SensorWebIDS: A Web Mining Intrusion Detection System

C.I. Ezeife
School of Computer Science
University of Windsor
Windsor, Ontario N9B 3P4
cezeife@uwindsor.ca
and
Jingyu Dong
School of Computer Science
University of Windsor
Windsor, Ontario N9B 3P4
woddlab@uwindsor.ca
and
A.K. Aggarwal
School of Computer Science
University of Windsor
Windsor, Ontario N9B 3P4
akshaia@uwindsor.ca

November 4, 2007

Author Biography

Christie I. Ezeife received her M.Sc in Computer Science from Simon Fraser University, Canada in 1988 and a Ph.D in Computer Science from the University of Manitoba, Canada in 1995 following a First class B.Sc. Honors degree in Computer Science in 1982. She has held academic positions in a number of universities and has been an associate professor of Computer Science at the University of Windsor, Canada since 1999. Her research interests include distributed object-oriented database systems, data warehousing and mining. She has authored several technical publications including three comprehensive journal articles in the Kluwer's international journals of Distributed and Parallel Databases, Data Mining and Knowledge Discovery, two articles in Elsevier's journal of Data and Knowledge Engineering and IDEA group international journal of Data Warehousing and Mining, as well as two books on Problem Solving and Programs with C by Thomson Learning Publishers, which have been successfully used for teaching hundreds of first year Computer Science students for over seven years.

Jingyu Dong received his M.Sc. in Computer Science from the University of Windsor, in Fall 2006 in the area of data mining for web intrusion detection, under the supervision of Dr. Christie I. Ezeife. Jingyu previously graduated with a B.Sc. in Computer Science in 1999 from the School of Mathematics and Computer Science, Hubei, China. He has been employed in the industry working in network, database and web related embedded system programming for a number of years now and is currently with Amazon.com.

Akshai Aggarwal received the Ph.D. degree from M.S.University of Baroda in 1980. He also received the Master of Engineering degree in 1967. He was Professor and Head of the Department of EE and Computers at M.S. University of Baroda from 1984 to 1989. He also worked as Dean of Students for the Faculty from 1984 to 1987. In 1989 he joined Gujarat University as Professor and Head, Department of Computer Science. He is currently with the School of Computer Science, University of Windsor. His research interests include Security of Distributed Systems and resource allocation and scheduling in grids.

SensorWebIDS: A Web Mining Intrusion Detection System

Abstract

Purpose of this Paper: Paper proposes a web intrusion detection system, SensorWebIDS, which applies data mining, anomaly and misuse intrusion detection on web environment.

Design Approach: SensorWebIDS has three main components: the Network Sensor for extracting parameters from real-time network traffic, the Log Digger for extracting parameters from web log files and the Audit Engine for analyzing all web request parameters for intrusion detection. To combat web intrusions like buffer-over-flow attack, SensorWebIDS utilizes an algorithm based on standard deviation (σ) theory's empirical rule of 99.7% of data lying within 3σ of the mean, to calculate the possible maximum value length of input parameters. Association rule mining technique is employed for mining frequent parameter list and their sequential order to identify intrusions.

Findings: Experiments show that proposed system has higher detection rate for web intrusions than SNORT and mod_Security for such classes of web intrusions like cross site scripting, SQL-Injection, session hijacking, cookie poison, denial of service, buffer overflow, and probes attacks.

Research Limitations: Future work may extend the system to detect intrusions implanted with hacking tools and not through straight HTTP requests or intrusions embedded in non-basic resources like multimedia files and others, track illegal web users with their prior web access sequences, implement minimum and maximum values for integer data, and automate the process of pre-processing training data so that it is clean and free of intrusion for accurate detection results

Practical Implications: Web service security, as a branch of network security, is becoming more important as more business and social activities are moved on-line to the web.

Original Results in Paper: Existing network intrusion detection systems are not directly applicable to web intrusion detection, because these intrusion detection systems are mostly sitting on the lower (network/transport) level of network model while web services are running on the higher (application) level. Proposed SensorWebIDS detects XSS and SQL-Injection attacks through signatures, while other types of attacks are detected using association rule mining and statistics to compute frequent parameter list order and their maximum value lengths.

keywords: Web Intrusion Detection, Network Intrusion Detection Systems, Association Rules Mining, Sensor-based, Web Business, Crafted Web Intrusions

Paper Type: Research

1 Introduction

Security of network systems is becoming increasingly important as more sensitive information are being stored and manipulated online and more attacks are being launched every second. The security of a computer system is compromised when an intrusion occurs as it may cause data theft or hacker making the computer systems more vulnerable. Many of the worms have used such vulnerable networks of computers to spread fast through the internet. Consider the example of the sapphire worm [CAIDA, 2003], which was the fastest spreading computer worm in history. This worm virus spread through the Internet in 2003. The number of infected computer systems doubled in size every 8.5 seconds. In 10 minutes after release, it infected more than 90% of vulnerable hosts. An intrusion detection system (IDS) is an increasingly important part of the security system of a network. Two types of traditional IDSs [Bai & Kobayashi, 2003] exist: (1) misuse IDS, which uses patterns of well-known attacks to identify intrusions (e.g., famous Norton Antivirus software is a type of misuse IDS), and (2) anomaly IDS, which formulates the normal usage patterns and determines whether deviations from these normal patterns are intrusions, (e.g., common firewall products belong to this type of anomaly IDS). These two types of traditional IDSs require a lot of human involvement. Security experts have to analyze every known intrusion using their domain knowledge, and hand-code signatures (patterns) for those intrusions, and this process is very time consuming and expensive. In addition, these IDSs need frequent updates using new signatures when new intrusions emerge. The speed of these updates is much slower than the rate at which the new intrusions spread. Thus, many vulnerable computer systems with IDSs, which have not yet been updated, may be compromised.

A web application is a type of network service. From the network's perspective, a web service uses TCP communications just like any other network application. Apart from traditional network attacks like Denial of Service (DoS), Buffer Overflow attacks, which are also applicable to web applications, some new security problems are found that are specifically targetted at web applications, e.g., Cross Site Scripting attack (XSS) ([Cgisecurity, 2002]), SQL-Injection [Spett, 2005]), session hijacking ([Impera, 2006]), and cookie poison attacks ([Wikepdia, 2002]).

1.1 Understanding Network Intrusion Types

Basically, there are four classes of traditional network intrusions [Bai & Kobayashi, 2003], that may also affect web services, and four additional classes that are exclusively specific to web intrusion:
(1) User to Root (e.g., Buffer overflow) Attack (U2R).
(2) Remote to User Attack (R2U).
(3) Denial of Service Attack (DoS).
(4) Probes Attack (Probes).
The security risk degrees of these four kinds from highest to lowest are: U2R > R2U > DoS > Probes. That is, U2R attacks cause the most security concern. Unfortunately,

current IDSs using data mining technologies are useful for capturing DoS and Probes attacks, but perform very poorly on detecting U2R and R2U attacks.

Web-based intrusion is a branch of network intrusion, which has some new features that make it unique and are:

5. Cross Site Scripting attack ([Cgisecurity, 2002]),
6. SQL-Injection ([Spett, 2005]),
7. Session Hijacking ([Impera, 2006]),
8. Cookie Poison Attack ([Wikepdia, 2002]).

1. User to Root Attack: In this scenario, the attackers have normal user accounts on victim hosts. They exploit some software vulnerabilities to gain root accesses on the victim hosts, and do what they want after that, e.g., sending buffers that are longer than the maximum buffer length, and hiding some malicious executable code in the extra spaces. Examples of U2R attack are Eject and Fbconfig. These are two commands used in Sun Solaris and Unix system. Certain old versions of these two utilities contain vulnerabilities. The Eject attack exploits a buffer overflow in the eject utility. Fbconfig is the generic command line interface to query and configure frame buffer, which is vulnerable to buffer overflow attack.

2. Remote to User Attack: In this kind of attack, attackers usually do not have accounts on victim hosts. They send network packets to victim hosts and exploit some software vulnerabilities to gain normal user access. After that, they may use U2R attacks to gain root access and cause more severe damages. Examples are Dictionary and Ftp-write attacks. The Dictionary attack uses the fact that many users do not carefully choose a secure password. So, the attackers use some hacking tools to generate some common passwords and use them to guess the real password. Such an attack can be detected by monitoring the failed login attempts in a period of time.

3. Denial of Service Attack: DoS attacks can be performed on most modern software and operating systems. The attackers use tools to generate mass normal request for a service (e.g., HTTP), and overwhelm the capacity of the service providers, make them too busy or too full to handle legitimate requests. Examples are SYN flood and Ping of Death attacks. The Ping of Death attack can be discovered by examining the size of all ping packets and flagging those that are longer than 64000 bytes as anomalies.

4. Probes Attack: A hacker launches a probes attack by using hacking tools to scan all reachable network computers, to gather information, and find known vulnerabilities. Ipsweep and Mscan are examples of tools, which can be used to scan for vulnerabilities of another system.

HTTP Protocol and Web Logs

HTTP [Fielding et al., 1999] is a client-server based protocol used between web browser and web server. To get the resources from web server, the client sends HTTP request to server through GET and POST commands. Every resource in the web server has a unique resource identifier - URI. The resource could be a picture, e.g., “http://myweb/1.gif” or a file “http://myweb/f.html” or a path, which will be translated by the web server to a default resource, e.g., “http://myweb” will be translated to “http://myweb/default.html”. A web page can accept user supplied data as input, e.g., “http://myweb/login.asp” can

accept two parameters - "username" and "password". Once the user fills in these two fields in the web browser, and clicks "submit" button, the two parameters and their values are included in an HTTP packet and sent to the web server. Based on HTTP RFC2612 ([Fielding et al., 1999]), there are three places in an HTTP packet that can carry the parameter and value and these are URI, cookie and data section. Three example HTTP packets in three different places carrying the same parameters "username" and "password" for the URI "/login.asp" are given below:

```
1) Parameters kept in URI

   GET /login.asp?username=neo&password=111 HTTP/1.1\r\n

   Content-Length: 25\r\n

   User-Agent: Mozilla/4.0\r\r

   \r\n



2) Parameters kept in cookie

   GET login.asp HTTP/1.1\r\n

   Content-Length: 25\r\n

   User-Agent: Mozilla/4.0\r\r

   Cookie: username=neo&password=111\r\n

   \r\n



3) Parameters kept in user data section
```

```
POST login.asp HTTP/1.1\r\n

Content-Length: 25\r\n

User-Agent: Mozilla/4.0\r\r

username=neo&password=111

\r\n
```

Upon receiving an HTTP request, the web server processes it and responds with a return code (200 for normal, 400 series for error and for information) indicating the result. Both HTTP request and response are recorded into the web server's log file. A sample line in a typical log file looks like:
67.2.3.4 - - [10/Oct/2000:13:55:36 -0700] "GET
/exampleCGI.php?id=20 HTTP/1.0" 200 2326
This means "client 67.2.3.4, at time [10/Oct/2000:13:55:36 -0700], used GET method requesting the resource /exampleCGI.php with parameter id=20, web server returned code 200 (normal), the total request packet was 2326 bytes in length". It needs to be emphasized that in the log file, web server only writes down parameters carried in URI (unique resource identifier). The parameters in cookie and user data section (e.g., for data like user password) are totally ignored by the log file due to security and performance reasons. For example, the data in cookie and user data section could be too large to be written into the web log and some parameters like passwords cannot be written into log file to prevent their being leaked out to unauthorized persons. To see all the parameters, a program can use a network sensor to capture the real-time network packets and extract all the HTTP parameters from the HTTP request, as our SensorWebIDS does.

Web Service Attacks

Most web service attacks occur through the user input area. Dynamic web service technology has three categories. The first is using server-side HTML embedded scripting languages, like Active Server Pages (ASP), Java Server Pages (JSP), and PHP. The second is using server-side add-on component, e.g., Common Object Model (COM), and Java Servlet. The third is using server-side executable files or script, and re-directing user input to executable files and outputting of the files to the users. Description of web service attacks listed earlier on are given next.

5. Cross Site Scripting (XSS): Our inspection of over 100 web sites on the internet, shows at least 20% of them have this kind of vulnerability. Cross site scripting occurs when a web application gathers malicious data from a user through perhaps cookie theft. For example, if I was logged in as "john" and read a message by "joe" that contained malicious Java Script in it, then it may be possible for "joe" to hijack my session just by reading his bulletin board post. At the time of writing this document, these XSS vulnerabilities were found on the internet in strings of the form: $< script > alert(document.cookie) < /script >$.

Assume an example site with XSS vulnerability is:
$http://web4.university2.ca/units/cs/LNKnowledgeBase.nsf/$
$WHAll!SearchView\&Query = YourQuestion\&Seq = 1\&SearchFuzzy = TRUE$
To inject HTML code to exploit these vulnerabilities on this URL, the user:
i) supplies a value as the query string for parameter "Query". The query string will be at the place of "YourQuestion". A typical XSS attack could input a value " $< script > alert(\text{"}Hello, world\text{"}) < /script >$" for that parameter. Now, the full URL will look like:
$http://web4.university2.ca/units/cs/LNKnowledgeBase.nsf/$
$WH\_All!SearchView\&Query =< script > alert(\text{"}Hello, world\text{"})$
$< /script > \&Seq = 1\&SearchFuzzy = TRUE$
ii) This request will be transferred to the web server. At the server side, this query string looks like a normal string and is sent into database engine for querying.
iii) In most cases, this query string does not exist in the database and the web server will return a message to the client, telling that the query string cannot be found. The original query string will be copied into the returned HTML script. For example, a returned HTML script looks like:
$< html >$
$< head >$
$//someheadercodehere$
$< /head >$
$< body >$
...
$< DIVCLASS = \text{"}inbody\text{"}STYLE = \text{"}margin-left : 0.5cm; margin-top : 0.5cm\text{"} >< B > SearchResultsfor < script > alert(\text{"}Hello, world\text{"}) < /script >< /B >$
$< pCLASS = inbodyALIGN = LEFT >$
...
$< /body >$
$< /html >$
We can see the user provided query string is still there.
iv) When the explorer receives this response, it will treat the " $< script > \ldots < /script >$" as normal HTML script and execute it. Therefore, a message box will jump out in this example.
v) For a dangerous usage, an attack could post a malicious URL with embedded HTML code to some places (e.g., forum post), and allure a victim to click on this URL. After that, this piece of malicious HTML code will be executed on the victim's computer. The malicious HTML code could redirect a user to some unpleasant web site, stealing user cookie, and more.
vi) The solution for XSS attack is to filter user inputs, by prohibiting user from inputting dangerous HTML tags like $< script >$.
6. SQL-Injection: is a technique for exploiting web applications that accept user data in SQL queries without stripping potentially harmful characters first. For example, there is a login web page that accepts user name and password. Inside the program, the user name and password are sent to web server where these parameters are used in SQL queries like "select * from users where userid= " UserID" and passwd= "Password" ". In normal usage, the user name and password could be 'tom' and '111'. Then, the SQL

statement will look like: " select * from users where userid='tom' and passwd='111' ". In attack mode, the intruder will inject SQL keywords into the user name or password fields that makes the condition for the query always TRUE. Solution for SQL-Injection attack is also filtering user inputs, by prohibiting user from inputting dangerous characters like single-quotes or double-quotes. Another solution for Apache web server is to enable "magic quote" option, which will automatically pad single-quote or double-quote with a backslash. This will make these special characters lose their power in SQL queries.

7. Session Hijacking - File Disclosure & **File Access**

This kind of intrusion could make the attackers read confidential files (/etc/password), write any files to override important system files and crash a system. This kind of attack is caused by unfiltered user input. For example, if there is a CGI script that can view specified TXT files in user directory, the filename is obtained from the user. A valid user input should look like "my_file_txt". However, if this script does not validate user input, the attacker can use "../ ../ ../etc/password" to view the password files or other confidential files. Solution for this kind of attack is to filter user input and disallow characters like period and forward-slash.

8. Cookie Poison

Cookies are data key-pairs, which are stored in a file on the client's hard drive. Each pair consists of a name for the cookie, and then a value related to that name. For example, cookie_for_xyx.com = background_red. The major issue is that you can visit a web site, and they can store information about you on your computer. The cookie can store any information it wants, but can store only information about you that you give it, such as preferences, or last time you visited that page, or even your name. The information stored at the client side can only be retrieved by the server that placed the cookie. Cookies are helpful to improve your browsing, but are a common target for hackers, who can edit the cookie in a way that allows them to gain access to the personal information of other users on a web site. Using a crafted cookie to fool the server is called 'Cookie Poison'.

1.2 Motivations and Contributions

The paper proposes a web intrusion detection system, SensorWebIDS, which contributes the following features:

1. SensorWebIDS, collects data from both real-time network data (using proposed network sensor) and web log files to ensure all user data are being monitored.
2. To combat buffer-over-flow attack, SensorWebIDS defines a new algorithm using the standard deviation theory to calculate the maximum lengths of parameter values.
3. SensorWebIDS considers the relationships amongst HTTP access records in a given time window in order to capture some important information regarding frequency of specific attacks, hidden in the access sequence.
4. The system uses data mining technique to deduce frequent parameter order and presence for detecting new and old attacks.

5. Our system abstracts common signatures and sensitive keyword signatures, and error return codes to match known attacks.

1.3 Outline of the Paper

Section 2 examines existing network intrusion detection systems and web intrusion detection systems. Data mining concepts, association rules mining and common algorithms are also introduced. Section 3 presents detailed techniques of the proposed SensorWebIDS system. Section 4 presents the experimental results of our prototype system. Section 5 presents conclusions and future works.

2 Related Work on Network IDS

In the past 20 years, a lot of research (Swatch [Hansen & Atkins, 1993], STAT, WWWstat [Fielding et al., 1999], Autobuse [Taylor, 1998], Logscanner [Tuininga, 1998], MADAM ID [Lee, Stolfo & Mok, 2002], CyberCop[Network Associates, 1998], [Lee, Stolfo & Mok, 2002], ADAM [Barbara et al., 1999], [Han et al., 2002]) has been done in the intrusion detection area, and many IDSs (intrusion detection systems) for general network intrusion detection have been proposed. IDSs designed for general network intrusion detection are not efficient or even useful in web intrusion detection. Since many web attacks focus on applications that have no evidence on the underlying network or system activities, they are seen as normal traffic to the general network IDSs and pass through the IDS successfully.

Misuse Based Intrusion Detection Systems
Examples of these misuse systems are SNORT [Roesch, 1999], STAT, and most antivirus software like Norton antivirus software. SNORT [Roesch, 1999] is an open source IDS. There are many detection rules in SNORT. A sample rule to detect P2P software looks like: alert tcp $HOME\_NETany <>$
EXTERNAL_NET 8875 (msg:"P2P Napster Server Login"; flow:established; content: "anon@napster.com"; classtype:policy-violation; sid:565; rev:6;). Here, the signature in this rule is (content:anon@napster.com), which means any packet that contains this string could be a P2P traffic.
Anomaly Based Intrusion Detection Systems
The usage patterns are manually established based on domain knowledge. So, they are specific to such domains and cannot be applied to other domains directly. For example, the firewall system for a University is an anomaly detection system. It may only allow connections to well-known ports from outside, e.g., 80 (http), 22 (ssh), etc. Other kinds of connections are not trusted and blocked. This will limit many normal usages such as

ftp. And this also cannot prevent those kinds of attacks that embed themselves into http protocols. Recent research is aimed at eliminating as much of manual processing while building an intrusion detection system by applying data mining techniques.

Applying Data Mining to IDS

The goal of techniques applying data mining is to eliminate the need for manually analyzing and encoding intrusion patterns, as well as the guess work in selecting statistical measures for normal usage profiles. The idea is to use association rules to guide audit data gathering during data pre-processing stage, using frequent episodes to guide feature selection, and building up a classification model based on training audit data to classify future audit data into classes of normal usages or intrusions. The first job for data pre-processing is to locate packets in the network traffic, each of which is a bit stream like "010001111", recognize the protocol and store them as structured database table records using network sniff tools like tcpdump. The typical output data of tcpdump is a table with the schema (timestamp, src_ip:src_port, dest_ip:dest_port, flag), which gives the timestamp, source IP address, source port, destination IP address, destination IP port, and TCP flag. Association rules mining [Agrawal & Srikant, 1994] is used to guide audit data gathering during data pre-processing stage, frequent episodes mining is used to guide feature selection [Lee, Stolfo & Mok, 2002], and a classification model based on training audit data is built, to classify future audit data into different classes (i.e. normal usages or intrusions). To facilitate data mining process, packet records need to be summarized into connection level, i.e., collapse the packets belonging to the same connection. In [Paxson, 1998], Paxson provided a useful tool - Bro, which can reassemble the network traffic into connection level, filter the traffic stream into a series of events (e.g., connection being closed normally is an event), and execute scripts that contain site-specific event handlers. This is the second pre-processing for the audit data.

Applying Association Rule Mining

Association rules are used to determine correlations in the connection records obtained after pre-processing. Lee [Lee, Stolfo & Mok, 2002] proposed a novel method to determine when the training audit data is enough. The idea is to use the number of frequent association rules as an indicator on whether the audit data is sufficient. This means that when the audit data is gathered, they compute the association rules from each new audit data set, and merge the new rules into the existing aggregate rule set. The added new rules represent new variations of the normal behavior. When the frequent aggregate rule set stabilizes because no new rules are discovered and added from new arriving audit data, they stop the data gathering since they believe collected aggregate audit data set has covered sufficient variations of normal behavior. This approach of merging rules is based on the fact that the frequent behavior types are limited and even the same type of behavior type will have slight differences across audit data sets. One approach for combining similar patterns into more generalized one is: Two rules r_1 and r_2 are merged into one more general rule r if their right and left sides are exactly the same or their right hand side and left hand side can both be combined, and the support and confidence values are close. Lee [Lee, Stolfo & Mok, 2002] uses axis attributes (the essential attributes in the record) to extend the Apriori algorithm [Agrawal & Srikant, 1994] to consider domain knowledge when mining rules in order to generate the most relevant rules. For

example, consider the Connection table with schema (Time, Duration, Service, Src_bytes, Dst_bytes, Flag). The Apriori algorithm may generate a frequent rule like (src_bytes = 200 → flag = SF), which is useless because there is no intuition for the association between the number of bytes from source (src_bytes) and the normal status (flag = SF). This problem of mining rules with non-relevant attribute associations, prevents frequent episode mining from being effective too. Thus, Lee's solution [Lee, Stolfo & Mok, 2002] is to first define axis attributes (e.g., Service), then, mine the Connection table using Apriori algorithm to find frequent item sets, e.g., (src_bytes = 200, flag = SF). Then, it removes all such frequent itemsets with no axis attribute in its set, like the one above (src_bytes = 200, flag = SF).

Level-wise Approximate Mining Algorithm

Lee [Lee, Stolfo & Mok, 2002] still used the level-wise approximate algorithm to find the low frequent but important patterns. In daily network traffic, some services (connections) like gopher, happen for low occurrences but their patterns are still needed to be merged into the network traffic profile or these services will be misclassified as intrusions or anomaly. Thus, the level-wise mining algorithm is used for low frequent but important frequent episodes. The idea of the algorithm is to first find all the episodes with high frequent axis attribute values, and then, iteratively, lower the support threshold to find the low frequent episodes with axis values, but these axis values must be new axis value which have not appeared before.

Applying Frequent Episode Mining Algorithm

Lee [Lee, Stolfo & Mok, 1998, Lee, Stolfo & Mok, 1999, Lee, Stolfo & Mok, 2002] proposed using mined frequent episodes from network connection records as guidelines for constructing temporal statistical features for building classification models. For example, suppose they obtain a set of frequent episodes by mining the connection records and 1) for a frequent episode, if the same value of an attribute is repeated several times, they should include a corresponding count feature. Thus, given a frequent pattern like (service = smtp, scr_bytes = 200), (service = smtp, scr_bytes = 200) → (service = smtp, scr_bytes = 200), [0.81, 0.42, 140], which implies that in the past 140 seconds, three smtp connections appear one after the other. This will lead to a feature that counts the number of connections having the same service and src_bytes as the current connection record in the past 140 seconds.

2) for a frequent episode, if an attribute with different values is repeated several times, they add an average feature corresponding to the different values. For example, given frequent pattern (service = smtp, duration = 2), (service = telnet, duration = 10) → (service = http, duration = 1), [0.81, 0.42, 140]. This will lead to adding a feature that represents average duration of all connections in the past 140 seconds. The optimal window size to use for frequent episode mining is discovered from experimentally mining frequent episodes using different window sizes until the number of frequent episodes becomes stable.

Building Classification Models

After raw audit data pre-processing and feature selection, IDS applying data mining [Lee, Stolfo & Mok, 1998, Lee, Stolfo & Mok, 2002] would use classification techniques on these audit data in order to build classification models. They use RIPPER, a rule learning program developed by Cohen [Cohen, 1995]. Other decision induction algo-

rithms that could be used are C4.5 and ID3 although RIPPER is argued to be more efficient than both. To build misuse and anomaly detection models, they start with a set of connection records containing diverse attacks as training data to RIPPER program. The input records have the class label of attack type and the schema of this input looks like: Connection (label, service, flag, hot, failed-logins, compromised, root-shell, su, duration). The output of the RIPPER program is the set of rules for building misuse or anomaly (deviating from most normal connection rules) detection model. Example output rules are: If number of failed logins is greater than 5, then this telnet connection is "guess", meaning a guessing password attack. During detection, the algorithm can now use this set of rules on new records to classify the record as a kind of known attack or normal traffic. When building misuse models, it starts with training data that capture a wide variety of attacks so that it can match known attacks. When building anomaly models, it starts with training data that capture a wide variety of normal connection behavior so that it can identify deviations from these.

The most famous Network IDS projects using data mining techniques are MADAMID ([Lee, Stolfo & Mok, 2002]), ADAM ([Barbara et al., 1999]), and MINDS ([Ertoz et al., 2004], [Chandola et al., 2006]).

Other Web Intrusion Detection Systems

There are a few papers and systems dedicated to web intrusion detection: Swatch [Hansen & Atkins, 1993], WWWstat [Fielding et al., 1999], Autobuse [Taylor, 1998], Logscanner [Tuininga, 1998], CyberCop [Network Associates, 1998], SNORT [Roesch, 1999], and mod_Security [Ristic, 2003]. These systems try to find problems in the log files of the web server. Limitations in some of these tools include their lack of support for the encoding scheme of hexadecimal characters defined in HTTP. This permits an attacker to easily avoid detection. The script languages used by some tools are not flexible enough as they have their own syntax, e.g., SNORT has its own expression for representing signatures of attacks. There are methods to filter out false alarms, such as canceling all events from certain domains, but it would be useful to define filters based on other properties. Autobuse [Taylor, 1998] allows the specification of a defined threshold per host, which allows reports to be suppressed until a given host has performed several attacks, but there is no distinction of the severity of received events. In [Ristic, 2003], Ivan developed an open source web application firewall named modsecurity. It operates as an Apache web server module or standalone. The purpose of modsecurity is to increase web application security, protecting web applications from known and unknown attacks. The highlights of this system is that it cannot only discover intrusions, but can prevent some basic web intrusions like SQL-Injection and XSS attacks, by using filter to remove unpleasant characters in user supplied data. If installed as an Apache module, it can monitor user-supplied data from all the sources (in URI, cookie, user data section). The shortcomings of this system includes:

1. it cannot detect sophisticated intrusions like buffer overflow because it does not know how long the maximum value length should be.
2. It has difficulty detecting DoS attack because it does not consider the relationship between subsequent events but evaluates intrusion possibility on each HTTP request individually.
3. In order to monitor user supplied data from all sources, it has to be installed as an

Apache module on the same machine as the Apache and this will influence the efficiency of the web server. Also, a hacker could find a bug in this IDS system itself and attack it to shutdown the Apache server so as to lunch a DoS attack.

3 The Proposed SensorWebIDS

The architecture of the proposed SensorWebIDS is shown as Figure 1, while the audit engine detection algorithm, that drives the system is shown as Figure 2. The web intrusion detection system (SensorWebIDS) consists of three main modules: network sensor (section 3.3), log digger (section 3.2), and audit engine (section 3.1). The network sensor is used to capture all HTTP requests in real-time from network traffic and to send them to the audit engine for intrusion analysis. The log digger is used to extract HTTP requests from web server's log files and send them to audit engine for intrusion analysis. The audit engine is the brain and the decision-maker of SensorWebIDS. The SensorWebIDS runs in two modes: learning and detecting modes. In the learning mode, the network sensor or log digger sends extracted data to the audit engine as training data. When in detecting mode, HTTP requests, captured by the network sensors, or log files are transported into the audit engine for anomaly analysis. The terminal could be a screen device or a text file. The SensorWebIDS uses a combination of anomaly based detection technique (with network sensor and frequent parameter presence computations using association rule mining) and misuse based detection technique features based on well defined signatures for web intrusions.

Signatures in SensorWebIDS
SensorWebIDS abstracts the common nature of existing web attacks into several high level signatures instead of finding pattern for each known web attack. These common signatures can be categorized into two classes (a) those based on web server return codes and (b) those deduced from forbidden keywords used in web intrusions. By tracking web server return codes generated by the same users within a short period of time, we can identify hacking activities if web server return error codes pass acceptable thresholds. Four most important rules used for signature class return codes are: 1) a client cannot access too many non-existing URIs in a short time, 2) a client cannot produce too many unauthorized error in a short time, 3) a client cannot produce too many forbidden errors in a short time, and 4) requests that cause internal errors are suspicious. For the second class of signatures (those for forbidden keywords), SensorWebIDS monitors parameters in user data to identify script attacks. For the cross site reference (XSS) attacks, the most dangerous keywords are $< /script >$, $< script >$, $< img >$, HTML tags in the format $< Tag >$. The forbidden keywords for SQL-Injection attack are single and double quotes in SQL-Injection and $< \%\% >$, $< /script >$, $< script >$, $< img >$ in XSS attack. These are obtained from monitoring web attacks and are updated manually when new attacks appear. A future work may also include automating the process of updating the signatures. The audit engine module can easily be connected to a database to store each

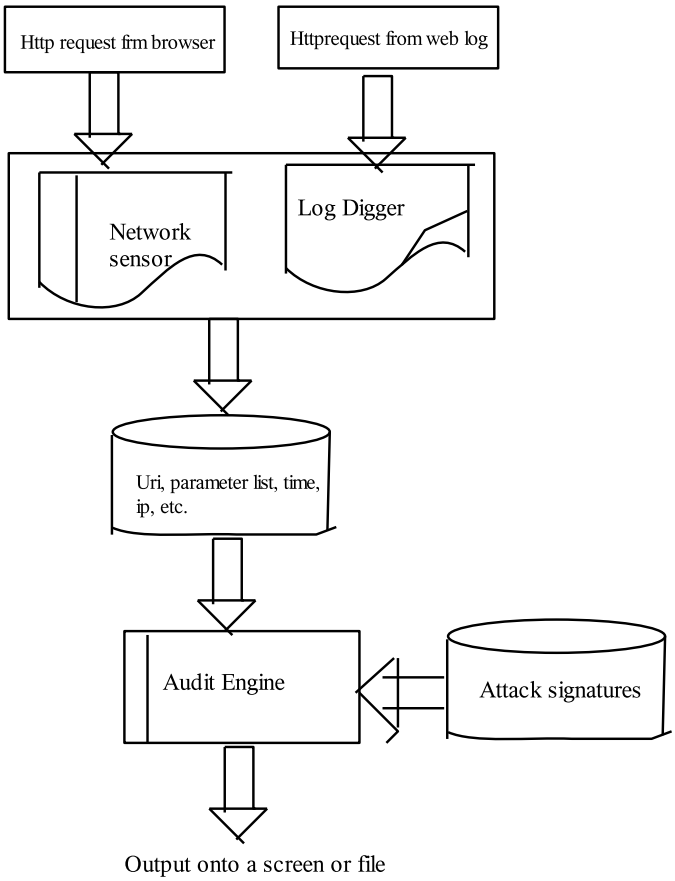

Figure 1: SensorWebIDS System Architecture

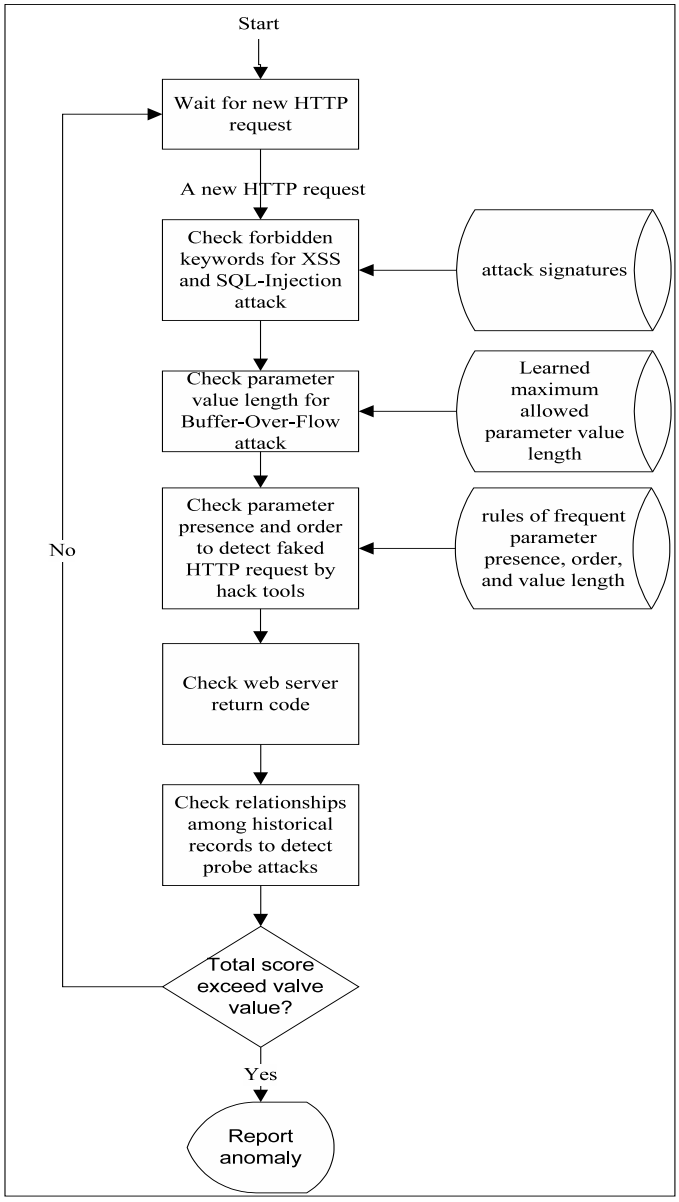

Figure 2: Audit Engine Detection Algorithm Flow

received record so that other data mining tools can be used to analyze the database.

Signatures Based on Web Server Return Code

The web server always makes a response to an HTTP request, which is recorded in the server web log file. The HTTP specification defines a set of HTTP return codes for web server. The web server returns an error code because a malicious request is received or a defective web page is accessed. Receiving continuous bad requests from the same user is especially suspicious as a hack activity. Thus, by tracking web server return codes, we can find hack activities. The four common rules useful for detecting some web attacks are:

1. A client cannot access too many non-existing URIs in a short time. This type of error is 404 'URI does not exist error'. The anomaly score we assign for this type of error is 1.
2. A client cannot produce too many unauthorized error in a short time. This is error return code 401 for authentication failure and we assign an anomaly score of 2 for this error.
3. A client cannot produce too many forbidden errors in a short time by trying to access such things as confidential files. The web server return error code is 403 for forbidden message and our anomaly score assignment is 2.
4. Requests that cause internal errors are suspicious and if at most times, an HTTP request causes the web server return code of 500 (for unexpected condition), we assign an anomaly score of 5.

Signatures Based on Forbidden Keywords

5. For XSS attack, the most dangerous keywords are </scrip>, <scrip>, and <img>. Other general HTML tags are less dangerous but should also be paid attention to. HTML tags are in the format of <Tag>. The audit engine detects HTML tags by searching for < and > in the parameter values of an HTTP record and for each pair of < and > found, an anomaly score of 10 is added to the anomaly score of that record.
6. For SQL-Injection attack, the dangerous character are single quote (' ') and double quotes (" "). The audit engine adds 10 to the anomlay score of the record for each of this these two keywords found. All these forbidden keywords are stored as array forbidden_keyword_list in the Algorithm 2.

3.1 Audit Engine

When the audit engine is launched, it first runs in its training mode using training HTTP parameters from sensor and log digger, to calculate parameter presence and order as well as the maximum value lengths of the parameters. Later, it continuously runs in detection mode using the algorithm shown in Figure 2 and Algorithm 2. The algorithm for the audit engine training mode is presented as Algorithm 1. In training or learning mode, it calls Algorithm 3 to calculate frequent parameter presence and order, and an algorithm (discussed below) to calculate maximum value length.

Algorithm 1 *(AuditEngine-Learn)*

Algorithm AuditEngine-Learn()
Input: record_set - containing URIs with parameters
Output: 1)URIs_FP - an array storing URIs with frequent
parameter list, URIs_MAX_LEN -array storing
maximum allowed value length for URI,
num-records (total number of database records);
Begin
/* find frequent parameter presence and orders (Algorithm 3).*/
1. Call FindFrequentParameterAndOrder(record_set, confidence, axis_item);
/* find maximum value length (algorithm described below) */
2. Call CalcMaxValueLen(record_set);
End

Algorithm 2 *(AuditEngine-Detect)*

Algorithm AuditEngine_Detect()
Input: record_set - containing URIs in HTTP,
parameter_list in the form [parametername, parametervalue],
receiving_time, client_ip_address, server_returned_code,
maximum_allowed_anomaly_score
Output: TRUE if this HTTP request is an intrusion,
FALSE otherwise
Other Variables:
forbidden_keyword_list (an array), /* for XSS and SQL_Injection attacks*/
ANOMALY_SCOREs (an array), /* for each check point */
URIs_FP (an array) /* for frequent parameter lists */
ERR_CODE (an array) /* for storing HTTP error codes */
history_queue /* for HTTP requests */
accumulated_anomaly_score /* for repeated requests */
Begin
/* Check for XSS and SQL_Injection attacks */
1. For each keyword k in forbidden_keyword_list
if (k is found in parameter_list)
score = score + ANOMALY_SCORE1;
/* Check for Buffer overflow attacks */
2. For each parameter_name p in the parameter_list
if (the current parameter_value length > maximum_length)
score = score + ANOMALY_SCORE2;
/* Check for parameter presence and order to discover faked
HTTP requests generated with hack tools */
3. For each parameter_name p in the parameter_list
if (parameter_name is not in the URIs_FP[uri].frequent_parameter_list)
score = score + ANOMALY_SCORE3;
/* Check server return code to count in server reaction to request*/
4. if (return_code is in the ERR_CODE_list)
score = score + ANOMALY_SCORE4;
/* Check relationship among multiple records for detecting probe attacks*/

5. For each record r in history_queue
   if (receive_time of new record minus receive_time of r) < N)
   Begin
   if (return_code in r is an error code)
   accumulated_score = accumulated_score + anomaly for error code
   End
   else break;
   push this record into history_queue
   if (receive_time in new request minus receive_time of tail
   record of the queue > N)
   delete tail record from the queue
   score = score + accumulated_anomaly_score
6. if (score > Max_Allowed_Score)
   return TRUE /* means report as an anomaly */
   else
   return FALSE /* means report as normal request */

End

Finding Frequent Parameter List and Order

The algorithm for finding frequent parameter list and their order, takes a set of URIs with parameter lists, and finds the frequent URI's parameter list. It does this by applying the Apriori algorithm on the parameter list to find the frequent rules having confidence greater than cfd (the given value or the desired confidence for frequent rules), and then storing these rules in array x. Next, each rule in array x is post-processed to delete rules not having the axis items (given parameters) as their antecedents or left hand attribute. The formal summary of the algorithm FindFrequentParameterAndOrder is given in Algorithm 3.

Algorithm 3 *(FindFrequentParameterAndOrder)*

Algorithm FindFrequentParameterAndOrder()
Input: param_list - URIs and their parameter lists,
cfd - desired confidence for the frequent rules,
axis_item - all frequent rules start from this
item(the URI name).
Output: frequent URI's parameter_list.
Begin
1. Apply Apriori algorithm on the param_list to
find frequent rules having confidence greater than
cfd, and store as array x;
2. For each rule r in array x do
   Begin
   if (r is not started by r[axis_item_idx])
   delete r from x;
   End
3. return x;

Table 1: Example Parameter List Computation

Record	Parameter List
1	/login.asp, username, password
2	/login.asp, username, password
3	/login.asp, username, password
4	/login.asp, username

End

For example, assume the system receives input record set shown in Table 1 with minimum confidence of 75% and axis item "/login.asp" as index. The algorithm for finding Frequent Parameter List would proceed to generate the following rules after applying the Apriori algorithm:
"/login.asp" → "username", "password", confidence=75%
"/login.asp" → "username", confidence=100%
"/login.asp" → "password", confidence=75%
"username" → "password", confidence=75%
"/login.asp", confidence=100%
"username", confidence=100%
"password", confidence=100%
Next, it eliminates those rules that are not started with "login.asp". So finally, we get frequent parameter lists as the rules on lines 1, 2, 3 and 5 of the list above. During detection stage, requests with parameter "username" only or "password" only or "password, username" will be assigned anomaly scores.

Calculating Maximum Allowed Parameter Value Length

Since all user input values are of string type, there should be a maximum parameter input length at the server side. We can use this maximum allowed value length to detect buffer overflow attacks. However, this maximum length exists in the source code on the server side and is unknown to the IDS. There is no way for the IDS to learn this length automatically. A simple implementation may use the maximum length observed in the history data in the web logs as the maximum length allowed in the source code. But this information is obviously not accurate and will inevitably lead to false alarms. For example, if the maximum allowed 'username' is 64 in the source code, and the observed maximum length in the web log is only 32, if a new request comes with length 40, the IDS will take this as an attack. To solve this problem, we compute the maximum length of each parameter using the statistics and probability theory's "empirical rule" of "68-95-99.7" for summarizing the relationship between the standard deviation, the mean and the data values in a data set [Wikepdia, 2002]. This means that, in practice, for an approximately normally distributed population, about 68.27% of the values lie within 1 times the standard deviation of the mean of the population, about 95.46% of the values lie within 2 times the standard deviation of the mean and about 99.73% lie within 3 times

the standard deviations (or 3σ) of the mean. Thus, to dynamically predict the maximum length of a parameter, our algorithm determines the number of elements (users) for each parameter (like username and password and location). Then, the algorithm computes the maximum length for each parameter X which has values x_1, x_2, ..., x_n and a mean $\overline{x}$ as their standard deviation $3 * \sigma + \overline{x}$, where:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X_i - \overline{x})^2}$$

This method accommodates each URI having more than one parameter, e.g., URI "/login.asp" has two parameters - "username", "password". Also, each parameter has a series of values from different HTTP requests. For example, for 'username' parameter, user A inputs "mary" and user B inputs "Joe", and if these are the only values for this parameter, 4 and 3 will be used to determine the maximum length for parameter "username". It also accommodates web servers having several URIs. The system should repeat the same computations for each URI.

Updating Learned Frequent Parameter Presence and Maximum Value Lengths
Currently, all frequent parameter presence and order and maximum value lengths are learnt during training period. In the detection period, these rules and values are used to match normal usage pattern. They are not updated after training period. New parameter rules and maximum value lengths may need to be updated after the training period under the following scenarios:

1. The administrator may install some new web applications on the web server. So there may be new URIs not present in our learned rules. For example, at the time when the system collected the training data, there were 2 web pages on the web server: "/login.asp" and "/search.asp". Then, the system learned all rules related to these two web pages. Later after that, the server administrator installed a new web page "/add.asp". Now, this new web page does not appear in the learned knowledge base, and intrusions related to this web page cannot be found by the old rules.
2. The administrator may update some old web applications in the web servers by adding, deleting or changing some parameters. For example, the web page "login.asp" used "username" and "password" in the first version in the training period. After that, the administrator updated this web page and added a new parameter "location". Then, this parameter is not included in the rules. New HTTP request including this new parameter will be marked as anomaly incorrectly. The technique for updating the rules is left for future work. For taking into account new data, updating process may require periodic re-training to replace old rules, and developing some incremental methods for updating learned rules, which may be running in another background process.

Audit Engine Detection Process
When the audit engine is running in detection mode, for each new incoming HTTP record, the system calculates its anomaly score by going through five checkpoints, and adding anomaly score for all checkpoints together as the total anomaly score for the record as shown in Algorithm 2. If the total anomaly score is greater than the maximum allowed value, this HTTP record is marked as anomaly. This maximum allowed value is an adjustable value based on experimental results and can be changed to balance between high

detection rate (with lower threshold value) and low false alarm (with higher threshold value to reduce false alarms). The five checkpoints are for detecting various classes of intrusions introduced earlier. Thus, in the extracted parameter values, it checks for:
1. forbidden keywords to find XSS and SQL-Injection attacks using known attack signatures.
2. parameter value lengths to see if they are longer than the maximum allowed value lengths for that same URI and parameter name in order to detect buffer-over-flow attacks.
3. parameter list, comparing its parameter presence and order with frequent parameter lists for the same URI. If that was not found, an anomaly score is added to the total score. This is useful for discovering faked HTTP requests that are generated with hacking tools. The frequent parameter lists are learned in the training period.
4. if web server returns an error code for a request, an anomaly value is added to total anomaly score.
5. relationship among multiple records to take into consideration historical damage caused by the same user. This is useful for detecting probes attacks.
The assigned anomaly score (of between 10 and 1) is based on the severity of the intrusions captured by the rules. For example, the maximum length rule has higher score than the forbidden keywords rule, etc.

3.2 Network Sensor

One contribution of this system is online web intrusion detection with a newly developed software network sensor. Sensors used by existing systems like snort get information from IP/TCP header, that are not specific to web intrusions. Our own network sensor is totally independent of the audit engine and can be put any where in the network to get the best picture of web intrusions. To capture intrusions embedded in HTTP parameters not recorded in web server's log files, we use a network sensor to capture raw network packets from real-time network traffic.

This network sensor program captures all the HTTP packets to/from the web server enabling monitoring and extraction of all user data in HTTP packets, which it sends to audit engine. Monitored data are in URI, cookie and user data sections and extracted data are URI, user IP, time of receiving, web server return code, parameter list. The sensor extracts:
1. parameters embedded in GET URI or HTTP GET request like
"$/login.asp?username = abc\&password = 123HTTP/1.1/r/n$", where the parameters are: "username" and "password", values are "abc" and "123".
2. parameters in content fields through POST request. POST method does not show parameters in URI as GET method does, but it puts the parameters in HTTP content instead.
3. cookie data stored at client side and from web server.
WinPcap [Risso & Degioanni, 1995] and LIBPcap [McCanne & Jacobson] are two free programming libraries that can be used by programmers to develop network sensor ap-

plication. While LIBPcap is the library for UNIX platform, WinPcap is the library for win32 platform, which has the same interface as LIBPcap. The libraries allow network card to be set into promiscuous mode so that all network packets can be received. To capture all packets on the network, the SensorWebIDS uses WinPcap - a free link-layer network access library ([Risso & Degioanni, 1995]). With this library, the network sensor sets the network card into promiscuous mode, which allows the network card to receive all network packets regardless of the destination of the packets. The algorithm for the Network sensor is given as Algorithm 4.

Algorithm 4 *(Networksensor)*

Algorithm Networksensor()
Input: Network packet
Output: 1) formatted HTTP request including URI,
user IP, time of receiving, web server
return code, parameter list
Begin
While(TRUE)
Begin
(1) Using WinPCap, Receive a network packet p
from network card;
(2) If (p is NOT HTTP packet)
(2.1) Continue loop;
(3) Extract URI, user IP, time of receiving,
web server return code, parameter list out of
HTTP packet, and format them in a desired style;
(4)Send the formatted data to audit engine;
End
End

3.3 Log Digger

Log digger is the part of the SensorWebIDS system that extracts parameters for HTTP requests in web server's log file. An example record in the log file looks like: 127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /exampleCGI.php?id=20 HTTP/1.0" 200 2326. Thus, the log digger accepts the web log as its input. The log digger then, parses each record in the log file and extracts useful information that it sends to audit engine for training. Output data returned by the log digger as its result are extracted formatted HTTP request including: (1) URI (e.g., /exampleCGI.php?id=20 HTTP/1.0), (2) user IP address or remote host that made the request to the server (e.g., - or hyphen in the above example indicating that his information is not available), (3) time of receiving the request (e.g., 10/Oct/2000:13:55:36 -0700), (4) web server return code (e.g., 200), (5) parameter list (e.g., size of file 2326 bytes). Note that the server return code beginning

with 2 indicates a successful response from server, a code beginning with 3 indicates a re-direction, a code beginning with 4 indicates an error caused by the client and a code beginning with 5 indicates an error caused in the server. The algorithm for the log digger is given as Algorithm 5.

Algorithm 5 *(LogDigger)*

Algorithm LogDigger()
Input: web log file name
Output: 1) formatted HTTP request including URI,
user IP, time of receiving, web server
return code, parameter list
Begin
(1) Open web log file
(2) While (not end of web log file)
Begin
(2.1) Extract URI, user IP, time of receiving,
web server return code, parameter list out of HTTP
packet, and format them in a desired style
(3) Send the formatted information to the audit engine
End
End

4 Experimental Evaluation

We developed a prototype system, SensorWebIDS, that implements the algorithms described in Section 3. The system consists of three main programs: audit engine (section 3.1), network sensor (section 3.2), and log digger (section 3.3). The goal of this experiment is to prove that our SensorWebIDS system can detect more types of web intrusions than other systems at the same or lower cost. We compared the SensorWebIDS with two other systems - SNORT [Roesch, 1999] and mod_Security [Ristic, 2003], both of which detect some web intrusions.

Implementation and Testing Environment

Our prototype system has been developed in C language under windows platform. However, with minor changes, it can be migrated to UNIX platform. These changes include replacing system dependent APIs (e.g., process manipulation APIs), creating make files for the system, and compiling the whole system under UNIX with C compiler. The experiment is divided into three parts. In the first part, we test our SensorWebIDS along with two other systems (snort and mod-security) using crafted web intrusions, to see how many intrusions each system can find. In the second part, the efficiency of the three systems are tested using continuous crafted web intrusions and comparing the CPU usage on the computer, that is hosting the IDS system. In the third part, the SensorWebIDS is tested using log files from real web servers to see how it performs in the real world.

Although all components of SensorWebIDS can be placed on the same machine, however, to simulate more complex setting demonstrating independence of the components, our testing environment consists of 4 computers (S1, S2, S3 and C1): S1 is running the web server (Apache 2.2 + Tomcat 5.5) and mod_Security [Ristic, 2003] (this system must be installed as a component to the web server). S2 is running SensorWebIDS (audit engine only) and SNORT [Roesch, 1999]. S3 is running network sensor and log digger parts for SensorWebIDS. C1 is the terminal, which is running windows XP SP2 with Internet Explorer 6.0 (both are trademarks of Microsoft Corporation), to simulate user input. They are all connected to a shared HUB device communicating through TCP connections, which ensures the traffic to the web server can be fully seen by our system. The sensor and log digger need to know the IP address for the audit engine at startup, to set up connection with it. The audit engine listens on a particular TCP port that sensor or log digger will send requests to. The hardware configurations for all the machines are the same - CPU AMD 2400+, 512M RAM. On the web server S1, we installed a simple web application named "login.jsp". To train the SensorWebIDS, we used the web browser in C1 to access the "login.jsp" page on S1, and repeatedly inputted legal username/password on that page for 9 times (each username/password for 3 times). Then, we took out the web server's log file from S1, and sent it to the log digger that is running on S3. The log digger formatted the HTTP requests and sent them to audit engine running on S2 for training. After that, the audit engine learned and stored the frequent parameter presence and order as ["login.jsp" → "username", "password"], and the possible maximum lengths for "username" as 9.83 and "password" as 8.89. We used a total of 60 test intrusions from six types of web intrusions, 10 intrusions from each class for test cases. The six types of web intrusions tested are: (1) Cross Site Scripting, (2) SQL Injection, (3) Denial of Service, (4) Buffer Overflow, (5) Cookie Poison, (6) Simulation of using hacking tool. The descriptions of the crafted intrusions from the 6 classes of web intrusions are provided in 2.

The result of detection is reported in Table 3. The few type 4 undetected are when user value length falls between our predicted and actual maximum lengths. Our system detects XSS and SQL-injection attacks through signatures, but other types of attacks including type 4, are detected through data mining based anomaly detection techniques using association rule mining and statistics to compute frequent parameter list order and their maximum value lengths.

On detection speed, the three algorithms have about the same detection speed although the SensorWebIDS detects more intrusions than the other two. The SensorWebIDS was also used to analyze real web logs of a small radio station web site (log1) (with 25184 records), a small-size company web site (log2) with 49755 records) and a University Computer Science web site (log3) (with 878521 records). The purpose of this experiment is to demonstrate the effectiveness of the system in real application domains. We used 20% of the log data for training, inspecting manually to ensure that the training data is free of intrusion. Then, we used 80% of the log data to test for intrusion detection. The result of the three logs analyzed with SensorWebIDS and the intrusions detected are shown in Table 4.

Table 2: Crafted Intrusions for Experiment

Group#	Attack	Example Input	Description
1	Cross Site Scripting	Use URL "http://S1/ login.jsp?username = <script>alert(1)</script> &password-123" in the C1's web browser	XSS embedded HTML tags in user's input. Our system captured the keyword <script> as the proof of the attack.
2	SQL Injection	In the 'login.jsp' web page, type username = Joe, and password = " or "1"=1	SQL-Injection attack uses " or ' to fool SQL server. Our system captured the keyword " as the proof of the attack.
3	Denial of Service	Use packet generating tool to send request with wrong username&password	The tool caused the web server return error code 400 at a speed of 200 time/sec.
4	Buffer over flow	In the 'login.jsp' page, use 'username' from length 1 to 10	The maximum allowed value for 'username' is 8
5	Cookie poison	In C1, change the stored cookie 'SESSIONID' to the value of 32 'b's	The maximum allowed value for cookie 'SESSIONID' is 16
6	Simulation of using hack tool	Use 'http://S1/login.jsp? password=111&username=Mary" in the C1's web browser	Hack tools do not use the same parameter order and presence as that of official web page. The correct parameter order and presence [username, password]

Table 3: Detection Rates with Well-known Attacks

IDS	Attacks	Detect -ed	Not Detected	Detect Rate
SWebIDS	60	59	one type 4 attack when the 'username' length of 9, (i.e., > maximum)	98.3%
SNORT	60	20	all attacks in type 3,4,5,6	33.3%
Mod_Sec	60	20	all attacks in type 3,4,5,6	33.3%

Table 4: SensorWebIDS on Real Web Logs

Log File	Records	Det_ ected	Not Det_ ected	Attack Type
Log1	25174	2	0	2 probes
Log2	49755	16	0	5 DoS, 4 probe, 3 buffer, 4 false alarms
Log3	878521	5	0	5 false alarms

5 Conclusions

This paper proposes a novel web intrusion detection system, SensorWebIDS, which combines the power of anomaly and misuse detection. It applies association rules mining to find frequent parameter presences and orders for anomaly detection. Based on our research on well-known web intrusions, we abstract some common and sensitive keyword signatures to match known attacks for misuse detection. Our system develops a software sensor to monitor real-time traffic. Being different from existing web intrusion systems, which get HTTP requests only from web log files that contain only part of user inputs, our system collects data from both real-time network data and web log files to ensure that all user data are being monitored. In HTTP request processing, existing web intrusion detection systems audit each record separately, while our system considers the relationships amongst records in a time window. This is based on the premise that it is suspicious when the same user issues requests that cause lots of errors in a short period of time.

Experiments show that SensorWebIDS has higher detection rate for web intrusions than SNORT [Roesch, 1999] and mod_Security [Ristic, 2003] at a faster or similar detection speed. It has improved the accuracy and efficiency for detecting web intrusions with high flexibility and scalability.

Future work may extend the system to detect intrusions implanted with hacking tools and not through straight HTTP requests or intrusions embedded in non-basic resources like multimedia files and others, track illegal web users with their prior web access sequences, implement minimum and maximum values for integer data, and automate the process of pre-processing training data so that it is clean and free of intrusion for accurate detection results.

6 Acknowledgments

This research was supported by the Natural Science and Engineering Research Council (NSERC) of Canada under an operating grant (OGP-0194134).

References

[Agrawal & Srikant, 1994] Agrawal, R. and Srikant, R. 1994. Fast Algorithms for Mining Association Rules in Large Databases. In Proceedings of the 20th International Conference on very Large Databases, Santiago, Chile, pages 487–499.

[Bai & Kobayashi, 2003] Bai, Y. and Kobayashi, H. 2003, Intrusion Detection Systems: Technology and Development. IEEE Advanced Information and Networking Application Conference, New York.

[Barbara et al., 1999] Barbara, D. and Couto, J. and Jajodia, S. and Popyack, L. and Wu, N. 1999. ADAM: a testbed for exploring the use of data mining in intrusion detection. ACM SIGMOD Record, SPECIAL ISSUE: Special section on data mining for intrusion detection and threat analysis, 30(4), 15-24.

[CAIDA, 2003] CAIDA and ICSI and Silicon Defense and UC Berkeley EECS and UC SanDiego CSE. 2003, Analysis of the Sapphire Worm. http://www.caidaorg/analysis/security/sapphire/.

[Chandola et al., 2006] Chandola, V. and Eilertson, E. and Ertoz, L. and Simon, G. and Kumar, V. 2006, Data Mining for Cyber Security, Data Warehousing and Data Mining Techniques for Computer Security. Springer Verlag.

[Cohen, 1995] Cohen, W.W. 1995. Fast Effective Rule Induction. Proceedings of the 12th International Conference on Machine Learning, pp. 115-123, 1995.

[Ertoz et al., 2004] Ertoz, L. and Eilertson, E. and Lazarevic, A. and Tan, P. and Kumar, V. and Srivastava, J. and Dokas, P. 2004, MINDS - Minnesota Intrusion Detection System. In Next Generation Data Mining, Mit Press.

[Fielding et al., 1999] Fielding, R. and Irvine, UC. and Getty, J. and Compaq-W3C and Mogul, J. and Frystyk, H. and W3C/MIT and Masinter, L. and Xerox and Leach, P. and Microsoft and Berners-Lee, T. 1999, HTTP protocol specification:RFC2616. ftp://ftp.isi.edu/in-notes/rfc2616.txt.

[Hansen & Atkins, 1993] Hansen, S.E. and Atkins, E.T. 1993, Automated system monitoring and notification with swatch. Proceedings of the seventh Systems Administration Conference.

[Han et al., 2002] Han, H. and Lu, X.L. and Lu, J. and Bo, C. and Yong, R.L. 2002, Data mining aided signature discovery in network-based intrusion detection system. ACM SIGOPS Operating Systems Review, 30(4), pages 7-13.

[Network Associates, 1998] Network_Associates Inc. 1998, Cytercop server, http://www.nai.com/products/security/cybercopsvr/ index.asp.

[Lee, Stolfo & Mok, 1998] Lee, W. and Stolfo, S.J. and Mok, K.W., 1998. A Data Mining Framework for Building Intrusion Detection Models, Proceedings of the 7th USENIX Security Symposium, 3(4): 227-261.

[Lee, Stolfo & Mok, 1999] Lee, W. and Stolfo, S.J. and Mok, K.W., 1999. Mining in a Data-flow Environment: Experience in Network Intrusion Detection. Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, pages 114-124.

[Lee, Stolfo & Mok, 2002] Lee, W. and Stolfo, S.J. and Mok, K.W., 2002, Algorithms for Mining System Audit Data, 166-189, Physica-Verlag, Germany.

[McCanne & Jacobson] McCanne, S. and Jacobson, V., 1992, The BSD Packet Filter: A New Architecture for User-level Packet Capture, USENIX, Winter, 1992.

[Paxson, 1998] Paxson,Vern. 1999, Bro. Bro: A system for detecting network intruders in real-time. Proceedings of the 7th USENIX Security Symposium, San Antonio, TX", PAGES 31-51, USENIX Association.

[Risso & Degioanni, 1995] Risso, F. and Degioanni, L. 1995, An Architecture for High Performance Network Analysis. Proceedings of the 6th IEEE Symposium on Computers and Communications (ISCC 2001) Tunisia.

[Ristic, 2003] Ivan Ristic, Ivan. 2003, Introducing mod_security. http://www.onlamp.com/pub/a/apache/2003/11/26/ mod_security.html.

[Roesch, 1999] Roesch, M. 1999, Snort - Lightweight Intrusion Detection for Networks. http://www.snort.org/docs/lisapaper.txt.

[Spett, 2005] Spett, K. 2005, SQL Injection - Are your web application vulnerable?: Whitepaper, http://www.spidynamics.com/spilabs/education/ whitepapers /SQLinjection.html.

[Taylor, 1998] Taylor, G. 1998, Autobuse - Internet, http://www.picante.com/ gtaylor/autobuse.

[Tuininga, 1998] Tuininga, C. and Holak, R. 1998, Logscanner, http://logscanner.tradeservices.com/index.html.

[Wikepdia, 2002] Wikepedia, 2002, The Free Encyclopedia :HTTP cookie, http://en.wikipedia.org/wiki/.

[Cgisecurity, 2002] www.cgisecurity.com, 2002, The Cross Site Scripting FAQ, http://www.cgisecurity.com/articles/xss-faq.shtml.

[Impera, 2006] www.impera.com, 2006, What is Session Hijacking?, http://www.imperva.com/application_defense_center/ glossary/session_hijacking.html.