate IT system requirements and resolve problems with minimal human intervention. As a result, IT professionals can focus on tasks with higher value to the business.

However, there is an important distinction between autonomic activity in the human body and autonomic activities in IT systems. Many of the decisions made by autonomic capabilities in the body are involuntary. In contrast, self-managing autonomic capabilities in computer systems perform tasks that IT professionals choose to delegate to the technology according to policies. Adaptable policy - rather than hard-coded procedure - determines the types of decisions and actions that autonomic capabilities perform.

Self-managing capabilities in a system accomplish their functions by taking an appropriate action based on one or more situations that they sense in the environment. The function of any autonomic capability is a control loop that collects details from the system and acts accordingly.

In a self-managing autonomic environment, system components - from hardware (such as storage units, desktop computers and servers) to software (such as operating systems, middleware and business applications) - can include embedded control loop functionality. Although these control loops consist of the same fundamental parts, their functions can be divided into four broad embedded control loop categories (Horn, 2001) (Ganek et al, 2003) (Kephartet al, 2003). These categories system (Aaron, 2005) (Zach, 2005) (Vincent, 2006) are considered to be attributes of the system components and are defined as:

- **Self-configuring**

Self-configuring components adapt dynamically to changes in the environment, using policies provided by the IT professional. Such changes could include the deployment of new components or the removal of existing ones, or dramatic changes in the system characteristics

- **Self-healing**

Self-healing components can detect system malfunctions and initiate policy-based corrective action without disrupting the IT environment. Corrective action could involve a product altering its own state or effecting changes in other components in the environment.

- **Self-optimizing**

Self-optimizing components can tune themselves to meet end-user or business needs. The tuning actions could mean reallocating resources—such as in response to dynamically changing workloads—to improve overall utilization, or ensuring that particular business transactions can be completed in a timely fashion.

Self-optimization helps provide a high standard of service for both the system's end users and a business's customers. Without self-optimizing functions, there is no easy way to divert excess server capacity to lower priority work when an application does not fully use its assigned computing resources. In such cases, customers must buy and maintain a separate infrastructure for each application to meet that application's most demanding computing needs.

- **Self-protecting**

Anywhere Self-protecting components can detect hostile behaviors as they occur and take corrective actions to make their components less vulnerable. The hostile and aggressive behaviors can include unauthorized access and use, virus infection and proliferation, and denial-of-service attacks. Self-protecting capabilities allow businesses to consistently enforce security and privacy policies.

It is an important to note that an autonomic computing system addresses these issues in an integrated manner rather than being treated in isolation as is currently done. Consequently, in autonomic computing, the system design paradigm takes a holistic approach that can integrate all these attributes seamlessly and efficiently. The autonomic system can be viewed as a collection of autonomic components (AC) or building blocks that each can support the four properties outlined above. That means, each AC can be dynamically and automatically configured (self-configuration), seamlessly tolerate any component failure (self-healing), automatically detect attacks and protect against them (self-protection), and automatically change its configuration parameters to improve performance. Once it deteriorates beyond certain performance threshold (Self-optimization), once these autonomic components become available, we can dynamically build an autonomic computing system to meet any static or dynamic requirement such as cost-effective high performance systems, high performance and secure systems, etc.

2.2 Computer Threats

Computer malicious software is evolved from what was once a manifestation of skills to what is now a worldwide criminal industry, motivated by financial profits as well as political and militarily persuasions. Computer viruses and worms are increasingly being used to carry out cyber attacks on corporations and individuals. Motivations for these attacks include financial gain, access to classified information, and disruption of services. This trend continues to grow at an extraordinary rate making it difficult for the current cyber defense industry to maintain full protection against attacks.

2.2.1 Computer Attack Detection

The goal of intrusion detection is to monitor network assets to detect anomalous behavior and misuse. This concept has been around for nearly twenty years but only recently has it seen a dramatic rise in popularity and incorporation into the overall information security infrastructure.

2.2.2 Types of Intrusion Detection Systems

Intrusion detectors nowadays used different technique to detect intruders and anomalies or attacks. Different structure implement different detectors, commonly intrusion detection systems fall into two broad categories. Which are:

- **Network based systems (NIDS):** These types of systems are placed on the network, nearby the system or systems being monitored. They examine the network traffic and determine whether the traffic is legitimate.

- **Host based systems (HIDS):** These types of systems actually run on the system being monitored. These examine the system to determine whether the activity on the system is acceptable.

2.3 Distributed Component Techniques

Component-Based Development is becoming a main stream software development paradigm which reuses the off-the-shelf components and assembles them together to form a component-based application.

A widely accepted perspective is that component is a black box, and the interface specification is an abstraction of component's behavior and also the abstraction of the service that components require or provide. Based on this principle, engineers and researchers provided their own solution to cope with software behavior description and analysis problem. For example OMG proposed the CORBA architecture, Microsoft proposed the COM/COM+ technique, and SUN proposed the EJB solution.

All these measures have a common feature that they all regarded interface description language (IDL) as interface specification and also regarded it as component's behavior specification. However, IDL only describes the services types, parameters and what services the component provide and require, and omit the more important facades, internal and external behavior/state transition process, so in this sense, the IDL does not entirely solve the component behavior relativity analysis problem(Wang Wei, 2008).

Based on that concept, a variety of distributed component models have been effectively implemented in industry, as well as in academia. DCOM, .Net, CORBA, EJB, CCA Microsoft's COM and DCOM are examples for distributed technique existing at the present time. These frameworks have been fundamental to interoperability in Windows based applications. Their current web service oriented .NET framework is also a component based and is gaining widespread acceptance. Web Services present another alternative distributed computing infrastructure; an alternative that is being strongly promoted as preferable to the use of distributed object middleware such as CORBA(Szyperski, 2002) or Java RMI.

Web service is a self-contained, self-describing, modular, loosely coupled and reusable application that can be published, located, and invoked across the web.

2.4 Related Work

In this part we present a handful of research done in different area, such as bio-inspired technology and immune system.

The development in the area of bio-inspired engineering is basically relying on the artificial immune system, swarm intelligence, evolutionary (genetic) algorithms, and cell and molecular biology based approaches (Camazing et al, 2003). The response of the immune system, even to unknown attacks, is a high-adaptive process. Therefore, it seems noticeable to apply the same mechanisms for self-organization and self-healing operations in computer networks.

Current researches areas are concern mostly in cell and molecular biology based approaches. All organisms are constructed in same way. Composed of organs, which consist of tissues and finally of cells (Dressler, 2004). This structure is very similar to computer networks, which is also true for the signaling pathways.

Therefore, research on methods in cell and molecular biology promises high potentials for computer networking in general and network security in particular. His focus is on cell biology as a key for computer networking. He first shows structural similarities followed by some information about the signaling pathways within single cells and between tissues.

Based on obvious similarities, high potentials of this analysis are worked out which will lead to paradigms and algorithms showing a higher efficiency in computer networking. Network security concerned of Cellular information exchange examines the information exchange in Cellular environments and to extract the issues in computer networks, that can be addressed by the utilization of these mechanisms .applying it to the computer network. Information exchange between cells, called signaling pathways, follows the same requirements as between network nodes. A message is sent to a destination and transferred, possibly using multiple hops, to this target (Dressler, 2004).

The information transfer works as follows. A specific signal reaches only cells in the neighborhood. The signal induces a signaling cascade in each target cell resulting in a very specific answer which vice versa affects neighboring cells. A cell is shown with a single receptor that is able to receive a very specific signal and to activate a signaling cascade which finally forms the cellular response (Dressler, 2004).

Such mechanisms are there in networking environments. Toward network security solutions, monitoring probes gather information about the ongoing traffic in the network. The collected data will be sent to an attached intrusion detection system for further processing. Firewall systems are configured with rules concerning the actual behavior in the network. General issues have been addresses in such a network are: Adaptive group formation, Optimized task allocation, efficient group communication, Data aggregation and filtering, Reliability and redundancy (Dressler, 2004).

Proteins are used as information particles between cells. A signal can be released into the blood stream, a medium which carries it to distant cells and induces an answer in these cells which then passes on the information or can activate helper cells (e.g. the immune system). The interesting property of this transmission is that the information itself addresses the destination. Only cells with a very specific receptor are able to receive the information, i.e. the protein binds at the receptor (Dressler, 2004).

The mechanism is self-organizing. In the past, such mechanisms were provided for identifying individual nodes based on an individual property, e.g. routing protocols are in some manner self-configuring by identifying neighboring nodes. yet, the final goal is to put new nodes into an existing network without any preconfigured knowledge. The properties of the nodes can be described in some common way. Bio-inspired communication mechanisms learnt from intercellular signaling pathways provides the appropriate mechanisms (Dressler, 2004).

Self-organization (Dressler, 2004) issues are promising to be the key answer to build large and complex systems fulfilling different tasks out of many simple independent autonomous entities. Such systems can be found quite often in computer networking. Network security, pervasive computing environments..ect.

Another biological paradigm (Bush et al, 2003) toward adaptation as a computer security paradigm is protein pathway mapping. Living organisms have complex metabolic pathways consisting of interactions between proteins and enzymes, which may themselves have multiple subunits, alternate forms, and alternate specificities.

Molecular biologists have spent decades investigating these biochemical pathways in organisms. These pathways usually relate to a known physiological process or phenotype and together constitute protein networks. These networks are very complex, with several alternate pathways through the same start and end point. The partitioning of networks into pathways is, however, often arbitrary, with the start and finish points chosen based on “important” or easily understood compounds. The models for biochemical pathways that have been developed thus far primarily demonstrate the working of the cellular machinery for specific tasks, such as metabolic flux and signaling.

Similar to the cellular networks in organisms, computer networks are complex in nature and collectively exhibit complex behavior. In these networks, start and end points can be arbitrarily chosen, and multiple paths may exist between the same nodes. Protein networks are predetermined and stay fairly static, whereas computer networks are constantly evolving with the addition of new nodes and network links. In protein networks, interactions among proteins, enzymes, and catalysts culminate in specific events. Analogously to protein networks, interactions among nodes of computer networks result in specific events or conditions in the network. The events may include propagation of viruses, denial-of-service attacks, and congestion on the network. Investigation of the network pathways along which the events propagate will enable us in forensic analysis to determine the root cause of the failures, as well as helping in developing intelligence for prediction of network events.

Bush used the biological paradigm of the immune system, coupled with information theory, to create security models for network security. Information theory allows generic metrics and signatures to be created which transcend the specific details of a system or an individual piece of code. They compare information-theoretic approaches with traditional string-matching techniques. They also provide an analytic model that uses the epidemiological paradigm to study the behavior of the nodes (Bush et al, 2003).

The role of the human immune system is to protect our body from pathogens Such as viruses, bacteria, and microbes. The immune system consists of various kinds of cells, which operate autonomously and through interaction with each other to create complex chains of events leading to the destruction of pathogens. At a high level, cells can be categorized into two groups: detectors and effectors. Detectors identify pathogens, and effectors neutralize them. There are two kinds of immune responses evoked by the immune system, innate response and adaptive response. The innate immune response is the natural resistance of the body to foreign antigens and is non-specific toward invaders in the body. During this response, a specialized class of cells called phagocytes (macrophages and neutrophils) is used. These specialized cells, which have surface receptors that match many common bacteria, have remained unchanged throughout evolution.

Artificial immune systems (Bush et al, 2003) consist of detectors and effectors that are able to recognize specific pathogen signatures and neutralize the pathogens. To detect pathogens, the signature of incoming traffic packets is matched against signatures of potential viruses stored in an immune system database. An immune system that is capable of recognizing most pathogens requires a large number of detectors. Low-specificity detectors that identify and match several viruses are often used to reduce the number of detectors at the cost of increased false positives.

Computational complexity of a computer immune system remains fairly high, and individual nodes are incapable of garnering enough resources to match against a large signature set. The computational complexity gets worse as network traffic grows due to use of broadband networks, and it is straining the capacities of conventional security tools such as packet-filtering firewalls. Massive parallelism and molecular-level pattern matching allow the biological immune system to maintain a large number of detectors and efficiently match pathogens. However, artificial immune systems have not achieved these levels of efficiency. To reduce the computational burden on any individual node in the network, all nodes need to pool their resources, share information, and collectively defend the network. In addition, such inspection should be done within the network itself, to improve efficiency and reduce the time required for reacting to an event in the network. This concept of collective defense enabled by a unified framework .To enable this concept of collective network defense; they have proposed an approach based on information theory principles using Kolmogorov Complexity measures.

To study the parameters and different schemes of detection and sampling in the immune system, Goel have developed a simulation model using RePast (Schaeffer et al, 2004), a simulation tool typically used for model.

The security models for detection and elimination of pathogens that overrun computer networks have been based on perimeter defense. Such defenses are proving inept against fast-spreading viruses and worms.

The current tools are unable to guarantee adequate protection of data and unfettered access to services. It is imperative to complement these existing security models with reactive systems that are able to detect new strains of pathogens reliably and are able to destroy them before they can cause damage and propagate further. Several biological paradigms provide a rich substrate to conceptualize and build computer security models that are reactive in nature. Three specific mechanisms in mammalian organisms present the most potential.

Falco Dressler (Dressler, 2007), creates two separate feedback loops as inspired by similar solutions found in nature. These feedback loops represent promoter /inhibitor functions, for example they either stimulate monitoring probes to send data, such as higher quality data or they suppress this amplification effect if the detection modules approach their maximum capacity. On-demand re-configuration is required in the case of resource shortages. Self-optimization refers to the overall detection quality. This can be achieved by exchanging information about identified attacks or suspicious network connections and also by statistically forwarding parts of collected data packets and network statistics to neighboring probes.

He develops a simulation models to analyze the scalability of the developed approaches. Basically, he implemented the behavior of monitoring, firewall, and attack detection systems. In order to allow practically significant simulations and to easily compare different configurations, he used previously monitored data for trace-driven input modeling, and then studied the configuration and possible adaptation of individual subsystems to increase the scalability and reliability of the overall system. It turned out that the dynamic reconfiguration depending on the current network behavior is possible without any global control.

He also identified appropriate mechanisms in cell biology and to adapt them to networking technology with the focus on self-organization based on adaptive feedback loops. A structural comparison of organisms and computer networks depicts that both show high similarities.

Also, the communications between the systems, the signal transduction pathways, follow the same requirements. His developed adaptive control scheme depends on the current network behavior, such as, on the observed traffic as well as on the current state of the individual subsystems. He wants to adapt the parameters of the monitoring environment depending on the load of the detection system and of the current network behavior.

Two kinds of feedback loops have been implemented by Dressler; they are used in combination, positive feedback for short-term amplification and negative feedback for long-term regulation. The intrusion detection reports detected attacks to the firewall that in turn blocks this traffic and, therefore, reduces the number of packets that have to be monitored. Additionally, the detection system reports legitimate traffic to the monitor. This monitor stops reporting on packets belonging to these flows and, therefore, reduces the number of packets that have to be analyzed. Obviously, both configurations cannot be permanent. Sources sending legitimate traffic might begin to send attack packets at any time. In addition, previously attacking machines might become 'corrected' and should not be starved by our firewalls. The target of his researcher is the adaptation of biologically inspired promoter/ inhibitor schemes for adaptive parameter control in network security environments, using an amplifying positive feedback loop and a suppressing negative feedback loop (Dressler, 2004)

Our Autonomic agent architecture is similar to IBM's approaches. However, we define general detection and component management interface (CMI) as well as configuration. We are going to build our Component Runtime Manager (CRM) as different agent to computerize and automate the control and management of networked software.

3.6 Fundamental of Detection and Identification of Malicious Software

In order to analyze or study computer attacks, one has got to have the resources to recognize and distinguish their presence and identify them. This is true whether one is trying to isolate viruses from diseased hosts or to simply study their properties. The presence of a virus in a host body is recognized by the occurrence of either of the following:

- Abnormal demonstration
- Direct observation of virus components.

The detection of new malicious, rarely depends on direct observation of virus components, since components specific to the virus are yet to be discovered. as an alternative, the malicious is usually discovered based on an observation of its abnormal manifestation. In fact, only after initial observation of the malicious actions, the malicious body is revealed after careful analysis of its internal components.

Whenever we suspect the presence of malicious software we must find a set of conditions, a suitable host, and a suitable replication environment in which the virus will reproduce and continue the infection process.

Based on the previews, any malware researches that consist of setting up experimental infection begin in an appropriate and a reliable environment established after malicious software introduction.

If the malware has been introduced to a node, an analysis can be performed in it behaviors or its body and so on.

- **Malware Propagation**

In view of the fact that the main property by which computer malware are recognized is their Infectiousness, the study of malware replication is based on measurements of the infectivity of a computer malware as a function of time after successful inoculation of the virus. The first and basic attempts to apply replication models of biological viruses to determine rates of propagation in their computer counterparts were done by Cohe(Cohen et al, 1988) and others.

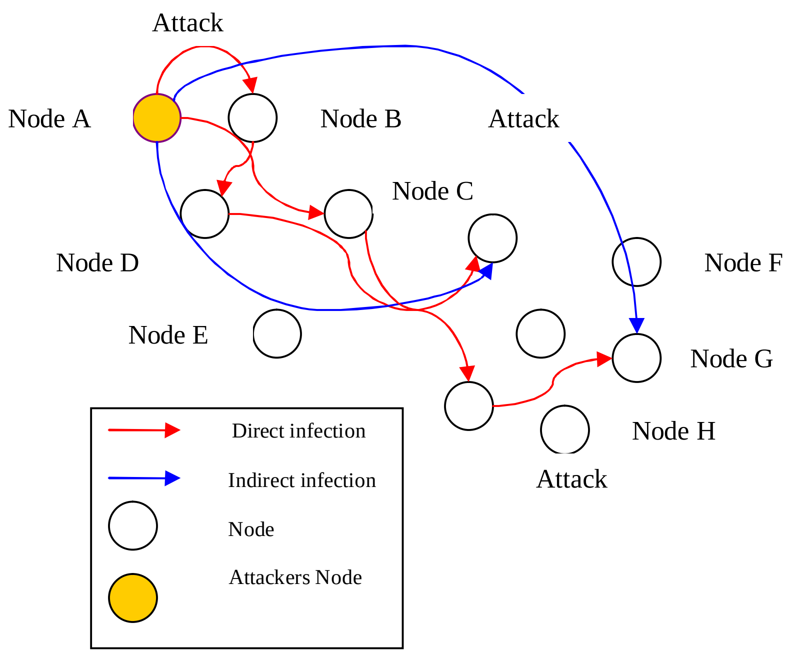


Figure3-3: Attacks Propagation

It said that malware can spread through computer system or network, which is if an infected Node A can infect Node B and propagate further to Node B through Node C. in this case we can say there is a propagation from Node A to Node G.

3.7 Current Detections Method

The most important indication of an attack infection on a vulnerable Node is a unexpected system malfunction, that is, the disruption of the normal operational process that accompanies a change in software components and/or system structure, as well as the presence of newly attack particles. The disruption is caused by attack operations that are often designed to be hidden from the main operation cycle. Such behavior depends on the specific computer malware implementation.

There has been a continuous adaptation of attacks to avoid new detection techniques, and at the same time, significant improvements to the detection process have facilitated an arms race between the attackers and the defenders.

Attempts have been made to straighten safety measures. These attempts achieved various levels of success and adaptation, and into these main categories:

- Monitors.
- Scanners.

3.7.1 Scanners

Earlier, a software toolkit made to target known attacks based on their signatures and behaviors appeared on the market, including Peter Tippett's heuristic scanner capable of detecting various malicious.

Scanner is a computer program designed to search for and identify code sequences or signatures that are indicative of the presence of known attack in an application, computer or network. With first versions developed in late 1980s, scanners still maintain the leading position. Although they are recognized by many (FitzGerald, 2003) as a very ineffective and unsuccessful protection approach in computer security, scanners are still being used to detect known malicious threads.

Relative simplicity of development and maintenance are the key factors in the high popularity of scanners. This is especially true for the programs that only facilitate detection of a known virus object, before the discussion on disinfection or further investigation. The Main components of such scanners comprise:

- Scanning engine
- Known attacks Database.

During initializations or execution, the scanner searches through the Node in an effort to find a match to a known virus signature stored in the database. Such signatures are often called scan strings. User maintenance of such scanners only requires systematic updates of virus signatures in databases, and systematic scans of the entire system to make sure that no programs are infected.

There are a few challenges in the development of such scanners, often directed on improvement of scan speed and detection accuracy, however, overall development maintenance is limited to extracting signatures from known malware and updating the database.

In addition, scanners provide the capability to detect malware without executing the infected program. This illuminates the potential risk of compromising the system during the execution of a malware, but it also comes at a price of having a false sense of security when the scanner is not properly updated, or it is not capable of detecting attacks under certain conditions (e.g. encrypted, polymorphic, unknown host type, etc.).

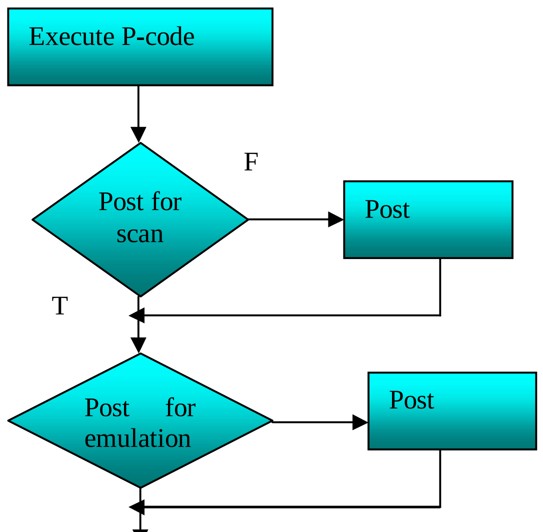
Scanners have boundaries as well. Since the technology is based on signature matching, it is only applicable to signatures that are already known. New attacks, as well as modifications of known attacks appear to be different from what the scanner would expect, and therefore, they will not be detected, identified and disarmed.

This difficulty is especially acute due to the extremely high rate of malware production (tens of thousands per year as of 2006, and the number increases each year). Except the database of signatures is being updated continuously, the scanner becomes out of date in a very short period of time. This provides users with a false sense of security, as the scanner reports no infections when in fact; it fails to identify new threats.

Users are required to maintain attack signature databases updated in order to be protected from new threats. Before the signature is ready to be included in an update to the database, it must be extracted from the malware in such a way that it will not interfere with any known non malicious programs, thereby avoiding false positives. While signature extraction and generation is a relatively simple process when done by a skilled professional, complications arise when the rate of new malware release is high. Since a fully automated, error prone way of signature generation is not yet feasible.

To exemplify the idea and concept of design and operation of a typical advanced scanner, this is capable of detecting the presence of known attack in infected node. We should be having multiple entry points (EP) by utilizing a scanning technique. In its attempt to detect an attack, the detector employs most modern detection techniques including Code Emulation and Scanning.

As described in figure below, the algorithm can detect pattern of malware by executing the P-code. There is certain malware definition embedded into the P-code. The definition has to be scanned and posted for emulation. If the patten is a signature to a malware, the algorithm takes decision to describe the software as a malware. Figure 3-4 describes the processes.



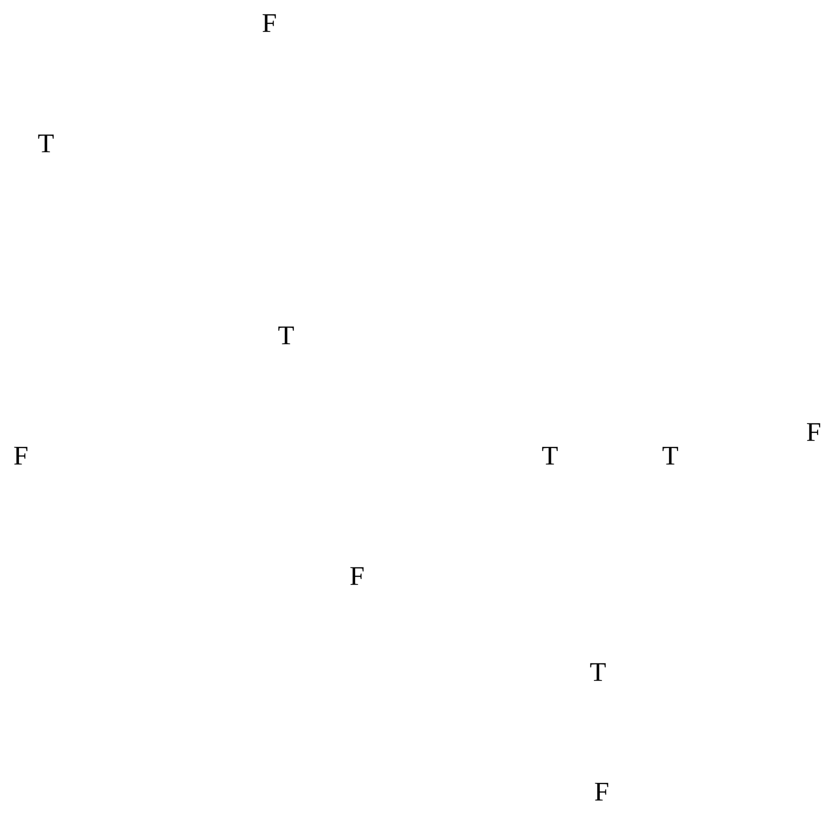


Figure 3-4: Show a typical scanning technique (Volynkin, 2007)

3.7.2 Monitors

Monitors are usually supplemental to malware scanners and are used in order to reduce the scanning area. Essentially, monitors are memory resident programs, which continuously monitor some functionality of the operating system. Depending on the system, the functionality includes access to dangerous interrupts, memory locations, system files, etc. Such functionality is considered and measured indicative of viral behavior. When a program tries to access dangerous areas in the OS, the monitor intercepts the demand and either denies it totally or asks the user for permission to release the program.

Monitors are not only for detecting malware. Therefore are capable of detecting previously unknown attacks based exclusively on the code's behavior. In addition, monitors are part of the decision making process in most modern anti-virus software tools.

Most computer epidemics are carried out via the Internet by the transmission of files that contain the code of a computer attacks. Upon receiving the file, the target computer executes the malicious code resulting in the reproduction of the virus or worm and the delivery of its potentially destructive payload. Self-replication, which is uncommon in legitimate programs, is vital to the spread of computer viruses and worms.

This mechanism allows them maximize the effectiveness of the attack and create computer epidemics. As with any function, self-replication is programmed; the sequence of operations resulting in the self-replication is present in the computer code of the virus. The implementation of the function of self-replication is not unique; there is more than one sequence of operations that can perform this task. Moreover, it is assumed that these sequences are dispersed throughout the entire body of the code and cannot be detected as an explicit pattern.

The implementation is for extracting self-replication sequences from such viruses. The technology presented herein is applicable to executable malware, which represent the most common and difficult to detect forms of malware. The detection is conducted at run-time during normal code execution under regular conditions by monitoring the behavior of every process with regard to system calls, their input and output arguments and the result of their execution. Unlike existing antivirus software, this methodology facilitates advanced proactive protection from both known and previously unknown attacks.

3.7.2.1 Definition of the gene of self-replication for process Behavior analysis

The GSR is viewed as a specific sequence of commands passed to the computer operating system by a running program that causes the program's code to be replicated through the system or multiple systems. Replication can be accomplished in several ways depending on a particular computer system and the software the system is running (Volynkin et al, 2007).

For example, early computer attacks designed for the Microsoft DOS operating system utilized direct access to hardware for this purpose. The widespread introduction of microprocessors that allowed for different privilege levels and operating systems supporting and enforcing these levels facilitated new methods of self-replication.

3.7.2.2 GSR Structure

Virtually every process running in a system issues system calls; however, they are not mixed and can easily be differentiated for every process and thread. The system calls generated at run time by a process represent a direct time line sequence of events, which can be analyzed during execution. Depending on the nature of the process and on the system resources it is trying to access, this sequence can be long or relatively short (Volynkin et al, 2007). Though the GSR is contained within the sequence of calls produced by a malicious process, it can be dispersed throughout that sequence.

3.7.2.3 Behavior monitoring:

Behavior monitoring is to catch calls and determine whether it malware or not. A complex computer operating system such as Microsoft Windows XP receives thousands of calls every second from many different processes. Most of the function calls, produced by an application in user mode deal with secure objects and hardware resources such as File System, Processes and Threads, Graphical System Services, System Registry, etc., are transferred into the Kernel mode of the operating system for further execution in a secure environment. During this process, function calls are processed into system calls for unification, compatibility, security and other reasons. At the Kernel level, system calls are processed by the System Service Dispatcher (SSD) and routed to a designated service (Volynkin et al, 2007).

The Operating System itself provides us with almost no support for monitoring its Kernel level for security reasons; therefore such a software monitor has to be created. While it is not a unimportant task, as it requires very low-level system design and implementation, a very basic idea for the monitor is shown in figure 3-5.

When the Kernel receives a function call from the user mode, it has to decide which Kernel interface to call to process this function. The API Processing Unit, also known as System Service Dispatcher (KiSystemService) is responsible for making this decision by looking up an appropriate system call handler in its System Service Table (SST) and invoking it. The SST stores handlers to every system call supported by the Kernel (see Appendix D.1 for details). If the handler to a particular system call in SST is replaced with a fake one pointing to other memory location, the System Service Dispatcher will simply execute a different function at that location. This extra function can be designed to gather information about the system call, its parameters and the origin of the call (Volynkin et al, 2007).

When all needed information is collected, the function calls, the original system call, and the entire system proceeds as usual.

All system calls, once invoked at the Kernel level, are expected and likely to produce a result. This result is represented by the output arguments of the

system call, as well as the return value that confirms successful execution, or indicates errors. All system calls, intercepted by the monitor, appear in two parts: system call with input arguments and system call with output arguments.
``

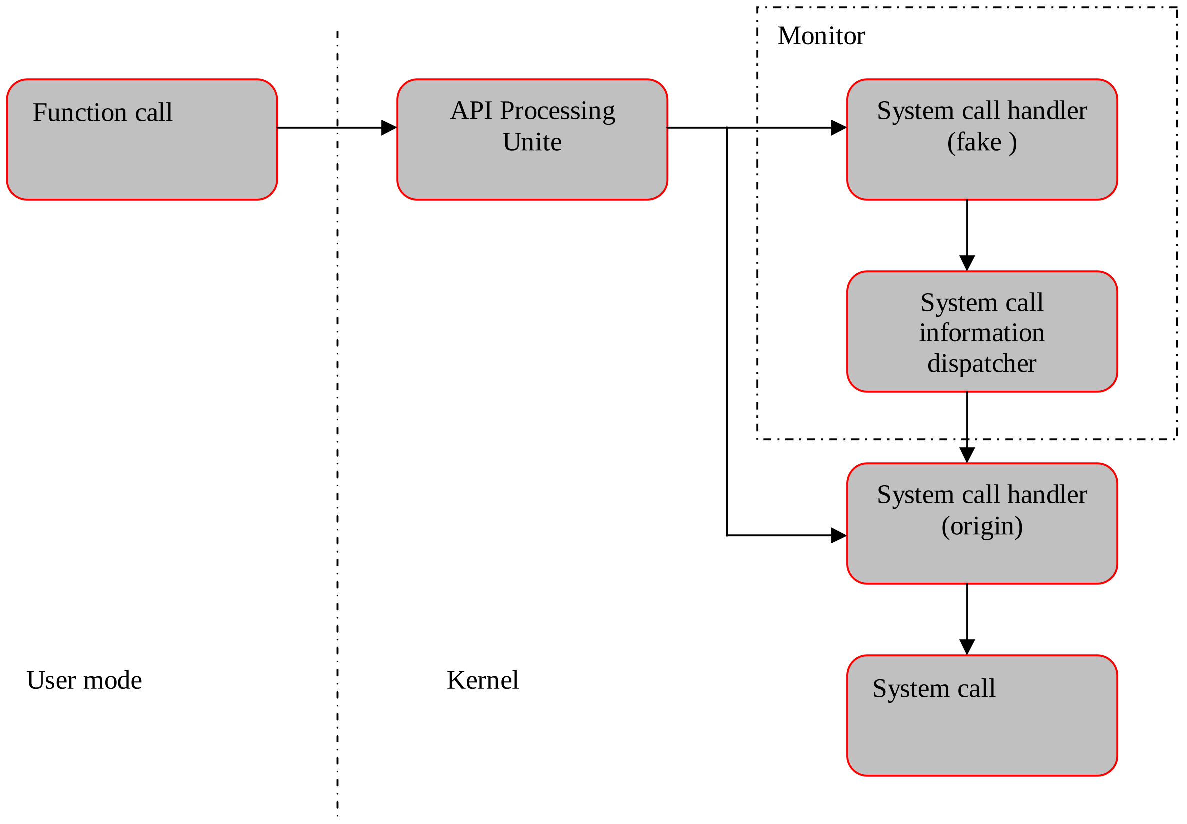

Monitor
Function call
API Processing Unite
System call handler (fake)
System call information dispatcher
System call handler (origin)
User mode
Kernel
System call

Figure 3-5: Functionality of the System Calls Monitor (Volynkin et al, 2007)

Having the information, observed by the monitor, it is possible to conclude, that a particular Thread that belongs to a particular Process, invoked a system call for the purpose of opening a file named “virus.exe”.

In order to detect if such a call belongs to any parts of the virus’s self-replication, most of its input and output arguments must be considered. While it is obvious that any system call by itself with all possible combinations of input/output arguments cannot be considered as a threat, certain APIs called with certain arguments are combined, they represent a clear pattern of self replication.

During the GSR detection process, every system call intercepted by our monitor comes into the Replication Detector, where it goes through a complete range of different detection and filtration mechanisms (Volynkin et al, 2007).

3.7.2.4 Replication Behavior Analysis

The concept of GSR definition requires building a pyramidal structure with basic system calls at the bottom, combinations of calls represented by Blocks in the middle, and the GSR itself at the top. While replication is usually not a very complicated process, it involves using system calls to perform a number of operations. Therefore, the complexity of the GSR definition depends on several facts:

- The number of unique system calls involved.
- The number of inter-functional relationships among system calls.
- The complexity of inter-functional relationships.

Since the margin between malicious and normal behavior can occasionally be small, it is important to keep the preciseness of the GSR definition at the highest level possible in order to avoid false positives. On the other hand, some flexibility when connecting blocks of the GSR is needed to prevent false negatives

3.7.2.5 Replication over the local Network and the Internet

Once computers started communicating with each other using local networks, malwares writers exploited this feature for self-replication. Networking opens continuous possibilities for an attack to replicate itself to as many computer systems as it possibly can within the network instead of just infecting a limited number of compoannts on a single host machine. Such remote replication is possible with the use of specific network protocols implemented by the operating system.

Replication over the network is very alike to local replication with the main difference being the necessity for a computer virus to enumerate available network resources before it can access target files on a remote computer. Therefore, a complete algorithm of virus replication for a parasitic virus, which attaches itself to an existing file by injecting its code into the body of the executable and replacing code entry points, would look as follows (Volynkin et al, 2007):

1. Open “Virus.exe”
2. Read “Virus.exe” Code
3. Enumerate network resources
4. Open remote “Host.exe”
5. Inject Code into “Host.exe”
6. Patch “Host.exe” Entry point

For this reason, we only need to add one block into the Gene’s syntax describing Network resources enumeration in order for the detector to recognize the behavior. However, enumeration can be accomplished in several different ways, including sockets, remote procedure calls, named pipes, NetBIOS and other networking APIs.