xMiner: Nip the Zero Day Exploits in the Bud

Muhammad Zubair Rafique and Muhammad Abulaish  

Center of Excellence in Information Assurance (CoEIA)  
King Saud University, Riyadh, Saudi Arabia  
Email: {zrafique.c,mabulaish}@ksu.edu.sa

Abstract—Vulnerability exploits present in malformed messages are one of the major sources to remotely launch malicious activities in different protocols. Sometimes, a single malformed message could be enough to crash remote servers or to gain unfettered access over them. In this paper, we propose the design of a generic vulnerability exploits detection system xMiner to detect malformed messages in real time for avoiding any network hazard. The proposed xMiner exploits the information embedded within byte-sequences of network messages. xMiner applies multi-order Markov process and principal component analysis (PCA) to extract novel discriminative features and uses them to detect attacks launched through malicious packets in real-time. The novelty of xMiner lies in its light-weight design which requires less processing and memory resources and makes it easily deployable on resource-constrained devices like smart phones. The system is evaluated on real-world datasets pertaining to three different protocols – HTTP, FTP and SIP. Five different classifiers are deployed to establish the effectiveness of the proposed system. On evaluation we found that the decision tree classifier performs well for HTTP and FTP datasets whereas, SVM shows highest performance in case of SIP packets.

Keywords—Network security; vulnerability exploits detection; feature extraction, machine learning.

I. INTRODUCTION

Remote attacks based on vulnerability exploitations are one of the most destructive security problems faced by the current security community. Particularly, the high profile security threat of automatic dissemination attacks or worms [1] are based on remote exploitation of vulnerabilities in compromised systems. One of the recent example of such attacks is the damaging Stuxnet worm in July 2010, which used four zero-day vulnerability exploitations to attack highly sensitive organizations (e.g. Iranian nuclear power plant) in many countries of the world [2]. Another example is the widely-spread Conficker worm in January 2009, which affected more than 15 million computers in more than 200 countries of the world [3]. Also, the successful eruption of well known CodeRed [4], Slammer [5], MSBlast [6], and Sasser [7] worms highlight the severity of damages caused in widely used services through vulnerability exploitation.

Beside the common web services, the emerging next generation networks like Voice over IP (VoIP) and IP Multimedia Subsystems (IMS) are also hot targets of remote exploitation attacks. In these services, the underlying flexibility of the application-specific protocols are exploited to launch remote attacks [8], [9]. The devastating impact of these attacks includes successful Denial of Services (DoS) attack, crashing of hosts or end points (e.g., Smart Phones), unfettered access to the systems and remote execution of malicious codes. Recent attacks like INVITE of Asterisks server [10], Remote DoS attack on Cisco Routers [11] and remote code execution in video-conferencing framework of Apple Mac OS [13] highlight the severity of damages that can be caused through vulnerability exploits. Unfortunately, patching as a first line of defense is not a perfect choice since it usually requires a service disruption (reboot of a system) and causes reduction in availability [1] which severely undermines the reliability of sophisticated real-time systems [1]. Additionally, it can not be guaranteed that the applied patch will not alter the behavior of the system. As an outcome, critical services remain susceptible to novel attacks and zero-day exploits for a considerable period of time which cannot be effectively remunerated by applying certain administrative or technical measures.

Several Intrusion Detection Systems (IDSs) like Snort [15] and Bro [16] and fire-walls like Hogwash [17] and Shield [1] are based on misuse detection where exploits are detected by modeling the signature of known attacks. The main drawback of signature-based techniques is that they cannot cope-up with exponential increase in new malicious exploits. Not only the size of signatures database will not scale but the time to match signatures also significantly increases [18], [19]. Last but not least, generation of signatures for one particular protocol or application can not be used in detection of zero-day exploits. Therefore, we argue that the design of light-weight and non-signature based intrusion detection systems to identify zero-day attacks in a real-time environment is one of the novel research problems.

In this paper, we propose the design of a light-weight intrusion detection system, xMiner, which applies supervised-learning approach to detect malicious messages. The proposed system is efficient and capable to detect vulnerability exploits targeting towards next generation services at the granularity of application layer. One of the motivating

1A reliable VoIP infrastructure must guarantee at least 99.9% uptime to stay competitive in the telecommunication market. VoIP servers are among the SANS top 20 Security Risks [14].
factors to target application-specific services is that they are widely used, constantly emerging and more vulnerable to zero-day attacks. Instead of using syntax-level features to model classification system, xMiner extracts byte-level features using multi-order Markov process and uses them to identify malicious messages. PCA (Principal Component Analysis) is applied to reduce the dimension of feature space and eliminate less discriminative features. Some of the novel features of xMiner are summarized as follows:

- **Generality**: Since xMiner uses byte-level features instead of syntax-level features, it can be applied on wide-range of protocols to identify malicious messages.
- **Efficiency and real-time deployability**: The efficiency of xMiner lies in its reduced set of features that are obtained using multi-order Markov process and principal component analysis. In addition, due to its design as a non-signature based IDS, for predicting the class (benign or malicious) of an incoming message xMiner is not subjected to search through a large list of signatures. Rather, it uses the trained model to predict the class of an incoming message efficiently and consequently, xMiner is real-time deployable. On experimentation, we found that for some protocols, xMiner achieves detection accuracy of more than 99% and false alarm rate ≤ 0.1%.
- **Novel attack detection**: Since, rather than using signature matching approach, xMiner models a classification system to characterize malicious and benign messages and uses the same for detection purpose, it can easily detect novel (zero-day) attacks in the network.
- **Modularity**: The modular design approach of xMiner allows its simple yet effective deploying functionality. Due to this feature, xMiner is easily configurable for different types of services.

We have evaluated the xMiner on three different real-world datasets related to three different services HTTP, FTP and SIP. We have deployed a real-world testbed to generate various vulnerability exploits using different types of security testing tools and scripts. The attack vectors include real-world SQL-Injections, buffer over flows, remote code execution, remote DoS and fuzzed message exploits. On evaluation, we found that in some cases xMiner achieves more than 99% detection rate and less than 0.1% false alarm rate for distinguishing benign messages from malicious messages.

The rest of the paper is organized as follows. Section II presents a summarized view of the related works on malicious message detection. The functioning detail of xMiner is presented in Section III. The experimental setup and evaluation results are presented in Section IV. Finally, we conclude the paper with an outlook to our future works in Section V.

### II. RELATED WORK

In this section, we present a brief review of the existing network intrusion detection techniques proposed by different researchers. The protection against application-level attacks using protocol syntax was first purposed in signature-based intrusion detection systems. In [16], Paxson has proposed a signature-based IDS, Bro, which uses different protocol parsers to identify malicious packets. The parsers developed are tightly coupled with the Bro’s signature engine and can not be used for wide range of services. Similarly, Roesch has proposed Snort in [15] which is also a signature-based intrusion detection system. Snort can perform protocol analysis, content searching/matching and can be used to detect a variety of attacks targeting towards servers. Some other signature-based intrusion detection techniques have been used by Niccolini et al. [23] and Apte et al. [24] specifically for SIP protocol. Genciatakis et al. [25] have also proposed signature generating methodology that prevent fuzzed message attacks on VoIP. The other signature-based techniques like binpac [26] and GAPAL [27] provide effective and generic procedures for detecting malicious packets by using protocol parsers. However, most of these techniques are dependant on signature database and hence cannot be used to detect novel attacks. Düssel et al. [28] have proposed a different approach which focuses on analyzing the payloads of application-level protocols for anomaly detection. The approach detects anomalous packets by computing similarity between the attributed n-grams/ tokens derived from the protocol grammar. Ingham et al. [29] have proposed a learning system on specific presentation by extracting token-based feature through delimiters specific to HTTP requests. A related work by Rieck et al. [30] proposes the design of a self-learning system for detecting malformed messages in SIP. The self-learning model operates on the tokens and n-gram based features, and the learned model from higher values of n-gram features performs better than the token-based attribute model. Kruegel et al. [31] have developed an IDS for HTTP, which uses a number of features like length, character distribution, etc. to detect malicious packets.

In contrast to the above-mentioned intrusion detection systems, xMiner uses byte-level transitions of network messages and applies multi-order Markov process to extract discriminative features. Further, it applies PCA on the extracted set of features to filter-out irrelevant features and reduce the dimension of the feature space. This boost the efficiency of the proposed method drastically and makes it deployable for real-time environment. Moreover, the use of machine learning approach to model the characteristics of benign and malicious packets makes xMiner capable to identify zero-day attacks.

---

2A market survey indicates that VoIP accounts for 49.7% of total voice traffic at the end of year 2007 [20].
III. ARCHITECTURE OF xMINER

In this section, we present the architectural detail of the proposed xMiner to detect vulnerability exploits in different types of application-layer protocols. xMiner is modular in nature and it is developed through analytical research of relevant issues in an engineering fashion. We systematically analyzed different potential solutions and then chose the one capable of meeting our challenges. This modular architecture allows our system to function on different servers and NGNs architectures. The main functionalities of xMiner are (a) message analysis, (b) feature extraction, (c) feature selection, and (d) model learning and vulnerability exploits detection. (see Figure 1). Further details about these functionalities are presented in the following sub-sections

A. Message Analysis

The main goal during this phase is to analyze network traffics based on the syntactical formation of the underlying application-specific protocol. The message analyzer module analyzes request and response messages of the underlying protocol to extract discriminative features from them. For network protocols like HTTP, FTP, or SIP, packets are captured in form of a raw byte payloads. The structured information of underlying protocols is then acquired by converting raw byte payloads of network traffic using binpac [26]. A sample INVITE request for a SIP protocol, which is used in SIP-based VoIP infrastructure to set up communication session between SIP clients is shown in Figure 2. The overflow of the colons in the second line is mangled to create a buffer overflow exploit. It is important to note that the control information in SIP header is ASCII conforming and contains the necessary information for session setup. The crafty attacker can fuzz different fields in the request message to exploit vulnerability in VoIP servers, which can lead to call processing delays, an unauthorized access or a complete denial of service [8]. It is not possible to predict in advance which fuzzed field can result in a denial of service attack. Therefore, we have considered the complete incoming messages as a syntactical input for our feature extraction module. The analysis is performed on the complete syntactical formation of the response and request messages.

B. Feature Extraction

A key challenge in extracting features for generic detection of exploits is to make the system adaptable for both text-based and binary protocols. Hence, the tight coupling of feature-set with the formation and syntax of specific protocol is not a feasible choice for our system to operate in diverse environment. Therefore, we have not considered the token-based feature extraction scheme [30] in which special delimiters are used to obtain the features’ strings for classification of attack instances in a specific protocol. Rather, xMiner treats a message $M_p$ of a specific application layer protocol $P$, as an order of elements $\Theta_{M_p} = [\Theta_1 M_p, \ldots, \Theta_l M_p]$, where $\Theta_{M_p}$ represents an order of byte values in $M$ and $l$ is the length of the message in bytes. Without loss of generality, we can also incorporate every consecutive $n$ bytes in $M_p$ as a distinctive feature. For example, if $\Theta_{M_p} = [b_1, b_2, b_3, b_4]$, and we consider two consecutive bytes as a feature ($n = 2$), we get the feature-set as $[b_1 b_2, b_2 b_3, b_3 b_4]$. The up-scaling

![Figure 1. Architecture of xMiner](image1)

![Figure 2. SIP R-DOS INVITE of death](image2)
results in higher dimensional feature space of the byte-level distribution of $M_p$. The large value of $n$ increases the underlying knowledge information and may result in sparse distribution in case the size of training data is not sufficiently large. Alternatively, the small value of $n$ increases the probability of false detection resulting in reduced system performance. Note that the order $n = k$ (where $k > 1$) is simply a joint distribution of byte values with order $n = k - 1$. The up scaling in joint distribution of byte-level features may contain redundant information that are useful for accurate analysis but, at the cost of processing overhead [32]. To this end, we apply number of statistical measures to quantify the order of byte sequence in different protocol messages. A correlation analysis of byte sequences provides valuable insights about the order of their joint distribution. Thereafter, the concept of multi-order Markov process is used to model the information contained in the byte-sequences of the messages. These steps are further explained in the following sub-sections.

1) **Correlation Analysis of Byte-Sequences:** Autocorrelation is used to study the correlation between the random variables in a stochastic process at different points in time or space. Mathematically, the autocorrelation function of a stochastic process $X_z$ (where $z$ is a time/ space index), for a given lag $e$, is defined using equation 1 in which $E\{\cdot\}$ is the expected value operator and $\rho_{X_z}$ is the standard deviation of the random variable $X_z$ at time/ space lag $z$.

$$\rho[e] = \frac{E\{X_0 X_e\} - E\{X_0\} E\{X_e\}}{\rho_{X_0}\rho_{X_e}} \tag{1}$$

The autocorrelation value lies in the range [-1, 1], where $\rho[z] = 1$ means perfect correlation at lag $z$ (which is obviously true for $n = 0$) and $\rho[z] = 0$ means no correlation at all at lag $z$.

To examine the reliance of byte-level sequences in $M_p$, we calculate sample autocorrelation functions for benign and exploit request messages of different services. Figure 3 shows the sample autocorrelation functions plotted versus the lag of HTTP, SIP and FTP messages. It can be noted in Figure 3 that the byte-sequences in request and response messages of different protocols usually follows a $1^{st}$, $2^{nd}$ and $3^{rd}$ order reliance because the autocorrelation shows peak at $n = 1, 2, 3$ for both benign and exploit messages of all three protocols. This property helps us to model the byte-sequences of messages in detecting zero day exploits.

2) **Modeling Byte Sequences using Multi-Order Markov Process:** Application protocols are specified by defined syntax (sequences of bytes) of their request and response messages. Due to dependencies among the bytes of protocol messages, we model the byte-sequences using Markov chain. The Markov chain uses conditional distribution instead of joint distribution, which results in small sample space, eliminating the redundant information in underlying feature space. A $k^{th}$ order Markov chain of $S$ states can be used to model byte transition probabilities of network messages in a transition matrix $T$ as given in equation 2. In this equation, each state $s_i$ corresponds to byte value $b_i$. The transition to state $s_{i+1}$ from the state $s_i$ can be computed as $t_{\{s_i, s_{i+1}\}}$ and the probability of state transition as $P_{\{s_i, s_{i+1}\}}$. The underlying assumption is that the probability of each state transition depends only on the previous state values, i.e., in a $k^{th}$ order Markov chain, the value of state $s_i$ can be determined by using the values of the previous states $s_{i-k}, \ldots, s_{i-1}$.

$$T_k = \{P_{s_0, \ldots, s_k}\}, \tag{2}$$

A simple (single-order) Markov chain representing the transition probabilities of a byte sequence can be a good choice to model protocol messages when dependencies are homogenous in byte-sequences. But, in our case the heterogeneous nature of byte-sequences in different protocol messages, which results in reduced system performance. Note that the order $n = k$ (where $k > 1$) is simply a joint distribution of byte values with order $n = k - 1$. The up scaling in joint distribution of byte-level features may contain redundant information that are useful for accurate analysis but, at the cost of processing overhead [32]. To this end, we apply number of statistical measures to quantify the order of byte sequence in different protocol messages. A correlation analysis of byte sequences provides valuable insights about the order of their joint distribution. Thereafter, the concept of multi-order Markov process is used to model the information contained in the byte-sequences of the messages. These steps are further explained in the following sub-sections.

1) **Correlation Analysis of Byte-Sequences:** Autocorrelation is used to study the correlation between the random variables in a stochastic process at different points in time or space. Mathematically, the autocorrelation function of a stochastic process $X_z$ (where $z$ is a time/ space index), for a given lag $e$, is defined using equation 1 in which $E\{\cdot\}$ is the expected value operator and $\rho_{X_z}$ is the standard deviation of the random variable $X_z$ at time/ space lag $z$.

$$\rho[e] = \frac{E\{X_0 X_e\} - E\{X_0\} E\{X_e\}}{\rho_{X_0}\rho_{X_e}} \tag{1}$$

The autocorrelation value lies in the range [-1, 1], where $\rho[z] = 1$ means perfect correlation at lag $z$ (which is obviously true for $n = 0$) and $\rho[z] = 0$ means no correlation at all at lag $z$.

To examine the reliance of byte-level sequences in $M_p$, we calculate sample autocorrelation functions for benign and exploit request messages of different services. Figure 3 shows the sample autocorrelation functions plotted versus the lag of HTTP, SIP and FTP messages. It can be noted in Figure 3 that the byte-sequences in request and response messages of different protocols usually follows a $1^{st}$, $2^{nd}$ and $3^{rd}$ order reliance because the autocorrelation shows peak at $n = 1, 2, 3$ for both benign and exploit messages of all three protocols. This property helps us to model the byte-sequences of messages in detecting zero day exploits.

2) **Modeling Byte Sequences using Multi-Order Markov Process:** Application protocols are specified by defined syntax (sequences of bytes) of their request and response messages. Due to dependencies among the bytes of protocol messages, we model the byte-sequences using Markov chain. The Markov chain uses conditional distribution instead of joint distribution, which results in small sample space, eliminating the redundant information in underlying feature space. A $k^{th}$ order Markov chain of $S$ states can be used to model byte transition probabilities of network messages in a transition matrix $T$ as given in equation 2. In this equation, each state $s_i$ corresponds to byte value $b_i$. The transition to state $s_{i+1}$ from the state $s_i$ can be computed as $t_{\{s_i, s_{i+1}\}}$ and the probability of state transition as $P_{\{s_i, s_{i+1}\}}$. The underlying assumption is that the probability of each state transition depends only on the previous state values, i.e., in a $k^{th}$ order Markov chain, the value of state $s_i$ can be determined by using the values of the previous states $s_{i-k}, \ldots, s_{i-1}$.

$$T_k = \{P_{s_0, \ldots, s_k}\}, \tag{2}$$

A simple (single-order) Markov chain representing the transition probabilities of a byte sequence can be a good choice to model protocol messages when dependencies are homogenous in byte-sequences. But, in our case the heterogeneous nature of byte-sequences in different protocol messages, which results in reduced system performance. Note that the order $n = k$ (where $k > 1$) is simply a joint distribution of byte values with order $n = k - 1$. The up scaling in joint distribution of byte-level features may contain redundant information that are useful for accurate analysis but, at the cost of processing overhead [32]. To this end, we apply number of statistical measures to quantify the order of byte sequence in different protocol messages. A correlation analysis of byte sequences provides valuable insights about the order of their joint distribution. Thereafter, the concept of multi-order Markov process is used to model the information contained in the byte-sequences of the messages. These steps are further explained in the following sub-sections.

1) **Correlation Analysis of Byte-Sequences:** Autocorrelation is used to study the correlation between the random variables in a stochastic process at different points in time or space. Mathematically, the autocorrelation function of a stochastic process $X_z$ (where $z$ is a time/ space index), for a given lag $e$, is defined using equation 1 in which $E\{\cdot\}$ is the expected value operator and $\rho_{X_z}$ is the standard deviation of the random variable $X_z$ at time/ space lag $z$.

$$\rho[e] = \frac{E\{X_0 X_e\} - E\{X_0\} E\{X_e\}}{\rho_{X_0}\rho_{X_e}} \tag{1}$$

The autocorrelation value lies in the range [-1, 1], where $\rho[z] = 1$ means perfect correlation at lag $z$ (which is obviously true for $n = 0$) and $\rho[z] = 0$ means no correlation at all at lag $z$.

To examine the reliance of byte-level sequences in $M_p$, we calculate sample autocorrelation functions for benign and exploit request messages of different services. Figure 3 shows the sample autocorrelation functions plotted versus the lag of HTTP, SIP and FTP messages. It can be noted in Figure 3 that the byte-sequences in request and response messages of different protocols usually follows a $1^{st}$, $2^{nd}$ and $3^{rd}$ order reliance because the autocorrelation shows peak at $n = 1, 2, 3$ for both benign and exploit messages of all three protocols. This property helps us to model the byte-sequences of messages in detecting zero day exploits.
messages is evident in the autocorrelation analyses. Therefore, we argue that the extraction of information from byte-sequences using a single-order Markov model may not be a perfect choice to model all possible discriminative features necessary for detecting exploits. It should be noted that the order of a Markov chain represents the extent to which past states determine a present state, i.e., how many lags should be examined to analyze higher order sequences [32]. Since, our correlation analysis shows $1^{st}$, $2^{nd}$ and $3^{rd}$ order dependencies in byte sequences of protocol messages, we have used up to three-order Markov chain to model byte-level information. In this process, we combine transition matrices $T_k$ of multiple orders to achieve more comprehensive modeling of the dependencies in byte-sequences. The transformation function $\mathcal{Y}$ operates on the bytes of each incoming packet $M_p$ and embeds spatial information in the form of probability values to multiple order state transitions. For a given dataset of Protocol $P$ containing $n$ messages, the multi-order Markov process for $i^{th}$ message in the dataset is computed using equation 3 in which $i = 1, \ldots, n$, $K$ is the set of values representing Markov chain orders, and pre-subscript $c$ represents concatenation operator, which combines all $T_k$’s into one matrix. For example, the output of transformation function $\mathcal{Y}$ for $K = 1, 2, 3$ can be obtained using equations 4 to 7:

$$\mathcal{Y}(m_i) \rightarrow \{c T_k^i\}_{k \in K}$$  (3)

$$\mathcal{Y}(m_i) = \begin{bmatrix} \hat{T}_1^i \\ \hat{T}_2^i \\ \hat{T}_3^i \end{bmatrix}$$  (4)

$$\hat{T}_1^i = [p_{s_0}^i, p_{s_1}^i, \ldots, p_{s_{255}}^i]$$  (5)

$$\hat{T}_2^i = \begin{bmatrix} p_{s_0,s_0}^i & \cdots & p_{s_0,s_{255}}^i \\ \vdots & \ddots & \vdots \\ p_{s_{255},s_0}^i & \cdots & p_{s_{255},s_{255}}^i \end{bmatrix}$$  (6)

$$\hat{T}_3^i = \begin{bmatrix} p_{s_0,s_0}^i & \cdots & p_{s_0,s_{255}}^i \\ \vdots & \ddots & \vdots \\ p_{s_{255},s_0}^i & \cdots & p_{s_{255},s_{255}}^i \end{bmatrix}$$  (7)

The transitions in byte-sequences incorporates the underlying information from the protocol messages. Any deviation in transition values reflects a different composition of incoming request or response message. This situation indicates an anomalous message, possibly an exploit message received by the server. Since, in the transition matrix not all transitions contain valuable information to detect exploits messages, we remove them from further consideration to reduce computation overheads.

### C. Feature Selection

In this section, we present a discussion about our feature selection process. Having too many features often results in the problem of having too many degrees of freedom leading to poor statistical coverage and thus poor generalization. In addition, each feature adds to a computational burden in terms of processing and storage. Hence, feature selection is an essential step to filter out non-discriminative features from the feature set and thereby to reduce computational overheads. For this purpose, we have used the concept of Principal Component Analysis (PCA) which is a statistical method for dimension reduction. PCA maps high-dimensional data points onto a lower-dimensional set of axes that best explain the variance observed in the dataset.

### D. Model Learning and Vulnerability Exploits Detection

Once we have identified the relevant features through applying multi-order Markov process followed by principal component analysis on training dataset, we map every instance of training data into a feature vector to learn classification models. A large body of literatures related to anomaly detection [33], [34], [30] or anomalous network payloads [35], [31], [36], [37] share the common thesis: anomalies are characterized as deviations from a learnt “normal model”. In this paper, we have trained five different classifiers – Naïve Bayes (NB), decision tree (J48), inductive rule learner (RIPPER), instance based learner (k-NN), and Support Vector Machine (SVM) using sequential minimal optimization to model the normal behaviors of malicious and benign messages. For model learning we have used Weka\(^3\), which is a collection of machine learning algorithms for data mining tasks. Once the classification models are trained on training data they can be easily deployed in real-time to detect malformed messages containing possible vulnerability exploits.

### IV. Experimental Setup and Results

In this section, we present the experimental setup including the dataset description and implementation details of xMiner. The evaluation results of xMiner on different datasets is also presented in this section.

#### A. Dataset Description

In order to analyze the detection capability of xMiner on real-world attacks launched through various vulnerability exploits, we have collected four real-world traces containing two HTTP, one FTP and one SIP dataset. The first dataset

---

\(^3\)An open source software issued under the GNU General Public License and can be downloaded from the URL: http://www.cs.waikato.ac.nz/ml/weka/
The FTP dataset is generated by simulating 10,000 clients on Microsoft IIS server with variable user names and passwords. For real-world SIP dataset, we contacted a VoIP vendor that has a customer base in North America. We developed a SIP traffic logger and deployed it on their SIP server and collected a SIP traffic log of more than 20 days containing 4,000 legitimate SIP request and response messages. The logs contain traces of SIP dialogs among several SIP terminals as well as SIP dialogs among various network nodes of VoIP infrastructure.

The exploits are collected by using Metasploit framework as well as from common security mailing lists xssed.com, sla.ckers.org and Bugtraq. Some of the exploits used in this study are shown in Table I. We have also used Hszp fuzzer by Krakow Labs [38]. We have launched attacks of malformed messages on Microsoft Internet Information Services (IIS) HTTP server and sniffed 5,000 fuzzed messages through our attack generating machine. This includes request and response fuzzing, authentication fuzzing and query parameter fuzzing. Similarly, we have generated 2,000 fuzzed FTP packets using FTP Fuzzer by INFIGO Information Security [39], which is a GUI based fuzzing tool for bench marking the performance of FTP servers. The fuzzing tests covered by the FTP Fuzzer unveiled a number of security vulnerabilities (overflows, format strings) in various implementations of FTP servers [40]. Finally, we have used SIP Security Evaluation Tool [41] for generating 30,000 malicious SIP messages. The tool is well known for discovering INVITE of Death vulnerability in the SIP stack of an open source SIP server [10]. We have normalized the attack instances by using the similar structure of benign messages injected to ensure that no obvious artifacts are introduced in attack dataset that make detection intuitively simple.

### Table 1

#### REAL-WORLD EXPLOITS OF HTTP, FTP AND SIP PROTOCOLS

<table>
<thead>
<tr>
<th>Table</th>
<th>ID</th>
<th>LVE</th>
<th>Description</th>
<th>Type</th>
<th>ID</th>
<th>LVE</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>1</td>
<td>2001-0590</td>
<td>ISS (IDQPath)</td>
<td>Buf</td>
<td>13</td>
<td>2005-4085</td>
<td>Blue Coat WebPony Host Header</td>
<td>Buf</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2001-0524</td>
<td>ISS (Primary Host Header)</td>
<td>Buf</td>
<td>14</td>
<td>2007-0774</td>
<td>Apache mod_pl 1.2.0</td>
<td>Buf</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2004-1134</td>
<td>RS/RS (w who.dll Query )</td>
<td>Buf</td>
<td>15</td>
<td>2005-0395</td>
<td>RadBle 2.5 EXT.dll</td>
<td>Buf</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2003-0109</td>
<td>ISS (WildDot ads.dll Path)</td>
<td>Buf</td>
<td>16</td>
<td>2005-0377</td>
<td>RadBle 2.729 Putsch</td>
<td>Buf</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2003-1192</td>
<td>IA WebMod 3.x</td>
<td>Buf</td>
<td>17</td>
<td>2005-1190</td>
<td>CA Technology (deny) Debug Mode</td>
<td>Buf</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>2005-0116</td>
<td>AllScan Remote Command Execution</td>
<td>Web</td>
<td>19</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>2003-0471</td>
<td>Al S WebAdmin Use</td>
<td>Buf</td>
<td>20</td>
<td>2007-1408</td>
<td>BM-TM 3.1.0 x64.exe</td>
<td>Buf</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2006-1148</td>
<td>PerlCat (c&lt;01216) URL Handling</td>
<td>Buf</td>
<td>22</td>
<td>2005-1348</td>
<td>MailEnable Authentication Header</td>
<td>Buf</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>2006-2847</td>
<td>SFP QoSView NIM-CGI</td>
<td>Buf</td>
<td>23</td>
<td>2004-2271</td>
<td>MiniMote 1.4.x</td>
<td>Web</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>2007-5667</td>
<td>Xamms Web Server</td>
<td>Buf</td>
<td>24</td>
<td>2006-5112</td>
<td>NaviCOPA 2.0.1 URL Handling</td>
<td>Buf</td>
</tr>
<tr>
<td>FTP</td>
<td>1</td>
<td>2006-3217</td>
<td>3web WebMatrix 2.0 FTP Intrusion</td>
<td>Buf</td>
<td>10</td>
<td>2006-3217</td>
<td>3web WebMatrix</td>
<td>Buf</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2006-2961</td>
<td>Cisco FTP 0.995 MKD Command</td>
<td>Buf</td>
<td>11</td>
<td>2005-2753</td>
<td>Snim/SNP LIST Concatenation</td>
<td>Buf</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2004-2074</td>
<td>Boltech Dream FTP Server 1.12</td>
<td>Buf</td>
<td>12</td>
<td>1999-0256</td>
<td>War-FTP 1.65 Password</td>
<td>Buf</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2006-3726</td>
<td>FileCops FTP Server pre 18 Jul Version</td>
<td>Buf</td>
<td>14</td>
<td>2006-4118</td>
<td>Texas Imperial Software WFTP 2.35 SIZE</td>
<td>Buf</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>2006-2407</td>
<td>ProFTP 1.0.10 Key Exchange</td>
<td>Buf</td>
<td>15</td>
<td>2004-1135</td>
<td>Win-FTP Server 3.03 MKD</td>
<td>Buf</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>2005-3683</td>
<td>IsoFTP 1.0 Username</td>
<td>Buf</td>
<td>16</td>
<td>2006-3952</td>
<td>Easy File Shaping FTP Server 2.0 PASS</td>
<td>Buf</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>2005-1235</td>
<td>NuFenx NoFTP USER</td>
<td>Buf</td>
<td>18</td>
<td>2006-4474</td>
<td>IsoFTP Win-FTP Server 3.03 XMD</td>
<td>Buf</td>
</tr>
<tr>
<td>SIP</td>
<td>1</td>
<td>2003-2267</td>
<td>Cisco 12.2(A)0(A)CE_CGI</td>
<td>R-DeS</td>
<td>10</td>
<td>2005-1112</td>
<td>HNtsc Fuzz (1.1.3)</td>
<td>RCE</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>2005-3683</td>
<td>Cisco IOS 12.2 through 12.4</td>
<td>R-DeS</td>
<td>11</td>
<td>2005-1112</td>
<td>HNtsc Fuzzzy-Voll (1.0.1)</td>
<td>RCE</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2007-4292</td>
<td>Cisco IOS 12.2 through 12.5</td>
<td>R-DeS</td>
<td>12</td>
<td>2005-1112</td>
<td>IPEf OpenSSH &lt;0.6.1</td>
<td>RCE</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>2005-2664</td>
<td>Apple Mix-DOS 6.11.0.1.0.4 8</td>
<td>RCE</td>
<td>13</td>
<td>2003-1099</td>
<td>Cisco IP Phone Model 7940</td>
<td>RCE</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>2005-1306</td>
<td>Antwink (v1.41 do) &amp;&lt; v1.26</td>
<td>R-DeS</td>
<td>14</td>
<td>2003-1114</td>
<td>Mediacom Telecom (SIPS) v2.643</td>
<td>RCE</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>2005-1973</td>
<td>INVITE of Death (OpenSSL 1.1.26 )</td>
<td>R-DeS</td>
<td>15</td>
<td>2003-1099</td>
<td>Cisco IP Phone Model 7960</td>
<td>RCE</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>2005-0961</td>
<td>Cisco PIX 500 and ASA 5500</td>
<td>R-DeS</td>
<td>16</td>
<td>2003-1111</td>
<td>DynamicAccess products</td>
<td>RCE</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>2005-1108</td>
<td>Alcatel OmniPCX Enterprise 5.0 Lx</td>
<td>RCE</td>
<td>17</td>
<td>2003-1111</td>
<td>Natural Networks SC Server 2000</td>
<td>RCE</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>2003-1110</td>
<td>Columbia SIP User Agent (xsp)</td>
<td>RCE</td>
<td>18</td>
<td>2006-1973</td>
<td>Linksys RT31P2 VoIP router</td>
<td>R-DeS</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>2005-1346</td>
<td>Ethereal before 0.10.11</td>
<td>Buf</td>
<td>19</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

(KSU11) contains 50,000 normal HTTP connections logged from the web server of our institute. The second dataset (CoEIJA11) has been collected from the web server of our research center that contains “pure” web application data. This dataset is generated through capturing HTTP traffic for a period of two weeks and contains 37,000 requests.

In our experiments, the purpose is to judge the classification accuracy of xMiner to distinguish between benign and exploit packets. We have done the experiments on an Intel(R) Core(TM) i7 920 @ 2.67 GHz processor with 12 GB RAM and 64 bit operating system (Windows 7). For each dataset, we have extracted the features using multi-order Markov process and PCA, and then trained five different classification models namely Naïve Bayes (NB), decision tree (J48), inductive rule learner (RIPPER), instance based learner (k-NN), and Support Vector Machine (SVM) using sequential minimal optimization. From the classification results, we calculate the true positive TP (number of malicious packets the system identifies as malicious), the false positive FP (number of benign packets the system identifies as malicious), true negative TN (number of benign packets the system identifies as benign), and the false negatives FN (number of malicious packets the system identifies benign). By using these values we calculate the standard performance measures true positive rate (TPR) and false positive rate (FPR), which are defined in equations

\[
\text{TPR} = \frac{TP}{TP + FN} \\
\text{FPR} = \frac{FP}{FP + TN} \\
\]
8 and 9. The true positive rate and false positive rate are also called detection accuracy and false alarm rate respectively.

\[
TPR = \frac{TP}{TP + FN} \tag{8}
\]

\[
FPR = \frac{FP}{FP + TN} \tag{9}
\]

We have used the stratified 10-fold cross-validation for performance evaluation of xMiner, i.e., for each category of data the dataset is randomly divided into 10 smaller subsets, out of which 9 subsets are used for training and 1 subset is used for testing. This process is repeated 10 times for every dataset. For each category of data, the TPR and FPR values obtained for different type of classifiers are shown in Table II.

It can be be observed from the results presented in Table II that xMiner achieves the detection rate of more than 0.99 for most of the application protocols and even approaching 1 in case of SIP packets. We have also found during experimentation process that the feature selection using PCA improves the detection capability of xMiner. On average, with the help of PCA the TPR is improved by 2% and the FPR is reduced by 3% in detecting malicious exploits of different application services. In our experiments, we have also evaluated the effectiveness of different classifiers on selected features from different datasets. It can be observed from Table II that the TPR and FPR of the decision tree classifier (J48) is optimum for HTTP and FTP packets whereas, the SVM performs well in case of SIP packets with respect to other classifiers.

V. CONCLUSION AND FUTURE WORK

In this paper, we have proposed the design of a generic light-weight vulnerability exploits detection system xMiner to detect malformed packets in real-time. xMiner operates on the byte sequences of network messages and model them using multi-order Markov process. It also applies PCA to reduce the dimensionality of feature space. xMiner is tested on real datasets of HTTP, FTP and SIP messages. On experimentation, we found that xMiner successfully detects exploit messages with average detection accuracy of 99% and false alarm rate of less than 0.1% for different application-specific services. This makes xMiner deployable to protect application servers and devices from real-world threats through effective filtering/ blocking malformed massages. As a result, the threat regarding instant unavailability of a service can be mitigated. Presently, we are working to enhance xMiner for SMS protocol.

REFERENCES


### Table II

<table>
<thead>
<tr>
<th></th>
<th>HTTP</th>
<th>FTP</th>
<th>SIP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TPR</td>
<td>FPR</td>
<td>TPR</td>
</tr>
<tr>
<td>NB</td>
<td>0.977</td>
<td>0.056</td>
<td>0.928</td>
</tr>
<tr>
<td>J48</td>
<td>0.994</td>
<td>0.003</td>
<td>0.98</td>
</tr>
<tr>
<td>RIPPER</td>
<td>0.992</td>
<td>0.008</td>
<td>0.971</td>
</tr>
<tr>
<td>K-NN</td>
<td>0.989</td>
<td>0.004</td>
<td>0.997</td>
</tr>
<tr>
<td>SVM</td>
<td>0.992</td>
<td>0.011</td>
<td>0.936</td>
</tr>
<tr>
<td>Average</td>
<td>0.9888</td>
<td>0.0164</td>
<td>0.9624</td>
</tr>
</tbody>
</table>


