Towards a secure Web server

by

Femitha Majeed

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Johnny Wong (Major Professor)
Les Miller
James Davis

Iowa State University

Ames, Iowa

2002

Graduate College
Iowa State University

This is to certify that the master's thesis of

Femitha Majeed

has met the thesis requirements of Iowa State University

Signatures have been redacted for privacy

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor and major professor, Dr Johnny Wong, for his support, patience, and encouragement throughout my graduate studies. It is not often that one finds an advisor who always finds the time for listening to the little problems and roadblocks that come up in the course of graduate studies.

My thanks also go to the members of my graduate committee, Dr Les Miller and Dr James Davis for taking the time to serve on my committee and reviewing my thesis. I would like to express my thanks to Yanxin Wang and Jeff Gu for all their help and suggestions.

My thesis wouldn't have been possible without the support and encouragement I received from across the seas. My husband, parents, sister and brother receive my deepest gratitude and love for their dedication and support.

Finally, to God, the most gracious, the most merciful...

ABSTRACT

Securing a web server on an insecure operating system can often prove to be unsuccessful. This leads us to consider structuring an operating system architecture specially configured for a secure web server. The first half of the paper presents an analysis of some common attacks against a web server. In the second half, the paper focuses on ways to secure a web server. An essential phase in securing a web server consists of securing the operating system on which the server is run. This is important because compromising a flaw in the operating system might lead to an attack on the web server. Denial of Service (DOS) attack is one of the most common attacks that are aimed at the web server. It can be addressed to a large extent by using a proper resource control mechanism. We propose a security architecture design that integrates resource control and accountability into Mandatory Access Control (MAC) architecture. The implementation incorporates resource control into SELinux, which has MAC built into it. This is then integrated with Mobile Agent Intrusion Detection System (MAIDS), which is a framework for an intrusion detection system that is modularly compatible with other detection systems. Integration with MAIDS is done to alert the system administrator whenever a DOS attack occurs. The MAIDS software will monitor the resource control mechanism to check whether a DOS attack has taken place or not. Finally, we present the design and implementation of a security tool that checks for configurations of the web server and the operating system on which it is run.

1 INTRODUCTION

1.1 Motivation

The Internet is one of the most widespread technologies in the world. There are more than a hundred million users connected to the Internet and the number increases everyday. With the rapid growth of the Internet, the number of attacks aimed at it has also escalated. According to Computer Emergency Response Team (CERT) [19], around 21,756 attack incidents took place during the year 2000 alone. According to the 2001 Information Security Industry Survey, attacks on web servers have doubled in the year 2001 compared to the year 2000 [19]. The findings of "2002 Computer Crime and Security Survey" confirm that the threat from computer crimes continues unabated [25].

Among all the servers, the web server is the most exposed server on the Internet and the majority of the attacks are web based. A web server is the front-end software that responds to requests via the HTTP protocol. While firewalls can prevent many kinds of Internet-based attacks, they must allow requests to legitimate services that the host site wishes to provide, such as web-based service. Since the requests to these services generally come from unknown or untrusted sources, some of these requests may be malicious in nature. Flaws in the web server software, web server configuration, and CGI scripts can permit malicious requests to get unauthorized access to confidential information or even cause damage to the computer system. In computer networked systems, the security of network is only as strong as its weakest link. If the web server is not secure, the network is no longer secure. Hence, security on the web is of paramount importance to everyone. It is essential to understand the causes for the vulnerability of the web server system before attempting to secure the web server. The web server's operating system, network, application programs and data are all vulnerable to attack.

The two general areas of vulnerabilities that potentially affect web servers are:

- The server may allow access to files located outside the file area rooted at the web server that is designated for WWW access. Intruders may be able to trick some web servers into returning system files such as the password file '/etc/passwd'.
- Most web servers support executable scripts that compute information to be sent back to remote users. This is the area of greatest vulnerability for a web server. The system often executes these scripts using input from remote users. This information is generally supplied via a fill-out form. If these scripts are not carefully written, intruders can subvert the scripts to execute arbitrary commands on the server system.

Attacks on the web can also be accounted for by the vulnerabilities in the following:

- Operating System:

Vulnerabilities in the operating system may be exploited by the attacker for gaining control over the system. For example, systems using SCO operating systems have vulnerability [11] in the home directories of the “dos” and “asg” user accounts. The vulnerability results from the fact that the default home directories for the "dos" and "asg" user accounts are /tmp and /usr/tmp respectively, both of which are writeable by all system users. This situation may allow unauthorized users to gain access to these accounts and the files that they own. The access may also be used to gain privileged access to the system.

- Network Service

Since the computers upon which web servers operate often run additional Internet services, a security breach in one Internet service could result in an attacker getting unauthorized control of the web server. For example, there exists a vulnerability in some versions of wu-ftpd which may allow local and remote users to gain root privileges. Due to insufficient bounds checking on directory name lengths that can be supplied by users, it is possible to overwrite the static memory space of the wu-ftpd daemon while it is executing under certain configurations. By having the ability to create directories and supplying carefully designed directory names to wu-ftpd, users may gain privileged access using buffer overflow [12].

- Web Server Software

Web servers are huge and complex programs that might contain bugs in themselves. They are also designed to be extensible by using modules, thus enabling connections to databases or other major programs. Moreover, addition of modules that are not properly implemented can also result in a server compromise. For example, there exists a vulnerability in Microsoft Information Server (IIS) in which a crafted HTTP GET request may return the contents of a file on the server [17]. The file might be a script that should only be executable and not readable by unauthenticated remote users. The contents of such a file might contain sensitive information such as user credentials for access to a back-end database. By viewing the contents of such a file, the attacker might gain confidential information.

1.2 Roadmap

This paper presents a classification and analysis of the web attacks and their countermeasures in Chapter 2. In Chapter 3, we study two security architectures and in Chapter 4, we propose a secure architecture design that integrates the features of mandatory access control (MAC) and resource control. In our prototype, the implementation is done using SELinux [4] as the platform and Apache [10] as the web server. In Chapter 5, we present the design and implementation of a set of tools that helps in the secure configuration of the web server running on an existing operating system. Finally, we describe our testing efforts in Chapter 6, followed by concluding remarks and future work in Chapter 7.

2 ANALYSIS OF WEB ATTACKS AND COUNTERMEASURES

2.1 Web Server Risks: A Classification

Server-side risks include the problems that occur due to vulnerabilities and/or misconfigurations of the operating system and the web server software.

These risks associated with these two systems can be again classified into four basic types:

- Arbitrary commands to be executed on the web server,
- Disclosure of confidential information,
- Denial of service attacks to be launched at the server, and
- System crashes.

By exploiting attacks in multiple risk types, it is also possible to gain information about a web server that may be used to break into the server system.

The attacks used to create the four classes of risks are explained in the remainder of this subsection:

Arbitrary Command Execution

In this class, the attacks are carried out by exploiting vulnerabilities in the applications. For example, buffer overflow can lead to the execution of commands at the remote server according to the attacker's wish. In buffer overflow, the attacker fills the stack with malicious code that should be executed. According to CERT, there exists a vulnerability in some IMAP servers and an attacker could overflow a buffer to execute arbitrary commands on the target site as the user running imapd, usually with root privilege [18]. Other attacks that come under this class are the ones that exploit CGI vulnerabilities, use hidden manipulation and execute Server Side Includes (SSI). These are explained in detail in Section 2.2.

Disclosure of confidential information

Attackers use methods like URL tampering, parameter tampering, cookie poisoning and forceful browsing to gain information that they are not authorized to view.

Denial of Service attacks

Denial-of-service attacks are attacks in which the attacker attempts to consume the resources at the client in such a way that it results in starvation of resources to the legitimate requests at the client. Attack of this type are explained in detail in Section 2.3.

System Crash

These include attacks that are carried out to cause improper functioning of the applications on the system that would result in crashing it. For example, in the buffer overflow attack, the attacker tries to input a larger value than that can be handled by the application. This causes the application to malfunction or sometimes crash the system.

The main focus of this paper is the denial of service attacks directed against a web server.

2.2 Server Attacks

2.2.1 Common Gateway Interface (CGI) Security

CGI is the interface used to provide interactivity between external applications and the web server. CGI programs are executed in real-time so that it can output dynamic information. Most of the web pages have HTML forms that use CGI to create interaction between the client and the server. When written without proper precautions, CGI scripts can lead to several different kinds of attacks. They can present security holes in two ways [14]:

- They may intentionally or unintentionally leak information about the host system that can be used in a break in.

- Scripts that process remote user input, such as the contents of a form or a "searchable index" command, may be vulnerable to attacks in which the remote user tricks the scripts into executing commands.

There are some CGI scripts that are installed along with the web server by default. One such CGI script is the test-cgi. The installation of some versions of this script can create a security hole in the system, as it indiscriminately gives out various system information, such as the directory in which the web server resides, the OS of the system and even the directory structure of the system.

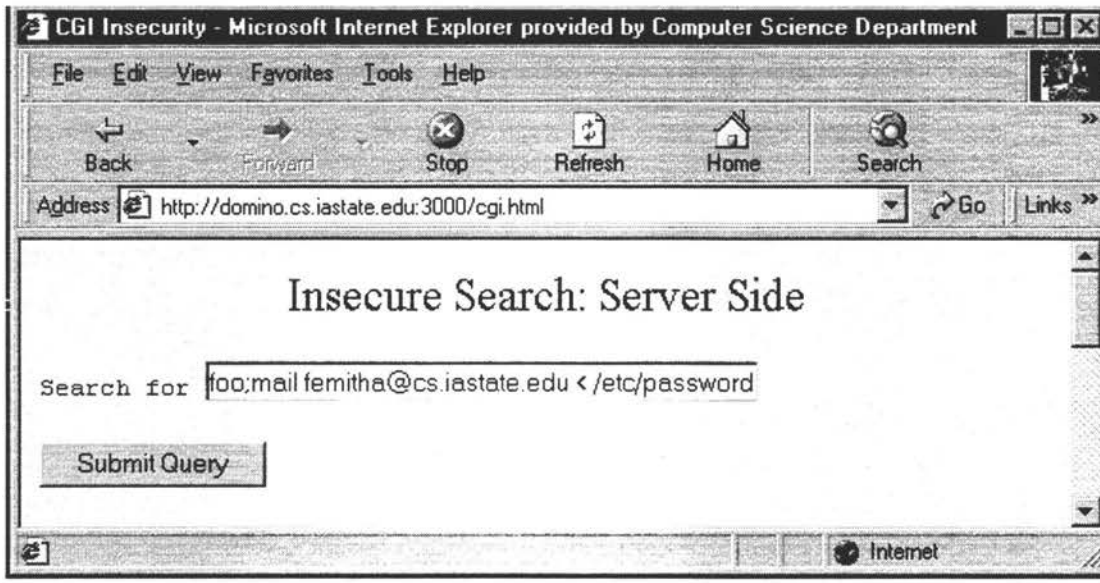


Figure 1: An example of CGI vulnerability

Shown in Figure 1 is a form filled out with user input that can be submitted to the web server. Upon receipt, the web server starts a program, usually a powerful interpreter program like Perl or a UNIX shell, and passes the user's input commands to that program. Perl or a UNIX shell can execute a wide variety of system functions. If a Perl script or a UNIX shell script receives the user's input, the risk is greater. A part of the cgi script in this example that gets executed upon clicking the submit button is:

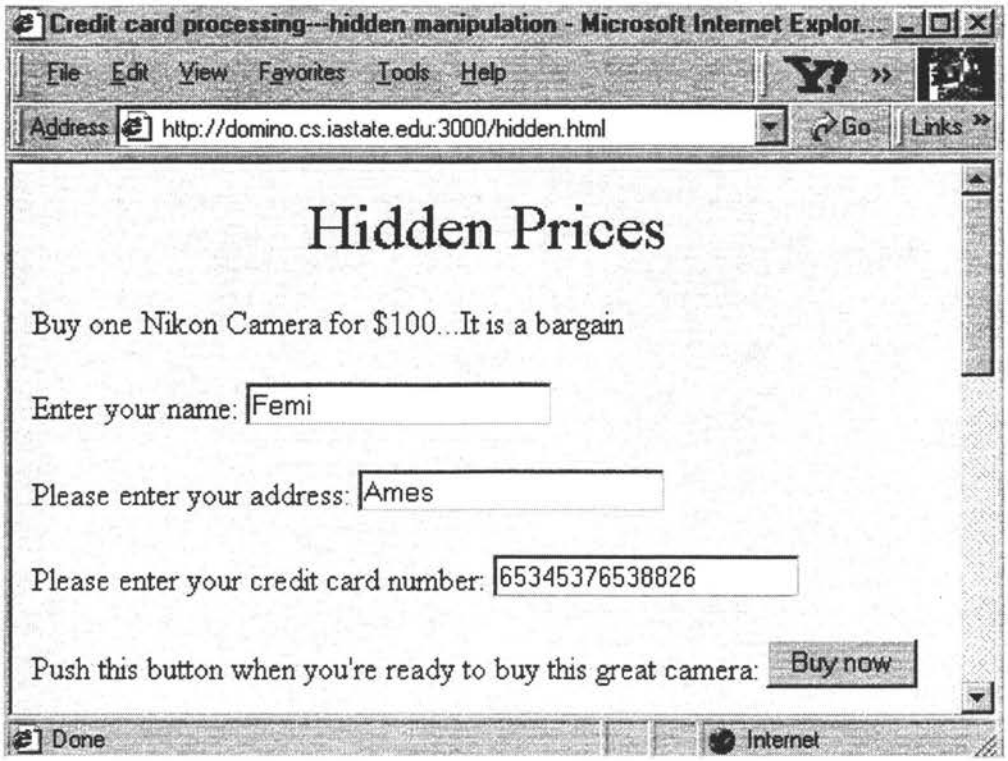
```
            :
            :
foreach $key (keys %in){
print "<li>$key: $in{$key}";
system(" find $in{$key}");}
            :
            :
```

In the above piece of code, the user's input causes the password file to be mailed to the email address specified.

2.2.2 Hidden Manipulation

This refers to the intentional manipulation of hidden fields in HTML forms. Hidden fields can be easily determined by viewing the source code of an HTML page. These fields could be altered by the attacker. The attacker can either execute the CGI script directly instead of filling in the form or he could save the page to a local file, change the values and send the new request.

Altering the values of hidden fields is unsafe especially in e-shops (online marketplaces) where prices are stored in hidden variables while a user proceeds with adding more items to his shopping cart. For example,



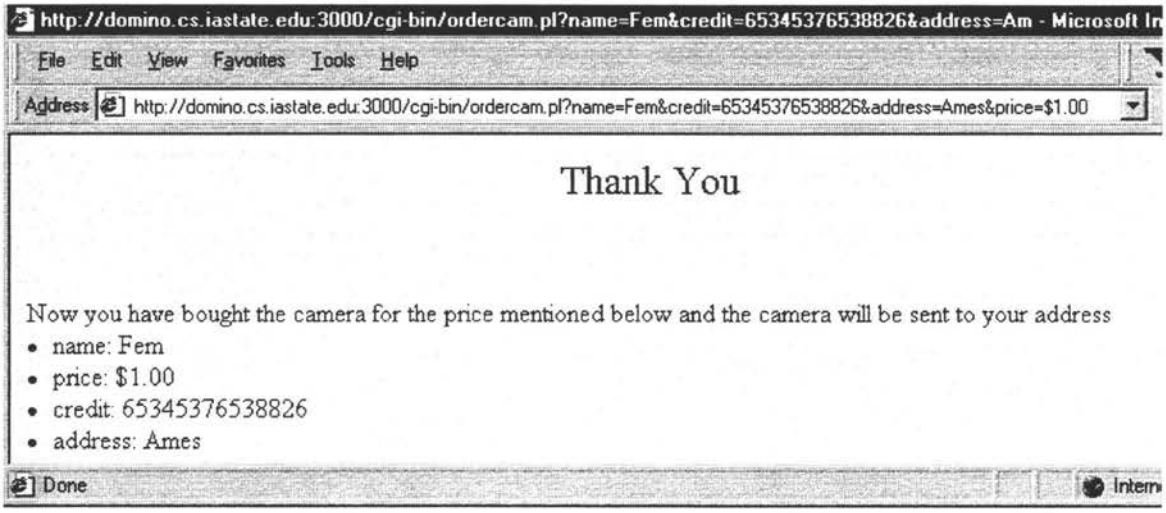


Figure 2: An example showing hidden manipulation

Figure 2 shows how a camera worth $100.00 can bought for $1.00 by manipulating the hidden variable, the price. The URL in Figure 2b shows how the attacker passes the

parameters to the CGI script. It can be seen that a value for the hidden field "price" is also passed.

This kind of attack could be avoided using the following precautions:

- Referral page: This will ensure that a given form was reached from the page that called the script. This checks that the script is not executed directly by a URL. However, we cannot rely entirely on referral checks because of two reasons: Since HTTP_REFERRER is a property of the browser, this could easily be changed by the attacker. Moreover, some users use anonymizers that result in no value for HTTP_REFERRER. An anonymizer is a privacy service that allows a user to visit web sites without allowing anyone to gather information about which sites they visit and without allowing a visited web site to gather information about them, for example, their IP address.
- Process and validate the form input field values entered by the user for range, expected input (e.g., numeric vs. alphabets), strange characters and any other associations specific to the user.

2.2.3 Buffer Overflow

This is one of the most common attacks that occur on the web. These attacks gained notoriety in 1988 as part of the Morris Worm incident on the Internet [26]. According to CERT/CC advisories, in the years 1998, 1999 and 2000, 58% , 53% and 11% of the attacks reported belonged to this category [19]. The decreasing trend is due to the employment of countermeasures.

Most web applications expect a specific size for its fields. Users may intentionally try to provide an input that is longer than the one expected. This might sometimes crash the application or even the system on which the application runs. This can be done in two ways:

- The malicious user tries to overwrite the return address if a function is involved.
- The malicious user tries to execute his own code by modifying the program stack.

This type of attack is especially dangerous when the compromised application has been configured with root permissions is compromised.

Security-problem buffer-overflows can arise in several situations:

- when reading input directly into a buffer;
- when copying input from a large buffer to a smaller one;
- when doing processing of input into a string buffer.

The strcat(), strcpy(), sprintf(), vsprintf(), bcopy(), gets(), and scanf() calls in C can be exploited because these functions do not check to see if the buffer that is allocated on the stack will be large enough for the data copied into the buffer.

An illustration of the attack is shown using Figures 3 and 4.

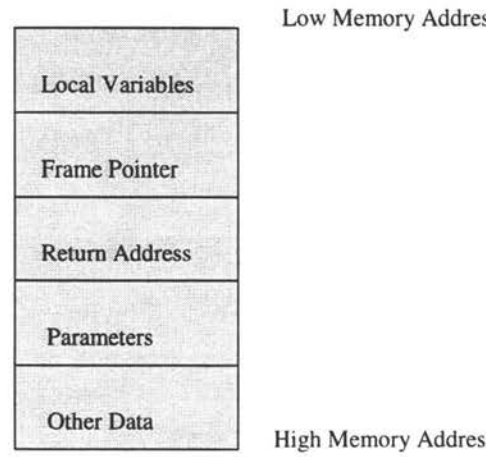


Figure 3: The original process stack in a Linux system

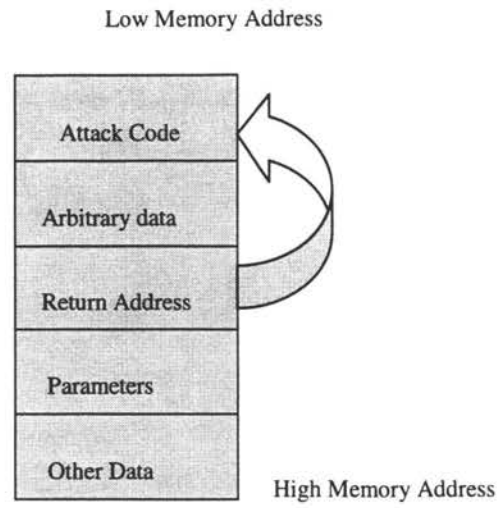


Figure 4: The stack from Figure 3 after it has been modified

The original process stack (in Linux) is as shown in Figure 3. It is possible to overflow the local variables in such a way that it can overwrite the return address. The attackers try to fill in the local variables area with the code they want to execute and make the return address point to the address of the local variable [7].

The following are some possible levels at which a defense mechanism against the buffer overflow can be inserted [15].
These mechanisms are briefly overviewed in the remainder of this subsection.

Language level

At the language level, we can use several techniques to protect against buffer overflow.

- Use a library module that implements "safe", bounds-checked buffers, such as the standard C++ string module
- Switch to a language that provides automatic bounds checking of buffers, such as Java, Perl or Python
- Target only those specific functions in C that are known to be dangerous, e.g. strcpy(), strcat(), gets() and sprintf(). These can be substituted by strncpy(), strncat(),snprintf() etc.

Source code level

A source code tool analyzes the source code of a program and tries to determine whether it contains any dangerous constructions that could lead to buffer overflows. Analysis tools such as *'purify'* analyzes the memory use of a program as it is run [20]. They can detect buffer overflow problems if they occur in a test run of a program. *Purify* is available for both UNIX and Windows.

Compiler level

A compiler tool changes the way a program is compiled, so that protection against buffer overflow is automatically compiled into the program. For example, StackGuard and StackShield are compiler level tools [21]. These tools use either of the ways given below to defend against the buffer overflow attack.

- Buffer overflows can be prevented by adding bounds checking to all buffers.
- The attack can also be prevented by protecting the return address from being overwritten.

Operating system level

The OS can also impose some restrictions that make it harder to use buffer overflow attacks. For example, a patch exists for Linux that makes the stack non-executable, thus making it impossible for the attacker to insert malicious code and execute it [22].

2.2.4 Parameter Tampering & Authentication Bypass

Parameter tampering refers to the intentional manipulation of URL parameters to access information that is otherwise accessed through a secure mechanism. Many times, form fields are used to send input provided by users to the back end web server for further processing.

For example, a user may input their user ID or full name to list information pertaining to their user account. Once the web server receives this information, it will issue a query to a relational database. The results of the query are displayed to the user. A malicious user may input field entries in such a way that the returned result provides additional information about other users as well.

Authentication bypass is the act of jumping directly to pages that are normally accessed through an authentication mechanism. For example, a vulnerability was reported in AdCycle that belongs to this category [16]. In the file AdLogin.pm, AdCycle uses the following SQL command to authenticate a user signing in:
SELECT * FROM ad WHERE LOGIN='$account' AND PASSWORD='$password'
If an attacker signs in, using an administrator account name of "ADMIN" and a password of X ' OR 1 # an attacker can cause AdCycle to use the following SQL command:
SELECT * FROM ad WHERE LOGIN='ADMIN' AND PASSWORD='X' OR 1 #'
The pound sign cause mySQL to ignore the trailing single quote. Since anything OR 1 is true, the query will return a recordset, and AdCycle will think that the attacker has authenticated as administrator. Administrator status allows one to modify the various ads.

2.2.5 Server Side Includes (SSI)

Server Side Includes are directives that can be placed in HTML documents to execute other programs or output such data as environment variables and file statistics. This causes the server to parse documents. It will affect both the performance and security of the server. It can be costly for heavily loaded servers to perform parsing of files while sending them. Further, it can be considered a security risk to have average users executing commands as the server's user. SSI also poses the same risks as those associated with CGI scripts. SSI can execute programs on the server with the "virtual" and "exec" directives. An attacker, who has the ability to place content onto a server, could include a malicious SSI directive. When the content of the directive is processed, it results in code of the attacker's choice running in local

system context. For example, suppose there is a guestbook on a website and a malicious user enters the following into the message.

<!--#exec cmd="mail attacker@bad.com </etc/passwd; rm -rf /"-->

If this message is written to an HTML file that is subsequently parsed for SSIs, each time the page is loaded, the command will be executed. Therefore, when users view the guestbook, their password files would be emailed to the attacker and all the files on their system would be removed.

2.2.6 Other attacks

Automatic directory listings

All directories under the web server root might not have default files like *index.html*. If permissions are not properly set on the directory, accessing that directory though the URL might give a listing of all the files and directories under the parent directory. This would give the attacker additional information about the files in the system. The attacker might exploit that information for future attacks.

Symbolic linkage

Some servers allow users to extend the document tree with symbolic links. This is convenient, but can lead to security breaches when someone accidentally creates a link to a sensitive area of the system. A safer way to extend the directory tree is to include an explicit entry in the server's configuration file.

2.3 Denial of Service Attacks on Web Servers

Denial-of-service (DOS) involves attacks in which the attacker attempts to consume the resources at the client in such a way that it results in starvation of resources to the legitimate requests from the client.

2.3.1 Classification of DOS Attacks

Denial-of-service attacks come in a variety of forms and aim at a variety of services. There are two basic types of the attack:

Consumption of Scarce Resources

Some of the resources that are consumed in excess are described briefly.

- Network Connectivity

  Denial-of-service attacks are most frequently executed against network connectivity. The goal is to prevent users from communicating on the network. An example of this type of attack is the "SYN flood" attack
- Bandwidth Consumption

  An intruder may also be able to consume all the available bandwidth on the network by generating a large number of packets directed to the network. Typically, these packets are ICMP ECHO packets, but in principle they may be anything.
- Consumption of Other Resources

  In addition to network bandwidth, intruders may be able to consume other resources that your systems need in order to operate such as CPU, memory etc.

Destruction or Alteration of Configuration Information

An improperly configured computer may not perform well or may not operate at all. An intruder may be able to alter or destroy configuration information that prevents the computer or network from being used. For example, if an intruder can change the routing information

in the routers, say, the hosts that are allowed or denied, the network may be disabled for certain users.

2.3.2 Some Examples of DOS Attacks

A DOS attack on a web server typically involves the misuse of standard protocols or connection processes with the intent to overload and disable the target web servers.

TCP SYN flood

The SYN Flood attack works by sending several SYN packets with fake source IP addresses to the target server. The server then allocates memory for the pending TCP connection. The source address cannot be an active IP address because if it were, that host would sent a RST (reset) message to the server, freeing the memory set aside by the initial SYN packet. The server then sends a SYN-ACK to the bogus IP address. The SYN-ACK message will time out and the server will send it again, keeping memory allocated for a longer period of time. If there are enough half-open TCP connections, the server will run out of memory and not be able to allow additional TCP connections and in some cases, crashing the server because there is no free memory.

Smurf

The smurf attacks involve sending a large number of ICMP/UDP echo (ping) messages to an IP broadcast address, with the forged source address of the intended target. All the hosts receiving the PING request reply to this target's address instead of the real sender's address. A single attacker sending hundreds or thousands of these PING messages per second can fill the victim's network with ping replies, thus consuming the network connectivity.

Host System Hogging

A host system hogging attack involves causing a program to run on the target system; a program that effectively ties up the CPU on the system, making it unavailable to other users.

2.3.3 Countermeasures

Some of the countermeasures of denial of service attacks are briefly described in this subsection.

Firewalls

Firewalls can be used as a countermeasure to DOS attacks in three ways.

- Firewall as a Relay: A connection to the host is established only after the three-way handshake is successfully completed. During an attack, the firewall responds to the SYN sent by the attacker; since the ACK never arrives, the firewall terminates the connection with an RST packet, and the host never receives the SYN. For legitimate connections, the firewall creates a new connection to the internal host on behalf of the client, and continues to act as a proxy for translating sequence numbers of packets flowing between the client and server
- Ingress filtering: Ingress filtering is the filtering of any IP packets with untrusted source addresses before they have a chance to enter and effect the system or network. This is best done at the ISP level where they can more cleanly handle the packets coming through their many networks [24].
- Egress filtering: This prevents attackers from using forged source addresses to launch a DOS attack. This is accomplished by an ingress filter at a router that checks for every packet's source address and verifies that the http request is originating from the network to which the filter provides connectivity [24].

Operating system improvements

Request Dropping: This admission control mechanism drops a pending request from a full connection request queue. The algorithm can pick a request at random, select the oldest

request, or use a combination of both, to deal with the request queue under attack. This has been implemented as a patch in Linux 2.0.21 [23].

Security Architecture

There is a three-step process called the Escort security architecture, to protect the Scout operating system against DOS attacks [2]. The three steps are:

- Accounting for resources consumed by every principal by allocating resources to each information flow.
- Detection of a DOS attack when a principal's consumption of resources exceeds levels allowed by the system policy
- Containment of an attack to reclaim consumed resources with as few additional resources as possible, for example, by terminating the information flow.

Cookies

When a client sends a SYN packet, the server calculates a one-way hash of the sender's sequence number, ports, the server's secret key, and a counter that changes every minute [29]. The server sends the result of the one-way hash to the client, and the connection is not established. When the client replies with an ACK packet, the server recalculates the same hash function and throws away the packet if it fails to authenticate with the server. If the packet is successfully authenticated, a connection is formed. If there is no reply with ACK packet from the client, no action is taken.

Stateless protocols

Stateful protocols have an upper limit on number of simultaneous connections, because there is a limited space available for storing connection state information [28]. When this limited space is exhausted, new connections are refused. This can be addressed by storing the state information on the client rather than on the server. To ensure integrity and confidentiality of state data and connection, the data stored on a client can be encrypted with the server's key.

Intrusion Detection

Intrusion detection methods are employed to spot and analyze misuse and anomalies in the network, and alert administrators of network attacks [27]. Intrusion Detection Systems (IDS) are systems that detect incorrect, inappropriate or anomalous activity. Intrusion detection systems that operate on a host to detect malicious activity on that host are called host-based intrusion detection systems, and those that operate on network data flows are called network-based intrusion detection systems. IDS detects intrusion by listening to all packets on a network in promiscuous mode. Network packets are next analyzed for rule violations by a pattern recognition algorithm. When rule violations are detected, the IDS may alert the administrator.

3 RELATED WORK

Two popular security architectures on which we base our proposed design are discussed in this section: the Flask security architecture developed by NSA and the Scout-Escort architecture by the University of Arizona.

3.1 Flask Security Architecture

Flask architecture is an operating system architecture capable of supporting a wide range of security policies [1]. It includes a security server to make access control decisions and object managers to enforce those access control decisions. The Flask Architecture is illustrated in Figure 5.

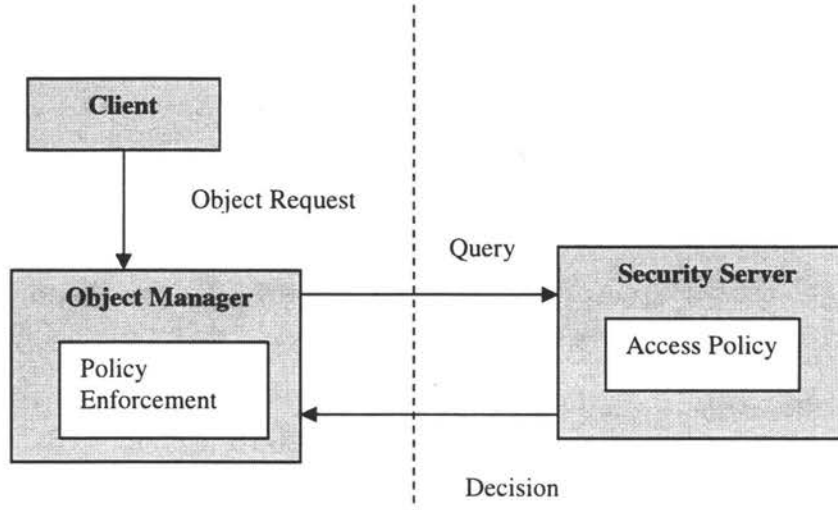


Figure 5: A block diagram of the Flask Architecture

The basic issues of each of the components in the Flask architecture are briefly overviewed.

Object Manager

The component that enforces security policy decisions is referred to as the object manager, which has interfaces for retrieving access, labeling and polyinstantiation mechanisms from a security server. Access decisions specify whether an object is allowed to access another object with a particular permission. Each object that is controlled by the security policy is labeled with a set of attributes called the security context. The labeling is used to describe the specific attributes to be assigned to an object. Polyinstantiation decisions specify which member of a polyinstantiated set of resources should be accessed for a particular request. Sometimes object managers base their decision on local knowledge, e.g, limiting a TCP module to just one port. Sometimes, they need to make policy decisions based on global knowledge. In such cases, the object manager consults the security server.

Security Server

The security server provides security policy decisions to the object manager. It provides interfaces for changing policies. It also enforces the policy changes on the object manager. The security policy encapsulated by the security server is defined through a combination of code and a policy database. Any security policy that can be expressed through the prototype's policy database language may be implemented by altering the policy database. The security server can be accessed only through the object manager.

The propagation of access rights is controlled by ensuring that the security policy is consulted for every access decision. Enforcement mechanisms enable fine-grained access controls and dynamic policy support allows the revocation of previously granted access rights. Dynamic policy support is possible by coordination between the security server and the object managers whenever there is a policy change. This is achieved by imposing two requirements on the system: The first is that after completion of a policy change, the behavior

of the object manager must reflect that change. The second requirement is that object managers must complete policy changes in a timely manner.

The limitation of the Flask architecture is that it provides no resource accounting thus making it vulnerable to denial of service attacks.

3.2 Scout-Escort Security Architecture

Escort is an architecture based on the Scout operating system that defends against denial of service attacks [2]. The architecture accounts for all resources consumed by each information flow in the system. This accounting mechanism is implemented as an extension to the path object in the Scout operating system that is described in detail below. Escort simultaneously does end-to-end resource accounting and supports multiple protection domains. The Scout-Escort architecture is shown in Figure 6.

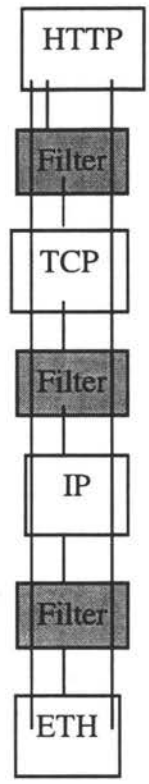


Figure 6: The module graph in the Scout-Escort architecture

Module Concept

The Scout architecture provides modularity by configuring different modules, which are units of program development and configurability that provide well-defined and independent functionality. Modules are connected to form a module graph. All the connections are through well-defined interfaces. A module graph for a web server system is shown in Figure 6, where each of the boxes represents a module.

Path Abstraction

A path is viewed as a logical channel through a modular system over which I/O data flows. The path is created by the path manager, which allocates resources to the path during its creation.

Filters

One of the security mechanisms that Escort adds to Scout are filters, which are modules that are automatically generated from a global policy engine. Filters restrict the interface between the adjacent modules.

Escort implements resource management by controlling the information flow. Resources are allocated on a per-path basis. This ensures that no path uses excess resources, thus defending against denial of service attacks. The limitation of the Escort architecture is that it assumes that the security policy is static.

3.3 Remarks

The first phase in securing a web server consists of securing the operating system on which the server is to be run. This is important because compromising a flaw in the operating system might lead to an attack on the web server. The second phase is securing the web server software. Securing a web server on an insecure operating system can often prove to be unsuccessful. This leads us to consider structuring an operating system architecture that is specially configured for a web server. The Chapter 4 describes the proposed security architecture design that integrates resource control, management and accountability and mandatory access control (MAC).

4 PROPOSED ARCHITECTURE

Section 4.1 describes some of the objectives of our proposed architecture. The design of the architecture is discussed in Section 4.2 followed by the implementation in Section 4.3.

4.1 Security Architecture Goals

A secure operating system design should fulfill the following three objectives:

Fine-grained access control

Fine-grained access control ensures that unless a permission is explicitly granted to an object, it cannot access the resource that is guarded by that permission. As a result there does not occur any malicious attacks that exploit the access permissions. When a path is created, it is assigned permissions based on the security policy currently in effect. Each access permission specifies a permitted access to a particular resource. This feature is essential in a secure web server.

Flexible security policies

Security requirements for different applications and computing environments widely vary. To provide for the varying security needs, we need a number of different security policies to address each of the security needs. Therefore, the security architecture must be flexible enough to accommodate a wide range of security policies.

Protection of information flow

Most of the attacks on the web are caused by denial of service. A denial of service attack occurs when the attacker's processes take up a significant portion of computer resources that leads to starvation of resources for legitimate processes. Protection of information flow helps in defending against denial of service attacks by employing proper resource allocation and

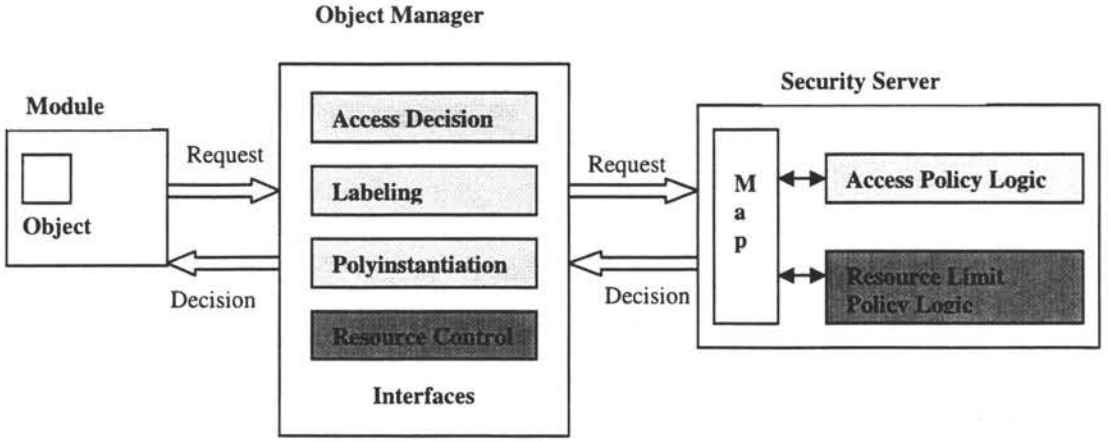
management. Each process is monitored to check if it exceeds its resource usage. If it exceeds the resource usage, an appropriate action is taken based on the security policy.

4.2 Proposed Security Architecture

The proposed security architecture design aims to address the limitations of the Flask and Escort architectures. This is achieved by integrating the features of resource control, management and accountability into mandatory access control architecture.

4.2.1 Design

As in the Scout architecture, our architecture provides modularity by configuring different modules. All the connections between modules are through well-defined interfaces. Modules can be considered to consist of objects that are managed by an object manager. The object manager is responsible for all security decisions related to the objects in the modules. However, global security decisions are made by a global component called the security server. To integrate resource control policy into the object manager and security server, the interfaces of an object manager and security server have to be clearly defined. Figure 7 shows an object manager and security server in more detail. An addition to the original security server of Flask architecture is shown in Figure 7 by a darkly shaded box. It represents the security server's internal policy to express concepts of resource control. The parameter list shows the several possible parameters contained in a request to the object manager by a client object. The four shaded boxes in the object manager are the interfaces. An additional interface that is added by the proposed design is the interface for resource control policy. It is shown as the darkly shaded box.



Request from Object:
Parameters (Client SID; Module1 Label; Object SID, Module2 Label; Permission requested)

Figure 7: Object Related Decision: Access/ Resource limit

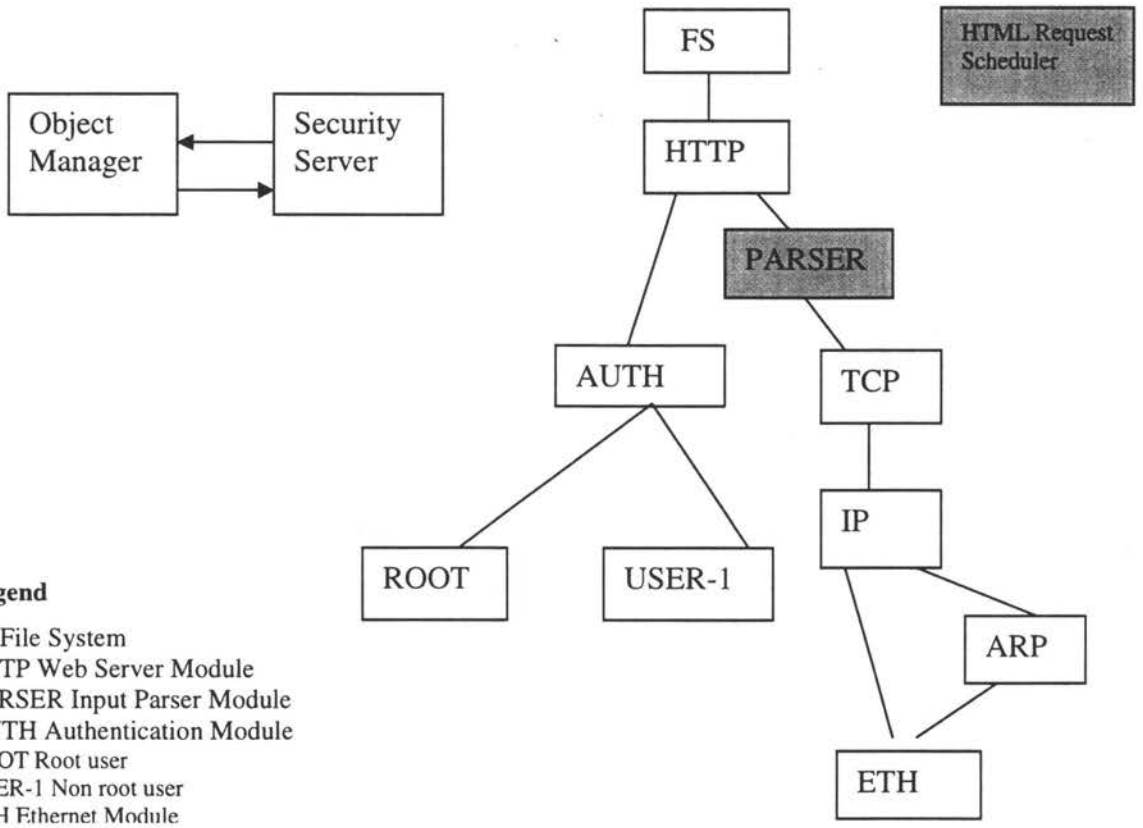


Figure 8: Web Server Security Architecture Layout

Figure 8 shows a web server security architecture layout. Each of the boxes represents a module. Any data or information flow through the modular system can be considered as a path. This idea is borrowed from the path abstraction feature in Scout. Resources are allocated to the path during its creation. Each path traverses through the modules from its source module to its destination module. During the traversal, one object in one module may need to access another object in another module. All object access decisions are taken by consulting the object manager. The object manager consults the security server and the security server sends its decision back to the object manager. The object manger then enforces the decision. The object manager may or may not restrict the data flow according to whether path meets or does not meet all the security policy specified by the security server.

This architecture provides fine-grained access control and protection of information flow. For example, each HTML request to the web server can be considered as a path. Whenever an HTML request has to access a file, the object manager is consulted. The object manager consults the access control policy of the security server. Based on the response of the security server, the httpd process is permitted to access the file. The security server also checks the resource control policy to see if the httpd process has exceeded its resource usage or not.

Protection Domains

The architecture can also be viewed as consisting of logical protection domains within the system as shown in Figure 9.

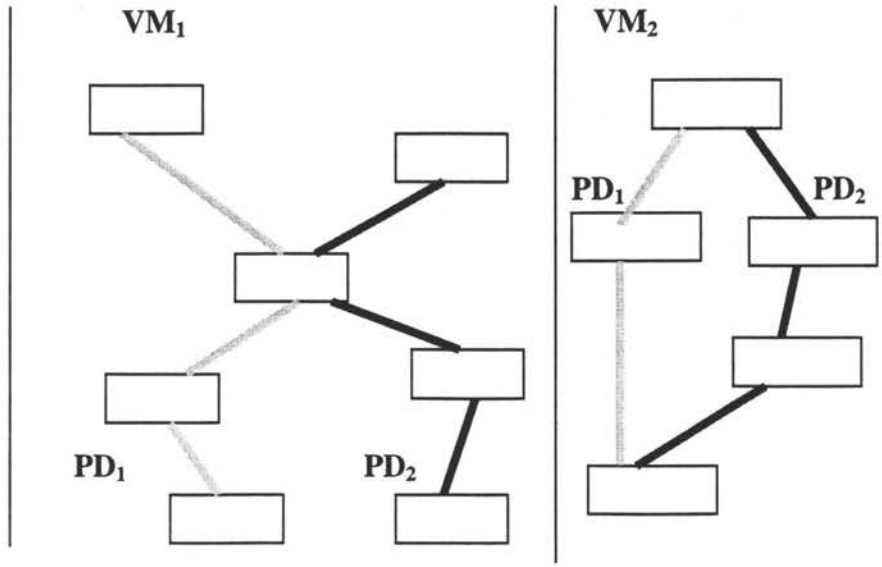


Figure 9: Virtual Machines (VM) and Protection Domains

The boxes represent modules and the shaded path shows two different paths. These paths can be considered as protection domains. The idea can be extended to a system with many virtual machines, each of which can be considered as a protection domain. Each of the protection domains can have many protection sub-domains. If a path exceeds its resource usage, it will be terminated. This may or may not affect the other paths (protections domains) in the same virtual machine, but it will definitely not affect the paths in the other virtual machines.

4.2.2 Evaluation

The resulting architecture has the following notable advantages:

- No object can access a resource unless the permission is explicitly granted to the object. Hence, the access control is fine grained. Fine-grain access control can thwart many web attacks that try to gain full control of the system. This can satisfy our first goal of the security architecture.

- The architecture separates policy enforcement from the modular structure, thus enabling flexible security policies. This meets the requirement of our second goal of the security architecture.
- Path abstraction and resource allocation enables resource accounting for each path and defense from denial of service attacks. This can fulfill the third goal of our proposed architecture.
- The creation of logical protection domains ensures one module does not bypass the interface to another module and access its memory. This can limit the affect of the attacks.

4.3 Prototype

Our prototype is based on the Security Enhanced Linux (SELinux), which is the Flask architecture incorporation on Linux from NSA [30]. The prototype goal is to incorporate the features of Scout architecture into SELinux. The main issues are discussed in the next section.

Modularization

Scout has a modularized system. One goal is to group related objects in SELinux into modules. For a simplified version of the proposed design, each module is considered to be consisting of one object.

Modification of the access query

Since objects are inside a module, every access query to the object manager will have two additional parameters.

The original access query in Scout is of the form:

$$\text{access\_query}(f_1, f_2, \text{oper})$$

The modified access query is of the form:

access_query(f_1,m_1, f_2, m_2, oper)

f_1 is the security identity of the file that does operation *oper* on the file whose security identity is f_2. m_1 is the security identity of the module in which f_1 resides and m_2 is the security identity of the module in which f_2 resides.

Since each module is considered to consist of a single object, it eliminates the need for interfaces between modules and eliminates the need for modifications to the access query.

Path Abstraction

Scout has path abstraction built into it. Each information flow is considered as a path. In our prototype, each process is considered as a path.

Resource accounting

The primary resources that need to be accounted for are CPU time, number of memory pages and number of files opened. Any running process that seems to take up more resources than its maximum allocation of the resource, might lead to a denial of service attack by starving all other processes that uses the same type of resource. A solution to this is to keep track of resource usage and notify the system administrator or terminate the process that crosses the limits.

The first attempt was to implement a user application that would check for resource usage of every process running on the system. But a user application has security problems as it could easily be terminated. As a result, this idea was abandoned.

The next attempt was to implement a kernel module that would execute the same procedure. Since it is a kernel module, it could be loaded and unloaded dynamically. It is also more secure as it uses the kernel space. It was to be decided when to call the module that checks the process resource usage. Finally, we decide to call the function that implemented the above procedure at every system call to opening files, allocating data segment space, writing to files and at every socketcall. This ensures that the function is called at sufficient intervals

to detect the excessive usage of resources that can lead to denial of service attack and take appropriate action.

A resource usage check is performed during the following operations:

- When an open system call is invoked to open a file.
- When a write system call is invoked to write to a file.
- When a brk system call is invoked to allocate dynamic memory.
- When a socket system call in invoked for socket communication.

This is done by intercepting the open, write, brk and socket system calls.
Figure 10 shows the control flow when an open system call is invoked in an application.

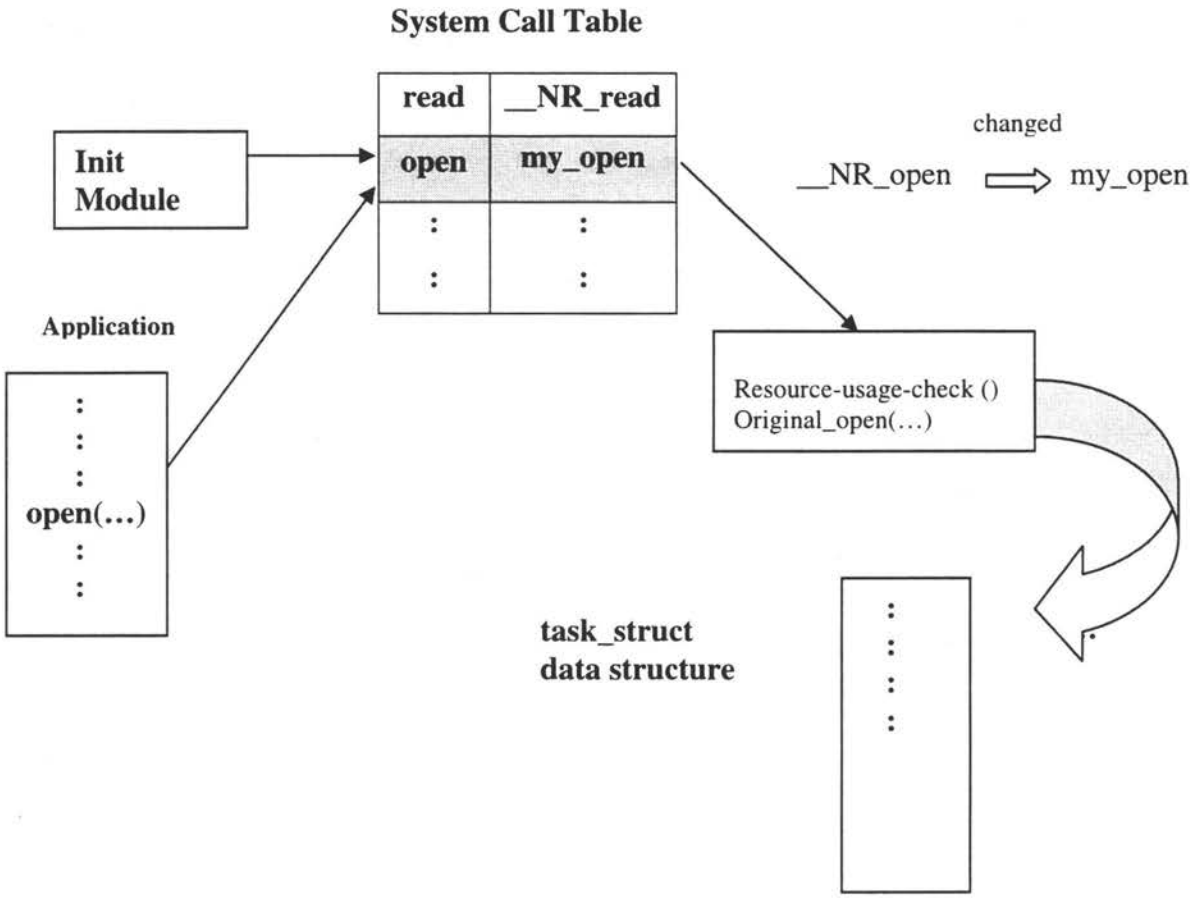


Figure 10: Intercepting open system call to perform resource check

When the kernel module is loaded, it modifies the system table entry corresponding to open system call by replacing the original call with the modified open system call. As a result, when an open system call is invoked in an application, the control flows to the modified open system call. The modified open call performs the resource allocation check before calling the original open system call. The resource usage check is performed as follows:

The implemented program collects all process related information from the task_struct data structure located in linux/sched.c. Each process's context is described by a *task_struct* structure. The task_struct holds data such as the scheduling policy, scheduler priority, real time priority, processor allowed time counter, processor registers, file handles (files_struct), virtual memory (mm_struct).

The data structure used in the implementation is shown below:

```
struct task_struct {
volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */
long counter;           /* program counter */
long priority;          /* priority of process */
unsigned long signal;
unsigned long blocked; /* bitmap of masked signals */
unsigned long flags;    /* per process flags, defined below */
int errno;              /*fault information */
long debugreg[8];  /* hardware debugging registers */
struct exec_domain exec_domain; /execution domain*/
struct linux_binfmt *binfmt;
struct task_struct next_task, prev_task;  /*doubly linked list */
struct task_struct *next_run,  *prev_run;
unsigned long saved_kernel_stack;
unsigned long kernel_stack_page;
int exit_code, exit_signal;
unsigned long personality;
int dumpable:1;
int did_exec:1;
int pid; int pgrp;  /*process id and group id */
int tty_old_pgrp;
int session;
int leader; /*boolean value for session group leader */
int groups[NGROUPS];                        /*links to processes*/
struct task_struct *p_opptr, *p_pptr, *p_cptr, *p_ysptr, *p_osptr;
struct wait_queue *wait_chldexit;     /* for wait4() */
unsigned short uid,euid,suid,fsuid; /*user info*/
unsigned short gid,egid,sgid,fsgid; /* group info*/
unsigned long timeout, policy, rt_priority; /*scheduling info*/
```

```
unsigned long it_real_value, it_prof_value, it_virt_value;
unsigned long it_real_incr, it_prof_incr, it_virt_incr;
struct timer_list real_timer;
long utime, stime, cutime, cstime, start_time; /*cpu usage */ /*swap info*/
unsigned long min_flt, maj_flt, nswap, cmin_flt, cmaj_flt, cnswap;
int swappable:1;
unsigned long swap_address;    /*swap address*/
unsigned long old_maj_flt;     /* old value of maj_flt */
unsigned long dec_flt;      /* page fault count of the last time */
unsigned long swap_cnt;  /* number of pages to swap on next pass */
struct rlimit rlim[RLIM_NLIMITS]; /*limits*/
unsigned short used_math;
char comm[16];         /*process name*/
int link_count;
struct tty_struct *tty; /* NULL if no tty */
struct sem_undo semundo; /*ipc info*/
struct sem_queue *semsleeping; /*ipc info*/
struct desc_struct *ldt;
struct thread_struct tss;  /* thread structure */
struct fs_struct fs;      /* file system info*/
struct files_struct files; /*open file information*/
struct mm_struct mm;   /* virtual memory management structure*/
struct signal_struct sig; /*signal handlers*/
int processor;   /*processor info*/
int last_processor;
int lock_depth; /*Lock depth*/
};
```

Remote administration

We have developed a mechanism by which resource accounting on a host can be done from a remote host. A snapshot of the GUI is shown below:

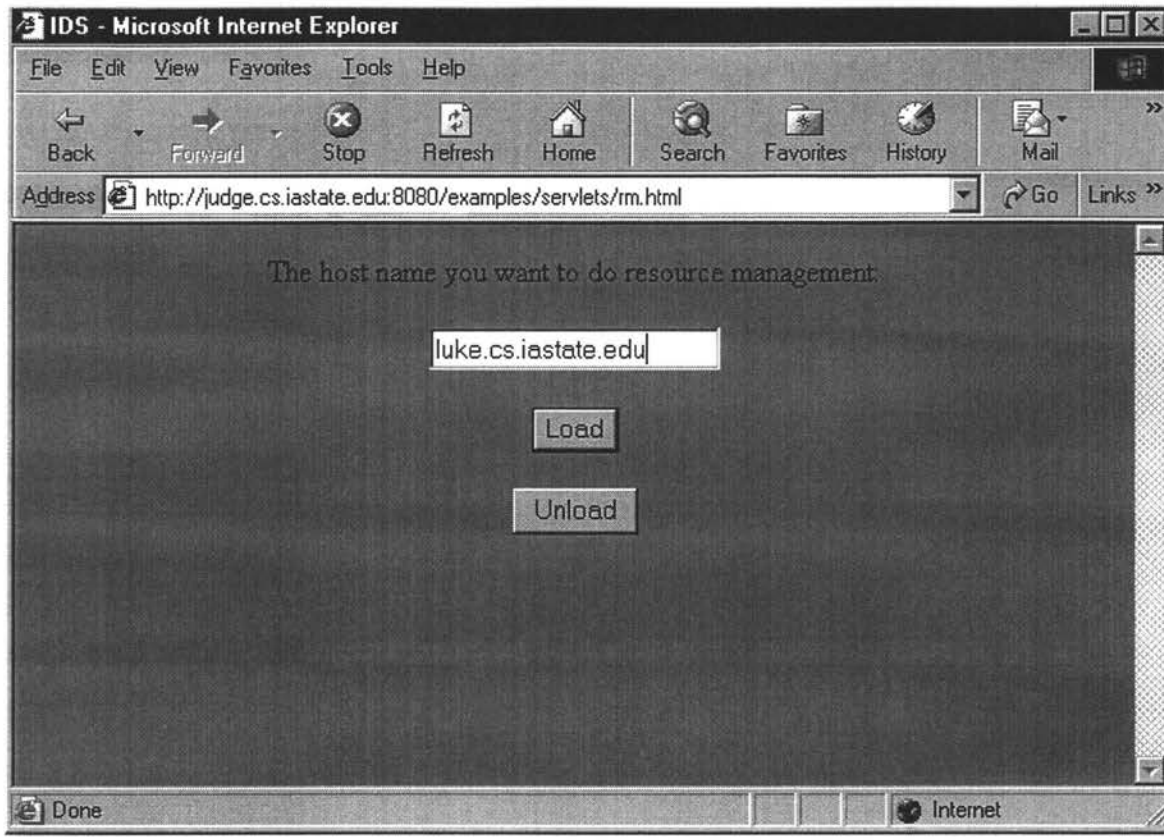


Figure 11: Remote resource accounting GUI

The hostname on which the module is to be loaded or unloaded is entered in the text field and the appropriate action is taken.

MAIDS Integration

The Mobile Agent Intrusion Detection System (MAIDS) project is an ambitious attempt to create a framework for an intrusion detection system that is efficient, fault-tolerant, modularly compatible with other detection systems, relatively simple to use and configure, and uniquely related to a disciplined requirements engineering process [9]. In the resource accounting mechanism, all the processes that use excess resources are logged in a file. Whenever there is a denial of service attack by host hogging, the MAIDS software gets alerted, by periodically checking the status of a global variable which is set by the resource accounting kernel module.

5 DESIGN AND IMPLEMENTATION OF CONFIGCHECK TOOL

ConfigCheck is a collection of tools consisting of bash shell scripts and perl modules that help in configuring a secure web server that runs on an existing operating system. It takes both detective and corrective measures on improper configurations of the system on which the web server is running as well as the misconfigurations of the web server itself.

5.1 Design

A web server running on an OS can be as shown in Figure 12. Vulnerabilities may exist at any level of the entire system.

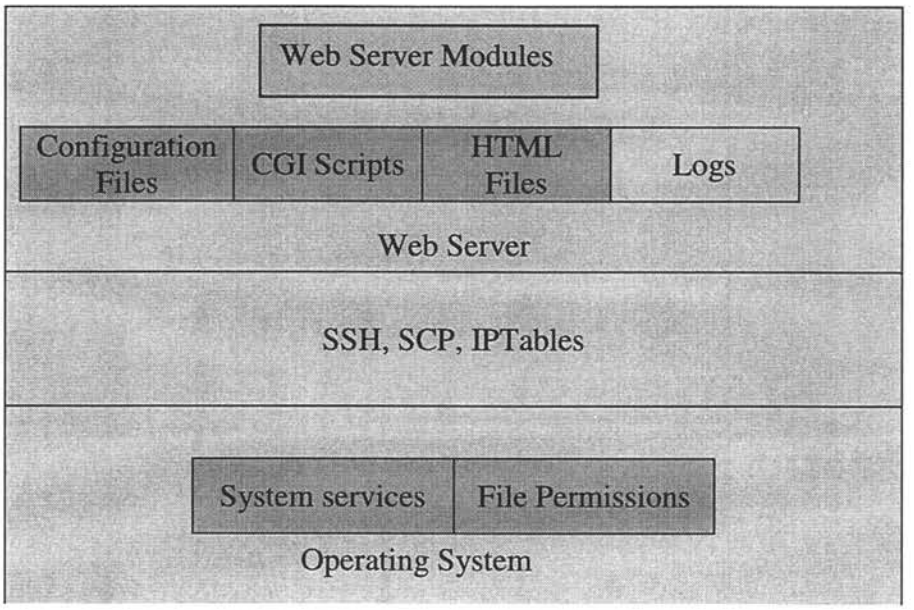


Figure 12: Computer System

ConfigCheck operates according to a security policy. An example security policy is as shown below.

Table 1: Example Security Policy

1) Run server with least privileges, in other words, the server's user and group should have minimum access to the rest of the system
2) Limit services running with super user permissions
3) Check all data entered by the user to see if they match the input expected
4) Restrict access of external programs to those that are absolutely necessary
5) Web content files should only be read, not written by web processes
6) Server configuration files cannot be read by end users
7) Disable unwanted programs, scripts and services
8) Disable file directory listings
9) Disable viewing of external files through web requests
10) Integrity check for all important system files and server configuration files

Each of the policy is briefly explained.

1) Apache should not be run as root because if a user exploits a flaw in a script running with root's permissions that allows them to write to files, he can either gain access to the system or crash the system.

2) "Set-UID root" programs run as root user regardless of who is executing them and they are a frequent cause of buffer overflows. Many programs are setuid and setgid to enable a normal user to perform operations that would otherwise require root, and can be removed if the users do not need such permission.

3) The user might try to enter malicious commands to gain unauthorized access into the system.

4) Disabling or removing programs that are not needed is one way of reducing threats caused by these programs.

5) If web content files are writable by everyone, any user can damage the contents of the files.

6) Server configuration files reveal a lot of information about the server. So they should not be readable by web users.

7) Unwanted programs or scripts may have a vulnerability or may help a user to break into the system.

8) File directory listings provide information about the directory structure of the system. This information can be exploited by an attacker.

9) Files that reside outside the web document area should not be viewed by web users for the same reason as above.

10) An integrity check of all important files can help the administrator to check whether they have been modified or not.

5.2 Implementation

The implementation is based on the Apache web server running on SELinux.

The Security Enhanced version of Linux (SELinux) is chosen as the platform for this work. SELinux has a strong, flexible mandatory access control architecture incorporated into the major subsystems of the kernel. The system provides a mechanism to enforce the separation of information based on confidentiality and integrity requirements. This addresses the threats of tampering and bypassing of application security mechanisms and confines the damage that can be caused by malicious or flawed applications.

Apache web server is used in the implementation because it is one of the most popular http daemons, distributed with many variants of the UNIX Operating System and maintained by the Apache Project. Popular surveys show that more than 62% of the world's web servers are Apache servers. The implementation as shell scripts and perl modules is based on the example security policy shown in Table 1.

5.2.1 Operating system level

Unwanted Services

The primary holes in a computer system are applications or services that have vulnerabilities in them. Disabling or removing unused programs and services from the system is one of the most effective ways to limit threats originating from a remote system. The implementation checks for the services given in Table 2 and disables them based on the administrator's choice to disable them or not.

Table 2: System Services

telnet	tftp	finger	daytime	chargen
systat	netstat	rlogin	rsh	talk
ntalk	dtalk	pop-2	pop-3	imap
smtp				

File Permissions

setuid /setgid programs are those that are invoked by the local user, and when executed, are immediately granted the privileges of the program's owner or the program's group. The implementation audits the system for programs using setuid or setgid permissions. It removes setuid and setgid permissions of a program according to the system administrator's choice. The script also modifies characteristics of files and directories such as immutable and append-only.

Changes ownership of important files to root

The permissions of log, conf and bin directories are set such that they are modifiable only by root. If non-root users are allowed to modify any files that root either executes or writes on

then the system is open to root compromises. For example, if the logs directory is writeable (by a non-root user), an attacker could replace a log file with a symlink to some other system file, and then root might overwrite that file with arbitrary data. If the log files themselves are writeable (by a non-root user), then someone may be able to overwrite the log itself with bogus data.

Remote file viewing

Apache configuration file httpf.conf might contain an entry (Alias /cgi-bin-sdb/ *apachehome*/cgi-bin). As a result, files in /cgi-bin/ can be accessed via URLS of the form http://target/cgi-bin-sbd as well. Because the path does not contain the string /cgi-bin/, improper permissions will be assigned, and the file itself will be sent instead of being executed at the server. This makes it possible to view the source code of CGI scripts stored in /cgi. The implementation checks for this misconfiguration and alerts the system administrator if it is present.

5.2.2 Web Server Level

Checking for eval statement

The eval statement will evaluate its command line in exactly the way the shell would evaluate it. If an attacker sends malicious commands to a cgi script that uses eval command to get data back to the stdout, it can prove dangerous. The implementation checks for the eval statements in all the cgi scripts under the cgi-bin directory.

Validating HTML data and hypertext links

There are two dangers with browser input data:

- Input containing special characters such as '!' and '&'could cause the web server to execute an operating system command or have other unexpected behavior.

- User input stored on the server, such as comments posted to a web discussion program like a mailing list, could contain malicious HTML tags and scripts. When another user views the input, that user's web browser could execute the HTML and scripts.

The presence of special characters and html tags are checked by a Perl module in the implementation. The Perl module deletes such special characters and html tags thus averting an attack.

Table 3 shows how ConfigCheck performs countermeasures to the attacks that were explained in Chapter 3.

Table 3: Attacks & Countermeasures

Attack / Vulnerability	Countermeasures
CGI Scripting	Check if server is run as nobody Check for dangerous commands, e.g. eval Remove default CGIs with known vulnerabilities Parse input data entered by the user Turn on Perl taint checks
Hidden Manipulation	HTTP_REFERER Check
Cross Site Scripting	Parse URLs and input data
Buffer Overflow	Check for fields in HTML forms without a size specified
Automatic Directory Listings	Set directory permissions in server configuration file
Symbolic Links	Turn off that feature in the configuration file
Parameter Tampering & Forceful Browsing	Validate input fields used for illegal characters

6 TESTING

This Chapter describes the tests that were done on the secured web server. The first series of tests were done to check if the kernel module implemented could detect and take action against the processes that exceeded their resource usage. Then the module was loaded and unloaded from a remote machine using a graphical user interface. The third test was to check whether the resource accounting program could be integrated with MAIDS. This was done by checking whether MAIDS software alerts the administrator when a denial of service attack happens. Finally, tests were done to determine the effect of exploiting CGI vulnerability, hidden manipulation, forceful browsing etc.

6.1 Denial of Service by Excessive Resource Usage

Memory Usage

A program that allocates memory in a loop is run. This causes denial of service attack to the other programs running on the machine as it consumes memory at a very fast pace without releasing or freeing it. Our software detected the process that exceeded the memory usage and terminated it. The memory limit specified and the corresponding time it took for the module to detect the excess usage is shown in Table 4.

Table 4: Memory usage and Detection time

Limit (in KB)	Time taken to detect (in secs)
2	1s
5	2s
20	5s

CPU Usage

HTTP connection requests to the web server were sent in a forever loop to open connections at a very fast rate. The kernel module detected the httpd processes that exceeded their CPU usage and terminated the malicious processes.

Table 5: CPU limit and Detection time

Limit (utime+stime)	Time taken to detect (in secs)
50	55s
70	1m 20s
200	3m 28 s

Number of open files

We execute a program which open files in a loop without closing them. This causes denial of service attack to the other programs running on the machine as it opens files at a very fast pace without closing them. The implementation detected the process that exceeds the number of open files and terminated it.

Table 6: Open file limit and Detection time

Limit	Time taken to detect (in secs)
100	11s
200	20s

500	51 s

The kernel module was loaded from a remote host *judge* on *frappe*. The MAIDS server and client were both run on *judge* and *mayoserver* while *frappe* was used as a proxy. The denial of service attack on *judge* was detected by the module and the MAIDS software reported it to the system administrator. Figure 13 shows the test configuration.

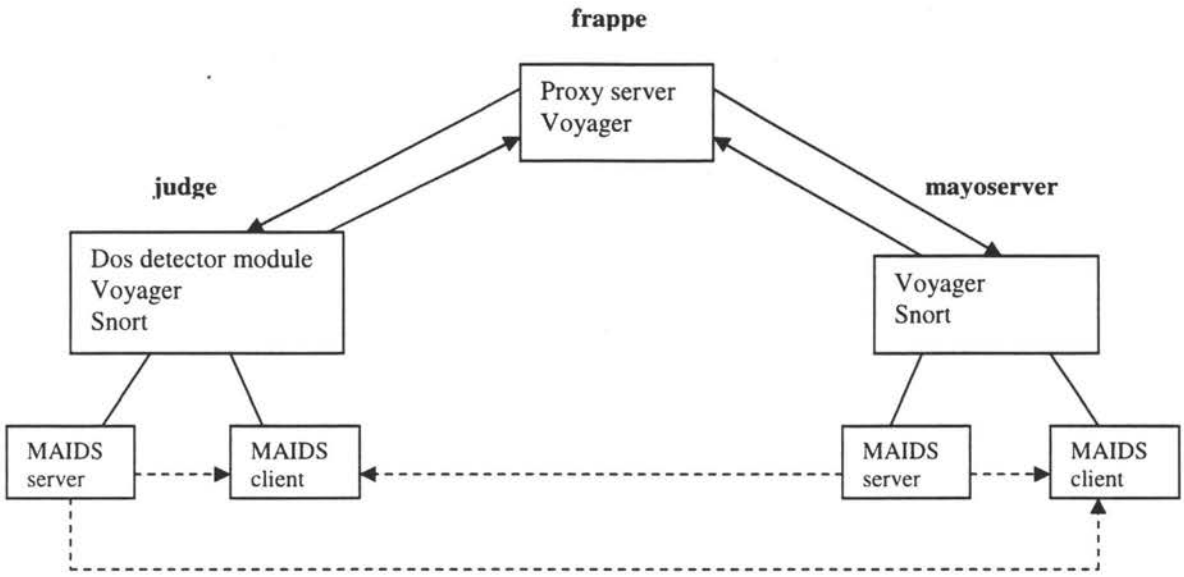


Figure 13: MAIDS Integration

Performance Overhead Testing

The web server was run with and without the resource accounting module in order to test the overhead. The time taken to process a request in both cases is shown in Table 7.

Table 7: Comparison of performance

Without module	480 milliseconds
With module	590 milliseconds

The amount of time taken to process a web server request when the resource accounting module is loaded is greater than the time taken to process a web server when the module is not loaded. This is due to the overhead of checking the resource usage of every process on the system whenever there is an open system call, write system call, brk system call and/or socket system call.

6.2 Arbitrary Command Execution

CGI script exploitation

The demonstration shown in Figure 1 that exploits the CGI vulnerability was tried out. The secret file was not mailed to the attacker. This is due to either of the two reasons:

- Apache server is running as nobody, which has limited privileges.
- SELinux mandatory access control is enforced.

Cross-site scripting

The cross-site script attack was not successful. This is due to the following reason. Every user input goes through a parser. When user inputs data that contains html tags such as <script>, the parser removes some characters that would make it a non-html tag, for example, the parser would convert '<script>'to 'script'.

Hidden Manipulation

The demonstration shown in Figure 2b was tried out. The attack was unsuccessful as it said that the page was reached through a bad URL. Attacks falling under this category are prevented by the proper inclusion of HTTP_REFERRER checks.

6.3 Disclosure of information

Parameter tampering

This attack was also unsuccessful due to the parser checks and referrer checks. The parser check removed all unexpected characters from the URL typed. The referral check made sure that the current web page was reached from the expected web page.

Forceful Browsing

When attempted to view directory listings, it resulted in an error message saying that we are not authorized to view the information. This is due to the directory permissions set in the web server configuration file.

7 CONCLUSION AND FUTURE WORK

7.1 Conclusion

The paper presents an analysis of some common attacks against a web server. Then it focuses on ways to secure a web server. It follows a two-phase method to secure a web server: the first phase consists of securing the operating system and the second phase consists of securing the web server software. Securing a web server on an insecure operating system can often prove to be unsuccessful. Therefore we consider structuring a security architecture specially configured for a secure web server. The two main issues that should be considered when building a security architecture is that it should be flexible to a variety of security policies and that it should help to defend the system from DOS attack which is the most common attack aimed at a web server system. DOS attacks can be addressed to a large extent by using a proper resource control mechanism. We propose a security architecture design that integrates resource control and accountability into Mandatory Access Control (MAC) architecture. The implementation incorporates resource control into SELinux, which has MAC built into it. We also develop a graphical user interface for remote administration. The resource control module is integrated with Mobile Agent Intrusion Detection System (MAIDS). Finally, we present the design and implementation of ConfigCheck that checks for some misconfigurations of the web server and the operating system on which it is run. We find that this set up successfully detects denial of service attack caused by excess use of resources like cpu, memory, number of open files etc.

7.2 Future Work

For future work, we would like to implement a denial of service detector that checks for resources like network bandwidth and tcp connections. This could be done by monitoring the TCP/IP connection table. We would also like to extend our work to detecting distributed denial of service attacks (DDOS). DDoS attacks involve breaking into hundreds or thousands of machines all over the Internet. Then the attacker installs DDoS software on them, allowing them to control all these burgled machines to launch coordinated attacks on victim sites.

These attacks typically exhaust bandwidth, router processing capacity, or network stack resources, breaking network connectivity to the victims.

8 REFERENCES

[1] R Spencer, S Smalley, P Loscocco, M Hibler, D Andersen and J Lepreau. The Flask Architecture: System Support for Diverse Security Policies. In *Proceedings of the Eighth USENIX Security Symposium*, pages 123-139, August 1999.

[2] Oliver Spatscheck and Larry Peterson. Escort: A Path-Based OS Security Architecture. Technical Report TR-97-17, Department of Computer Science, The University of Arizona, Tucson, AZ 85721, November 1997.
http://citeseer.nj.nec.com/spatscheck97escort.html
(Date accessed: Sept 23, 2002)

[3] O Spatscheck and L Petersen. Defending Against Denial of Service Attacks in Scout. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, pages 59-72, February 1999.
http://citeseer.nj.nec.com/article/spatscheck99defending.html
(Date accessed: Sept 23, 2002)

[4] P Loscocco and S Smalley. Meeting Critical Security Objectives with Security Enhanced Linux. In *Proceedings of the 2001 Ottawa Linux Symposium*, July 2001.

[5] P Loscocco, S Smalley, P Muckelbauer, R Taylor, J Turner and J Farrell. The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments. In *Proceedings of 21st National Information Systems Security Conference*, October 1998.

[6] E Felten, D Balfanz, D Dean and D Wallach. Web Spoofing: An Internet Con Game. In *20th National Information Systems Security Conference* (Baltimore, Maryland), October 1997.

[7] R Macgregor, A Aresi and A Siegert. How to Build a Secure World Wide Web Connection. Upper Saddle River, NJ. Prenticw Hall, c1996.

[8] Aleph One. Smashing the Stack for Fun and Profit.
http://destroy.net/machines/security/P49-14-Aleph-One
(Date accessed: Sept 23, 2002)

[9] Slagell, M. The Design and Implementation of MAIDS (Mobile Agents for Intrusion Detection System). Masters Creative Component paper, Mark Slagell, Iowa State University, May 2001.

[10] Apache Project. http://httpd.apache.org/
(Date accessed: Sept 23, 2002)

[11] SCO operating system vulnerability. http://www.ciac.org/ciac/bulletins/d-24.shtml

(Date accessed: Sept 23, 2002)

[12] Wu-ftpd vulnerability. http://ciac.llnl.gov/ciac/bulletins/j-065.shtml
(Date accessed: Sept 23, 2002)

[13] MURI Project. http://theory.stanford.edu/muri/
(Date accessed: Sept 23, 2002)

[14] WWW Security FAQ. http://www.w3.org/Security/Faq/
(Date accessed: Sept 23, 2002)

[15] http://www.rsasecurity.com/rsalabs/technotes/buffer/buffer_overflow.html
(Date accessed: Sept 23, 2002)

[16] AdCycle Vulnerability. http://qdefense.com/Advisories/QDAV-2001-7-2.html
(Date accessed: Sept 23, 2002)

[17] Microsoft IIS vulnerability. http://www.kb.cert.org/vuls/id/264272
(Date accessed: Sept 23, 2002)

[18] IMAP vulnerability. http://www.cert.org/advisories/CA-1998-09.html
(Date accessed: Sept 23, 2002)

[19] Compupter Emergency Response Team (CERT) Homepage http://www.cert.org
(Date accessed: Sept 23, 2002)

[20] Purify. http://www.rational.com/products/purify_unix/index.jsp
(Date accessed: Sept 23, 2002)

[21] StackGuard. http://www.cse.ogi.edu/DISC/projects/immunix/StackGuard/
(Date accessed: Sept 23, 2002)

[22] Linux Kernel Patch. http://www.openwall.com/linux/
(Date accessed: Sept 23, 2002)

[23] SYN Flood Patch ftp://ftp.dhp.com/pub/linux/security/linux.2.0.21.syn-flood.patch
(Date accessed: Sept 23, 2002)

[24] Ingress and Egress Filtering.
http://www.tru64unix.compaq.com/docs/iass/OSIS_53/admin/DNSTLSXX.HTM
(Date accessed: Sept 23, 2002)

[25] 2002 Computer Crime and Security Survey
http://www.gocsi.com/press/20020407.html (Date accessed: Sept 23, 2002)

[26] Morris Worm
http://www.software.com.pl/newarchive/misc/Worm/darbyt/pages/worm.html
(Date accessed: Sept 26, 2002)

[27] Intrusion Detection
http://www.sans.org/newlook/resources/IDFAQ/ID_FAQ.htm
(Date accessed: Sept 26, 2002)

[28] Stateless Protocols
http://www.tcs.hut.fi/old/pub/papers/aura/aura-nikander-icics97-abstract.html
(Date accessed: Sept 26, 2002)

[29] Syn Cookies
http://cr.yp.to/syncookies.html
(Date accessed: Sept 26, 2002)

[30] SELinux
http://www.nsa.gov/selinux/
(Date accessed: Sept 26, 2002)

9 APPENDIX

9.1 Loadable Kernel Module (LKM)

9.1.1 Introduction

If we want to add code to a Linux kernel, the most basic way to do that is to add some source files to the kernel source tree and recompile the kernel. In fact, the kernel configuration process consists mainly of choosing which files to include in the kernel to be compiled. These files may be patch files or other files that modifies the kernel.

An alternative is to add code to the Linux kernel while it is running. The code that we add in this way is called a Loadable Kernel Module (LKM), which need to be compiled first and then loaded. These modules can do lots of things, but they typically are one of three things: 1) device drivers; 2) filesystem drivers; 3) system calls. The kernel isolates certain functions, including these, especially well so they do not have to be intricately wired into the rest of the kernel. When LKMs are loaded they become a part of the kernel. The part of the kernel that is bound into the image that we boot, i.e. all of the kernel *except* the LKMs, is "base kernel." LKMs communicate with the base kernel. The advantages of using LKMs over binding into the kernel is as follows:

- The kernel need not be rebuilt so often. This saves time and reduces the possibility of introducing errors during rebuilding and reinstallation.
- LKMs can save memory, since the module is loaded only when it is used and unloaded when not used.
- LKMs help us to diagnose and isolate system problems and they are faster to debug and maintain.
- LKMs are not slower than base kernel modules. Calling either one is simply a branch to the memory location where it resides.

Sometimes we have to build something into the base kernel instead of making it an LKM. Anything that is necessary to get the system up far enough to load LKMs must obviously be

built into the base kernel. For example, the device driver for the disk drive that contains the root file system must be built into the base kernel.

9.1.2 Structure

A loadable kernel module has the following structure:

```
/* Standard headers for LKMs */
#include <linux/modversions.h>
#include <linux/module.h>

#include <linux/tty.h>      /* console_print() interface */

/* Initialize the LKM */
int init_module()
{
  :
  eg: // In the case of intercepting system calls, the original system call
in the system call table is replaced by the modified system call.
  :
  return 0;
}


/* Cleanup - undo whatever init_module did */
void cleanup_module()
{
  :
  // The modified system calls in the system call table are replaced by the
original system calls
  :
}
```

9.1.3 Insertion and removal of a loadable kernel module

To insert a module into a kernel, the command is

```
insmod module_name
```

To remove an LKM from the kernel, the command is

```
rmmod module_name
```