Of daemons and men: A file system approach towards intrusion detection

G. Mamalakis *, C. Diou, A.L. Symeonidis, L. Georgiadis

Department of Electrical and Computer Engineering, Aristotle University of Thessaloniki, Greece

A R T I C L E   I N F O

Article history:
Received 7 August 2013
Received in revised form 16 May 2014
Accepted 29 July 2014
Available online 17 September 2014

Keywords:
Intrusion detection systems
Information security
Machine learning
Data mining
File system
Anomaly detection

A B S T R A C T

We present FPDS a file system, host based anomaly detection system that monitors Basic Security Module (BSM) audit records and determines whether a web server has been compromised by comparing monitored activity generated from the web server to a normal usage profile. Additionally, we propose a set of features extracted from file system specific BSM audit records, as well as an IDS that identifies attacks based on a decision engine that employs one-class classification using a moving window on incoming data. We have used two different machine learning algorithms, Support Vector Machines (SVMs) and Gaussian Mixture Models (GMMs) and our evaluation is performed on real-world datasets collected from three web servers and a honeynet. Results are very promising, since FPDS detection rates range between 91% and 95.9% with corresponding false positive rates ranging between $8.1 \times 10^{-2}$ and $9.3 \times 10^{-4}$%. Comparison of FPDS to another state-of-the-art filesystem-based IDS, FWRAP, indicates higher effectiveness of the proposed IDS in all three datasets. Within the context of this paper FPDS is evaluated for the web daemon user; nevertheless, it can be directly extended to model any daemon-user for both intrusion detection and postmortem analysis.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

1.1. About intrusion detection

In contemporary computer and communication networks almost everybody uses the Internet backbone to exchange personal or sensitive information in a daily basis. Through cellphones, laptops, net-pads and smart sensors people exchange data on top of various types of applications like email services, web transactions, social networks, file transfers, etc. In tandem with this extreme growth of information exchange via the Internet, cyber-threats evolve to exploit the expanding attack surface. In order to protect end-users, specialized software and hardware solutions have been deployed (firewalls, antiviruses, spam detectors, Intrusion Detection Systems, sandboxes, etc). It is common knowledge, though, that none of these solutions alone is enough to offer absolute protection; usually a combination of them is essential for providing an adequate level of cyber-protection. Intrusion detection systems (IDS), known as the “computer world’s burglar alarm”, are based on the idea of identifying attacks when or after they occur, and fire an alarm or take some action – Intrusion Response Systems (IRS) – according to their configuration. Different types of IDS have been proposed during the last two and a half decades. Network IDS (NIDS) [3,10,13,28,36] monitor network activity, while host-based IDS (HIDS) monitor data generated within a host, including command histories [15], system calls [2,6], function calls [26] and file system data [34]. Hybrid IDS [16,40] monitor both network and host activity. Misuse IDS [8,14] are trained with malicious data to identify attacks, while anomaly-based IDS [6,7] raise an alarm whenever monitored activity diverges from a normal usage profile. It is not uncommon to use distributed architectures of different IDS types in order to enhance the security perimeter of large computer networks.

1.2. Our approach

We have built a system that is able to distinguish the man from the daemon on a running server. In the following paragraphs we describe how such a system can be used to allow intrusion identification of compromised daemons.

As most experienced system administrators know, no matter how well configured, hardened and up-to-date a running system
offering network services is, it still has a chance of being compromised (it only takes a clumsy PHP developer to create an easily exploitable vulnerability even on the most protected web server). On a server system the attack surface consists mainly of the public services it offers (HTTP, FTP, SSH, etc.) and furthermore, attackers exploiting a vulnerability on a service are usually rewarded with a remote shell running with the privileges of the exploited daemon-user. Once this type of access is granted, attackers are allowed to run arbitrary commands as this daemon user. Hence, one way to identify when a daemon has been compromised is to monitor the daemon process for abnormal behaviour.

We formulate our objective as a machine learning problem and try to identify novel features that are informative enough for various HIDS anomaly detection algorithms. Within the context of this work we argue that this distinction of daemon-human behaviour can be accomplished, since daemons usually behave in a very specific and repetitive way, serving web pages from certain paths or delivering mails to pre-configured mailboxes. Attackers, on the other hand, after compromising the vulnerable daemon usually perform actions like searching the system or network for further vulnerabilities, inspecting the system to identify its architecture and OS version, cleaning log files from their trails, downloading additional utilities to gain more privileges, defacing a web-page and generally performing actions that deviate from the way a daemon usually behaves. We argue that this divergence is reflected on the overall behaviour of the daemon and therefore an IDS should be able to identify the few diverging actions of an attacker against the daemon’s normal usage profile.

So, in this paper we present FIDS, a multiprocessing, python-based, file system IDS, that is able to identify attacks by reading BSM audit records both on-line and off-line (from /dev/auditpipe and BSM binary files, respectively), generating feature vectors from monitoring file system activity and, ultimately, by employing alternative machine learning techniques on the generated feature vectors. FIDS utilizes file system data because: (a) the file system stores most attackers’ traces on all OS’s, and (b) this storage is permanent (contrary to the system’s memory, or CPU registers, for example). The best and easiest source for monitoring file system activity on our FreeBSD1 servers that have been used for our experiments, was FreeBSD’s Audit System that generates BSM records.2 Except from FreeBSD, BSM is also available for Solaris and Mac OS X, hence our proposed IDS can run on those systems as well. Furthermore, MS Windows Advanced Auditing and Linux audit are mechanisms that generate audit records analogous to BSM, so FIDS can be ported to support those OS’s too. It is important to stress at this point that FIDS can be used in parallel with other types of IDS (HIDS, NIDS etc.) that are capable of identifying attacks not reflected on the file system. FIDS practically adds an additional level of awareness regarding the usage of the file system from the running daemons.

In the context of this paper FIDS reads audit records generated from the httpd daemon user (www), builds a normal usage profile and then monitors incoming activity for divergence through a moving window mechanism. We have employed both one-class Support Vector Machines and Gaussian Mixture Models (see Section 4) for novelty detection and have carried out experiments on datasets originating from three real-world web servers. The malicious activity dataset is inferred from commands gathered from a honeynet we have deployed just for this purpose. Our experimental results indicate that FIDS achieves high detections rates with low corresponding false positive rates and when compared to FWRAP [34], FIDS outperforms it.

1.3. Related work

In her seminal paper [4], Denning expresses the idea that intrusions against computers and networks may be detected if we assume that computer and/or network usage activity can be automatically profiled, and that the trails of intrusions are present in this activity. Therefore, if a security-bienn usage profile can be created for a monitored system, then all subsequent profiles created by the system’s later activity can be compared with this baseline using some meaningful metric and if great divergence is found, the system may fire an alarm to inform about the incident. Denning was the first to talk about how anomaly detection could be used for computer systems and specifically for host-based intrusion detection. Her initial idea came to life with the deployment of Multics Intrusion Detection and Alerting System (MIDAS)30, an expert system whose rule-base used Production Based Expert System Toolset(P-BEST) and was developed by the National Computer Security Center (NCSC).

A few years later, Forrest et al 5,6,9,39 demonstrated the notion of immune systems, inspired by her studies in natural immune systems. The key concept in her work was how to define the sense of self in the UNIX processes so as to detect intrusions by identifying abnormalities through processes’ deviations from self. Her representation of self in UNIX was through sequences of system calls. The approach of analysing system calls has been adopted by many other researchers [18,24,38,41], but as Wagner et al describe in [37], HIDS based on sequences of system calls seem to be susceptible to mimicry attacks.

Even though a lot of research has been held with the subject of intrusion detection, only a few papers have been involved with anomaly based IDS that identify attacks based on file system data and none of them seems to use the BSM audit mechanism to collect them, as we do. Some of the HIDS that used MIT’s Lincoln Laboratories’ DARPA’s ‘98 and ‘99 BSM datasets [20], have trained SVM’s [2,42] or other one class classifiers [12] for their detection mechanisms but, contrary to our proposed IDS, they have not used file system semantics on their features. Lastly, the legacy, discontinued EMERALD’s [27] eXpert-BSM [19] monitor implements an expert system IDS that uses BSM data as its input. But due to the fact that expert-BSM is a misuse IDS, instead of being trained with benign data to form a norm and try to identify attacks on newly arriving data by inspecting how related they are with respect to the norm, it uses a malicious rule base in its expert system that probes newly arriving data against it.

One more interesting work, although more relevant to permanent data storage, is the one proposed by Stanton et al. [33]. Even though they follow a very different modeling approach and use different types of data (block level data combined with file system data), their idea illustrates some features that are worth mentioning. They propose a 3-tiered anomaly based IDS, File and Block Surveillance System (FABS), that monitors file system and device controller data for abnormal behaviour. FABS builds a normal usage profile by studying sequences of events (disk accesses at the file and block level) and classifies as malicious all events that deviate from it. It uses C-miner’s [17] rule based engine to build their IDS and proposes a GUI prototype (VisFlowConnect-SS) for visualisation.

Another interesting approach that uses file system information is that of Stolfo et al. [34], where the proposed system, File Wrapper Anomaly Detector (FWRAP), uses features extracted from the file system of a Linux system, through the use of a kernel module the authors wrote, in order to build a normal usage profile; FWRAP uses the information extracted from this module to detect attacks based on a Bayesian estimation technique. Although our general idea uses the same source of information as Stolfo’s – the file system – our technique differs significantly. First of all, the feature set and the algorithms used are entirely different. Secondly, Stolfo et al. introduce a command entry in each feature vector which

---

renders the problem more related to command−histories IDS’s [31]. Finally, the experiments focus on totally different aspects than ours: their approach is to build a normal usage profile by monitoring root’s and other non−privileged human accounts’ activity on a Linux workstation and then identify attacks spawned by one of these human users, whereas our approach is to build a normal usage profile by auditing a daemon user’s file system activity (www in our case) and try to identify actions that a human user performs masqueraded as this daemon. Despite $F^2$DS and FWRAP differences, a comparison between $F^2$DS and FWRAP is meaningful, since FWRAP is currently the state−of−the−art IDS that is, at least, partially based on file system information directly extracted from the OS kernel. We therefore implemented FWRAP and used it as a baseline in the experiments presented in this paper.

Summarizing, our proposed IDS’ novelties with respect to the state of the art are: (a) the feature space we have created to map file system activity, (b) the daemon−user−centric angle we look at the IDS problem from, and (c) the source of our file−system dataset which uses the BSM mechanism.

1.4. Paper outline

The rest of the paper is organized as follows: Section 1.3 discusses related work and the state of the art, with respect to file system IDS’s. Section 2 presents our IDS architecture and the way it operates. Section 3 describes the features we have selected to map file system activity on, while Section 4 discusses the machine learning techniques we have used for building our detection engines in our experiments. These experiments as well as their results along with comparisons with the FWRAP and its algorithm, PAD, are presented in Section 5, followed by a brief discussion of our results (Section 6) and a conclusion with future work ideas in Section 7.

2. Proposed IDS architecture

Our IDS comprises several components (shown in Fig. 1) and works in three modes: training, ids and postmortem. In short, BSM records are collected, parsed and preprocessed and are then used to generate a set of feature vectors. These are in turn used to build an anomaly detection model during training mode, or to assess the probability of attack during ids or postmortem mode.

The various components/modules of this system as well as its modes of operation are outlined in the following paragraphs. As will be shown in subsequent sections, the focus of this work is on adequately expressing the web server’s file system activity, when it is observed from BSM audit records. Appropriate selection of features to be extracted allows development of highly effective anomaly detection models with high detection and low false alarm rates.

2.1. BSM audit records

BSM records are used as input to our system. Sun’s Basic Security Module (BSM) [32] is a mechanism that allows for fine grained auditing and is available for different OS’s (Solaris, FreeBSD, Mac OS, etc.). The implementation we have used in this paper is of OpenBSM, part of the TrustedBSD project, since the servers we have collected our data from were running the FreeBSD OS.

BSM audit records are generated by different OS facilities (kernel and user−land) and are handled by the auditd daemon. A sample audit record printed by the FreeBSD’s praudit utility is depicted in Table 1. Each record comprises a number of audit tokens and each audit token describes different system’s aspects. The audit record

<table>
<thead>
<tr>
<th>Table 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSM audit record printed with praudit.</td>
</tr>
<tr>
<td>header,131,11,access(2),0,Pri Jan 18 12:07:54 2013, + 393ms path,/lib/libjail.so.1 attribute,444,root,wheel,91,730199,2923288 argument,2,0x0,mode subject,www,root,root,root,wheel,67997,0,0,0,0,0,0 return,success,0 trailer,131</td>
</tr>
</tbody>
</table>

of Table 1 shows a different token in each line; the header, path, attribute and subject tokens are the ones that have been used in our feature generation process. An extended explanation of the BSM mechanism, its records and tokens can be found in [32].

The filtered record is parsed by the feature generation component ($fgc$) to form a feature vector; however, this procedure is explained in the following paragraphs.

From the header token we use the event attribute that shows us the system call that has been used when the file in question was accessed (access(2)). Hence, this part of the header token gives us the mode of access to the specific file. The path token contains only one attribute, the path (in our example /etc/libjail.so.1), which is the basis of forming a feature vector. From the subject token we are consulting the effective UID of the user in order to acquire only those records that are related to the specific daemon user (www in our example).

2.2. IDS components and running modes

The proposed IDS filters out BSM audit data unrelated to file system activity, creates feature vectors and generates an innocuous usage profile from standard usage data in order to identify attacks. This procedure is depicted in Fig. 1 and is implemented by the following IDS components:

The audit sensor component (asc) either listens on /dev/auditpipe for incoming BSM audit records or reads audit records from a binary BSM file. This module is used for data gathering and its output is passed right to the input of the pre−processing component that follows.

The preprocessor component (ppc) collects the binary audit records, filters them based on the system’s configuration and parses them in order to produce meaningful data objects to be used by the feature generation component. Filters are boolean expressions that are applied on each audit record and on each of its tokens to determine whether the record/token will be discarded or retained for further processing. These filters can range from very simple ones, like “is there a path token in the record’s path”, to more sophisticated, like “the path should contain the substring /etc and the user should be www or the time should be less than now” and provide a simple, yet very powerful, fine grained control mechanism as to which audit records and tokens will be processed by the feature generation mechanism that follows. When $F^2$DS is running in training mode, ppc is responsible for generating a database from which the statistics are calculated during the feature generation procedure.

The feature generation component ($fgc$) is responsible for the creation of feature vectors from the data obtained by ppc. The path and the mode of access of the audit records are used for querying the database, and statistics are computed that are used for the formation of the different features. In our experimental set−up, once a feature vector is created it is passed to the machine learning component for further processing. A detailed analysis of the features used in the web server case, discussed in this paper, is presented in Section 3.

Finally, the decision making component (dmc) is used (a) for training the machine learning model when the IDS runs in training mode,
(b) for generating an attack probability for feature vectors when $F^2$IDS runs in postmortem or ids mode, and (c) for detecting attacks and taking some action upon intrusion identification when running in postmortem or ids mode. In our set-up, an attack probability is computed and assigned to each feature vector; this attack probability is updated the attack score of a moving window and an alarm is triggered when the mean attack score of the records contained in this window exceeds a certain threshold.

The various IDS components behave differently in each running mode. The three running modes – ids, training and postmortem – are analysed as follows:

Training mode: in this mode audit records are either read online – from the audpipe – or off-line – from a file – through asc, and are subsequently parsed and pre-processed by ppc. The same component is responsible for generating a statistics database to be used for feature generation (Fig. 1). Next, ppc’s output is passed to fjc where feature vectors are created and the feature vectors are further passed to dmc for training. Once the training procedure is completed, the trained model is saved and used as the baseline of attack identification.

Ids mode: in this mode audit records are read on-line from the audpipe, they are parsed and pre-processed by ppc and given to fjc for feature generation; fjc’s output is passed into dmc for generating an attack score and identify potential attacks.

Post-mortem mode: this mode is similar to on-line mode. The only difference is that in this case, audit records are read off-line from a binary file. The system can inform the analyst as to which audit records triggered an alarm.

3. Feature extraction

Each incoming BSM audit record is processed to extract a small number of features that capture information relevant to anomaly detection. We identify two types of features:

1. Frequency features: Relative frequency values that measure how frequently files and/or directories are accessed by the web daemon.
2. Binary features: Binary values that identify qualitative characteristics of file system directories (e.g. whether a parent directory contains configuration files or executables).

3.1. Frequency features

In order to generate frequency features, a large number of records (hundreds of thousands to millions) is initially collected from the monitored daemon so as to extract statistics on the access frequency of files and directories. Table 2 lists the values that are collected in the database, which are subsequently used to extract the frequency features indicated in Table 3. The use of probabilities $Pr(f)$ and $Pr(m|f)$ is intuitively justified: a record indicating access of a file $f$ that is not normally accessed by the web daemon (at least not with mode $m$) is used as evidence to detect unusual behaviour. However, it is common for system daemons to create multiple temporary files and directories that have not previously appeared in the database. Features $x_1$ and $x_2$ alone cannot discriminate this type of normal operation from an attack and need to be complemented with additional features to avoid false alarms. The probabilities $Pr(fp)$, $Pr(pp)$ and $Pr(mp)$ provide additional information on how common it is to access the parent folder $p$ of $f$, $f$ within $p$ and files in $p$ with mode $m$ respectively. Furthermore, $Pr(pp)$ conveys information on the frequency of the parent folder with respect to its own parent folder; this can be discriminative in cases of normal behaviour with low $Pr(f)$ and low $Pr(p)$.

3.2. Binary features

Apart from the frequency of file and directory access there are additional characteristics of the accessed files and directories which are important for anomaly detection. A write attempt at a folder that contains library files is not common behaviour for a web daemon user and should be distinguished from write attempts at a folder used for file uploads, even if the access statistics are similar in both cases.

Table 2

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>Records in the database</td>
</tr>
<tr>
<td>$N_f$</td>
<td>Times file $f$ has been accessed</td>
</tr>
<tr>
<td>$N_{mf}$</td>
<td>Times $f$ has been accessed with mode $m$</td>
</tr>
<tr>
<td>$N_p$</td>
<td>Times the parent directory $p$ of file $f$ has been accessed</td>
</tr>
<tr>
<td>$N_{mp}$</td>
<td>Accesses of files with parent $p$ and mode $m$</td>
</tr>
<tr>
<td>$N_{pp}$</td>
<td>Accesses of the parent of $f$’s parent directory</td>
</tr>
</tbody>
</table>

Table 3

<table>
<thead>
<tr>
<th>Feature</th>
<th>Probability</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$Pr(f)$</td>
<td>$N_f/N$</td>
</tr>
<tr>
<td>$x_2$</td>
<td>$Pr(fp)$</td>
<td>$N_{mf}/N_f$</td>
</tr>
<tr>
<td>$x_3$</td>
<td>$Pr(p)$</td>
<td>$N_p/N$</td>
</tr>
<tr>
<td>$x_4$</td>
<td>$Pr(pp)$</td>
<td>$N_{mp}/N_{pp}$</td>
</tr>
<tr>
<td>$x_5$</td>
<td>$Pr(m</td>
<td>f)$</td>
</tr>
<tr>
<td>$x_6$</td>
<td>$Pr(mp)$</td>
<td>$N_{mp}/N_p$</td>
</tr>
</tbody>
</table>
Thus, the frequency features of Section 3.1 are complemented with a set of binary features (i.e. with values in \{0, 1\}) that correspond to the truth value of assertions about the accessed file and its parent directory. These are summarized in Table 4.

Features \(x_7\)–\(x_{11}\) are determined based on a predefined list of directories provided by the system administrator. For example, the administrator may indicate that directory /etc contains configuration files. If an audit record detects file access on a file in /etc subdirectory then \(x_7\) is 1, otherwise it is 0. An exception to this pattern is \(x_{12}\) that has been used to emphasize previously unseen file access records.

### 4. Anomaly detection models

Anomaly detection in our IDS is based on unsupervised machine learning. A set of BSM records is collected during system operation and the extracted features are concatenated to form a set of 12-dimensional feature vectors \(x_i\) one for each record \(i\). This collection is used to train a model that allows us to decide whether the following expression holds:

\[
p(x|y = 0) < T
\]  

where \(x\) is the input BSM record feature vector, \(y = 1\) if the record corresponds to an attack and 0 otherwise and \(p(x|y = 0)\) is the probability density function (pdf) of normal operation in the feature space. The threshold \(T\) is used to determine whether \(x\) corresponds to unusual behaviour. Setting \(T\) to higher values leads to higher detection rate, while setting it to lower values leads to lower false positive rate. We have used two different approaches to solve problem [1], namely Gaussian Mixture Models and Support Vector Machines, to show that the detection results are not necessarily related to the algorithm of the detection engine.

#### 4.1. Anomaly detection with GMM

GMMs [23] estimate the target pdf as a mixture of multivariate normal pdfs, i.e.

\[
p(x|y = 0) = \sum_{i=1}^{n} a_i K_i \exp \left( -\frac{1}{2} (x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i) \right)
\]  

where \(K_i = (2\pi)^{-k/2} |\Sigma_i|^{-1/2}\) is a normalisation factor, \(n\) is the number of mixtures (distributions), \(a_i\) correspond to the mixture weights such that \(\sum_{i=1}^{n} a_i = 1\), while \(\mu_i\) and \(\Sigma_i\) are the mean value and covariance matrix of the \(i\)-th mixture respectively. We may allow \(\Sigma_i\) to be a diagonal or full matrix, depending on the desired expressiveness of the model and the number of parameters that we are able to estimate with the available sample. Parameter estimation is achieved via expectation-maximization (EM) in a set of sample records.

The proposed prototype IDS has been implemented in Python and the scikit-learn package [25] was used for the implementation of GMM. A typical web server quickly collects thousands or millions of BSM audit records, so we selected full covariance matrix estimation. Selection of the number of mixtures \(n\) and threshold \(T\) is discussed in Section 5.

#### 4.2. Anomaly detection with SVM

For SVMs we use Schölkopf’s one-class SVM algorithm [29]. In this case we do not directly estimate the pdf \(p(x|y = 0)\), but instead aim at producing a model \(f\) such that \(f \geq 0\) when (1) holds and \(f < 0\) otherwise. This approach therefore solves a simpler problem than the one of Section 4.1, that involves full estimation of the pdf. Function \(f\) is estimated by adapting the SVM binary classification algorithm to consider all available samples as members of one class and the origin as the only member of the opposite class.

One-class SVM uses

\[
f(x) = \text{sign}(\mathbf{w}(\mathbf{x}) - \rho) = \begin{cases} 1 & \text{if } \langle \mathbf{w}, \mathbf{x} \rangle - \rho > 0 \\text{otherwise} \end{cases}
\]  

where \(\mathbf{w}(\mathbf{x})\) is unknown, but its inner product is computed via a kernel

\[
K(x, y) = \phi(x)\phi(y)
\]  

that satisfies Mercer’s conditions [1]. We wish to compute \(\mathbf{w}\) and \(\rho\) such that the margin between the two classes is maximized, while the number of misclassification errors is minimized. After defining the optimization problem (details in [29]) the one-class SVM function becomes

\[
f(x) = \text{sign} \left( \sum_{i=1}^{N} a_i K(x_i, x) - \rho \right)
\]  

where \(x\) is the input feature vector, \(x_i\) are the \(N\) samples used for training, \(0 \leq a_i \leq 1/N\) are determined during optimisation and \(\rho\) satisfies

\[
\rho = \sum_{j} a_j K(x_j, x_i)
\]  

for all \(i\). The parameter \(\nu\) controls how “tight” the separating bound will be around the target distribution and is thus related to the threshold \(T\) of Eq. (1). In our experiments we used the Radial Basis Function kernel, i.e.

\[
K(x, y) = e^{-\gamma|x-y|^2}
\]

### 5. Experiments

The way we have created our training sets and test sets, as well as the methods we have employed to run our experiments along with their results are presented in the paragraphs that follow.

#### 5.1. Experiment setup

FFDS is written in Python and its design and modules are presented in Section 2. In order to read BSM binary data, we used the pybsm\(^4\) library. As mentioned earlier, all machine learning algorithms were written using scikit-learn.

For our experiments with SVMs we have used a fixed setting for the threshold \(T\) of Eq. (1) (i.e. fixed value of \(\nu=0.5\)). For GMM, we chose a full covariance matrix and selected 10 to be the number of mixtures. Furthermore, for numerical stability we have used log probabilities and selected the threshold \(T\) to be

\[
\log T = \text{mean} - 3 \times \text{std}
\]

where \text{mean} is the average GMM log-probability density score of the training set with which the GMM one-class classifier has been trained (i.e. \(\text{mean} = 1/N \sum_{i=1}^{N} \log p(x_i|y = 0)\)) and \text{std} is the corresponding standard deviation.

---

We conducted four types of experiments to validate both our IDS and feature set performance. They answer the following questions, in this order: (1) Are the selected features expressive? How does a complex classifier – like SVM – behave on different training set sizes and method of training-set selection? (2) If the proposed feature set is effective, are all the features necessary in order to distinguish the man from the daemon? (3) How complex is the IDS anomaly detection problem with the proposed features? How does a simpler classifier – like GMM – work on our feature space? (4) How would the proposed anomaly detection approach be applied in a real IDS and how effective would it be? The results were proven more than promising for both SVM and GMM classifiers. Discussion of the experiments performed, followed by an analysis of the results are illustrated in Sections 5.3–5.5 and 5.7.

As one may infer, not all of our experiments can use the same datasets. Experiment set 1 uses different training sets of various sizes to show how training set selection affects detection and false positive rates. From the results of experiment set 1 we choose a representative training set for each server and use it on all other three experiments. Evaluation sets are the same for all experiments, but change slightly in experiment set 5 to become compatible with the moving window mechanism we employed (Section 2.2) to make our IDS suitable for on-line use. More information on our datasets is found in the following paragraphs.

5.2. The dataset

As we have already mentioned, our datasets have been created from three real-life web servers. Vergina hosts 27 sites, www.ee hosts 4 sites and thmmy hosts 4 sites. Some of these sites have been created with CMS’s, some are comprised of static HTML pages and others have been written with custom PHP code. The servers are running 3 different versions of FreeBSD (6-STABLE, 8-STABLE, 9-STABLE) and are all using apache\(^5\) as their web server. We followed no specific rules during the data collection, since all servers have different visiting patterns. We just started collecting data and stopped when the audit file’s size was at least 1 GB large. Server thmmy reached this limit in almost a day, www.ee needed 3 and vergina 2.

The only publicly available datasets used in IDS evaluations that contain BSM audit records are DARPA’s ‘98 and ‘99 datasets from MIT’s Lincoln Laboratories.\(^6\) Moreover, the operation of $F^2 DS$ presumes that BSM audit records originate from the daemon-user, in our case www, to identify file system specific abnormal behaviour. Unfortunately, as illustrated in [20,21], the only attacks against the web server (apache) in these datasets are DoS attacks which are not related to file system activity. Furthermore, the web server – like most other services and system processes – on these datasets was running as user root, so there is no way of distinguishing between audit records coming from the web server and those that don’t. Thus, a new dataset had to be created for the proper evaluation of our IDS. This was achieved by monitoring three of our Department’s web servers and the dataset creation process is described in the remainder of this section. The dataset that was used in the experiments can be found on-line.\(^7\) It is important to note at this point that the fact that $F^2 DS$ focuses on attacks that are related to the file system does not pose any restriction on using additional types of IDS (HIDS, NIDS, etc.) to extend the range of attacks that can be detected.

The main problem we had to solve when creating our datasets was the generation of malicious activity. The best way to create malicious activity is to know exactly how all attackers think and act, and emulate their behaviour by running various exploits against the monitoring servers in order to first compromise the host and then gain escalated privileges. Yet, unfortunately, successful exploits are not easy to write or to be found in our days, since most OS’s are equipped with stack protections, heap protections, etc. Even if we intentionally created a vulnerability for apache and managed to write an exploit and shell-code for it, it would be difficult to run it on our monitoring servers, since each of them runs a different major version of FreeBSD which means that a candidate exploit would have to be able to run on all FreeBSD versions or we would have to write different versions of the exploit for each OS, thus increasing its already high complexity. Moreover, proper evaluation would require far more than one exploit, and writing more than one implies even higher complexity. Therefore, we decided to approach this problem from the point where the attacker has compromised the victim host and forth.

Attackers rarely stop activities on the compromised host right after successful intrusion. No matter how access was gained on the victim system (e.g., via and exploit with a shell-code, by taking advantage of some PHP programming error, via an SQL injection, etc.), attackers are expected to use the privileges of the daemon user towards meeting their objectives. Examples of objectives include gaining escalated privileges, installing a back-door, opening a covert channel, searching for additional information with respect to the victim’s host, using the host as a hop to issue subsequent attacks on other systems, installing spam engines, installing bitcoin generators, etc. Most – if not all – of these actions involve some interaction with the file system. Hence, if we identified when the daemon user performs such actions, we would be able to infer that an attack has taken place and fire an alarm. This is equivalent to firing an alarm when a burglar picks the lock or breaks the window, but rather once they step on the floor. Of course, this does not prevent $F^2 DS$ from identifying the moment of intrusion as long as it involves interaction with the file system. Effectively, what we decided to do in our experiments was to –at least– identify human actions over daemon’s actions by monitoring the daemon’s file system activity.

5.2.1. The honeynet

In order to understand how an attacker behaves and be able to emulate their behaviour we have chosen to deploy a honeynet. First, we created two distinct, fully interactive honeypots running FreeBSD, that had six users with easy to guess passwords which were configured to log each user’s input on an append–only file. Those systems were built inside FreeBSD jails, and were highly hardened in order to protect the system from further compromising that could lead to erasing the log file. Moreover, we set-up a web server on each honeypot, configured to serve a popular CMS system having default admin usernames and passwords that were monitored too. After one month of operation there was no successful attempt on any of our honeypots, so the next honeypot implementation we decided to deploy was kippo,\(^8\) because it allows multiple passwords per user and is –at least– semi-interactive.

Kippo emulates a system running the SSH protocol and saves all successful and unsuccessful connection attempts –along with the usernames and passwords used– in its database. Once an attacker guesses a password correctly and logs into the system, all subsequent input and output is stored in kippo’s database. Additionally, the user is granted super-user privileges inside the emulated system which means that when an attacker logs into a kippo honeypot they already possess all available user privileges.

\(^6\) http://www.ll.mit.edu/mission/communications/cyber/CSTCorpora/ideval/data/.
\(^8\) https://code.google.com/p/kippo/.
Table 5

<table>
<thead>
<tr>
<th>Dataset</th>
<th>nus</th>
<th>nuts</th>
<th>mts</th>
</tr>
</thead>
<tbody>
<tr>
<td>virginia</td>
<td>522,257</td>
<td>150,000</td>
<td>3133</td>
</tr>
<tr>
<td>www_e</td>
<td>1,399,474</td>
<td>150,000</td>
<td>2433</td>
</tr>
<tr>
<td>thmmy</td>
<td>3,340,123</td>
<td>150,000</td>
<td>1035</td>
</tr>
</tbody>
</table>

Our honeynet comprises six kippo semi-interactive nodes plus the two initial FreeBSD-jail full interactive nodes, that span three different subnets. Throughout a three month monitoring period, two of our kippo nodes were compromised, 27,160 connections have been established, 643 of which resulted in successful logins. On 21 of those sessions the attacker issued at least one command from which 130 different lines of input have been identified. From this input, a set of 22 unique Unix commands has been extracted forming 34 different commands containing specific paths. The main reason why the total amount of unique malicious commands is relatively small is because the attackers were already connected as root, so they had no motive in trying to investigate the system for further vulnerabilities (Local-to-Root, L2R) and/or try to circumvent its security perimeter. Hence, in order to enrich our malicious command set, we asked our sysadmins to login into one of our servers as user www and act as if they had compromised it, trying to gain root access. By inspecting their command histories we gathered 15 more commands to add to our list, so our total number of malicious commands became 49.

5.2.2. Training and evaluation sets

Due to different server configurations, not all malicious commands could be issued on all our servers, so in www_e we issued 46 of them, on thmmy we executed 47 of them and in virginia we were able to execute all 49 of them. To produce our malicious dataset on each server, we configured them to log BSM audit records that were generated when we ran the appropriate malicious commands-set as user www.

Except from the malicious dataset, which served as our malicious test set (mts), we also created (a) a training set (ts) with normal usage data activity, and (b) a normal usage test set (nuts) to test how our system behaves on newly arrived normal activity, so as to be able to train and validate our IDS. Of course, since our data had been collected from real-world servers, there was no certainty that any of the audited systems had not been compromised during or before the data-gathering procedure, which means that our IDS performs unsupervised anomaly detection. Our final datasets are shown in Table 5.

So, overall, a training set, a malicious test set and a normal usage test set have been generated for each server. Due to the large size of our initial normal usage datasets (nus), only a sufficient subset has been used as our ts in our experiments to reduce training and anomaly detection complexity. Table 5 shows that thmmy’s mts is considerably smaller than the others’. This is because thmmy’s server configuration is hardened, and non-root users (like www who we used in our experiments) are unable to execute more than half of the malicious commands (24/47). When www runs these commands the system responds with “permission denied” errors (21/24) or “no such file or directory” errors (3/24). Even though we initially thought to filter out the commands failing to execute, we decided to keep them and explore further how F2DS responds on hardened systems. These errors generate a standard set of BSM records on the server, regardless of the executed command. The inability to run these commands lowers the detection rate on the specific dataset in experiment sets 1–4 where the detection engine evaluates each record separately; but interestingly, as we see in experiment set 5, this inability does not affect the real detection rate of F2DS that uses a moving window, proving its robustness on such phenomena.

As far as the test sets are concerned, the malicious part consists of the feature vectors generated by the malicious dataset and the each innocuous test set was populated with 150,000 feature vectors from each nus’ tail, and was – of course – excluded from the training set. We have performed a number of experiments that used all nus as the innocuous test set (excluding training samples) for all servers and we achieved the exact same false positive rates, indicating that an innocuous test set of this size is sufficient and indicative.

5.2.3. Training and evaluation sets for FWRAP and PAD

As discussed earlier, for comparison purposes we implemented the FWRAP IDS [34], which was the only alternative we found in the related literature that is closest to our approach. As described in [34], FWRAP utilizes seven features for intrusion detection:

1. UID, Which is the user ID running the process.
2. WD, Which is the working directory of a user running the process.
3. CMD which is the command line invoking the running process.
4. DIR which is the parent directory of the touched file.
5. FILE, which is the name of the file being accessed.
6. PRE-FiLE, which is the concatenation of the three previous accessed files.
7. FREQUENCY. Which encodes the access frequency of files in the training records. This value is estimated from the training data and discretized into four categories:
   (a) NEVER (for processes that do not touch any file).
   (b) FEW (where a file had been accessed only once or twice).
   (c) SOME (where a file had been accessed about 3–10 times).
   (d) OFTEN (more than SOME).

In order to run FWRAP with our data, all datasets needed to be transformed to fit its algorithm, PAD [35]. From these features, WD and CMD had to be excluded during our transformation procedure because they did not exist in our dataset. Nevertheless, due to the nature of our daemon-related analysis, none of them would contribute during the intrusion detection process anyway, since their values would not vary. That is because: (a) the working directory (WD) of the user running the process would always be equal to the working directory of the apache process, and (b) the command line invoking the running process (CMD) would almost always be the path of the php command with the exception of the few times that command httpd itself would run.

Moreover, as shown later on experiment set 3, we ran a comparison of PAD with our feature set and its original feature set (described above). The problem we had to face was that some of our features (frequency features in Table 3) are not categorical, whereas PAD uses categorical features. Therefore, for each such feature on each dataset, we found its maximum and minimum value and we divided the space between these two values into 20 labeled intervals of equal length. Then, in order to create the PAD-compatible dataset, all values of the specific feature were parsed and a new value was assigned to them that was equal to the label of the interval they belonged to.

5.3. Experiment set 1: model effectiveness and training set size

Experiment set 1 evaluates the effectiveness of the proposed system using one class SVM as the anomaly detection mechanism for different training set sizes and method of training set feature vector selection. We selected the training set with two ways: (a) contiguous feature vectors, (b) randomly selected feature vectors. First, as we explained in the previous paragraph, for each server we selected 150,000 features from nus to form our nuts. Then, for
a range of training set sizes and for each method of collection, we
trained a one-class SVM and evaluated its false positive rate (fpr)
based on the common nuts, and its detection rate (dr) based on mts.
During the evaluation process each record was assessed separately
and an alarm was triggered if it deviated from the norm. The train
size ranged from 10,000 to 140,000 feature vectors, and for each
train size more than one experiments took place. Figs. 2 and 3 show
the minimum dr and the mean and minimum fpr for all datasets for

![Fig. 2. Average fprs and drs on all datasets when selecting feature vectors contiguously.](image)

![Fig. 3. Average fprs and drs on all datasets when selecting feature vectors randomly.](image)

![Fig. 4. Average, minimum, maximum, standard deviation and cl2opt values for thmmy when selecting feature vectors contiguously.](image)
different train sizes and for continguously and randomly selected training sets respectively. Fig. 4 is a more analytic view of thmmy that depicts mean, max, min, std, cl2opt for fprs and drs for continguously selected training sets. Metric cl2opt (close to optimal) for fprs is equal to the percentage of fprs that are less or equal to 1.2 × \( \min(fpr) \) for the specific train size, while for the drs it is equal to the percentage of the drs that are greater or equal to 0.8 × \( \max(dr) \) for the specific train size. For each training set size (tss), a different number of experiments (k) have taken place. k depends on tss and nus' length. k is computed as follows: For our contiguous experiments and for a specific value of tss, we start from the beginning of nus and select tss number of records. For the next experiment of this size, we start at an offset of step(tss) and choose tss records again. For the n-th experiment of this size, we start at offset \( n \times \text{step(tss)} \) and choose tss records, with \( \max(n) = \text{div(len(nus), step(tss))} \), and it depends on nus' length and the specific tss. \( \text{step(tss)} \) is calculated as follows:

\[
\text{step(tss)} = \begin{cases} 
\frac{tss}{2} & \text{if tss} = 10,000 \\
\frac{tss}{2} & \text{if tss} > 10,000 
\end{cases}
\]

So, \( k = \max(n) \) is the total number of experiments we ran for each different tss size, for both continguously and randomly selected training sets.

As we can observe from Figs. 2 and 3, the behaviour of the detection engine is quite similar for both methods of selection for the same tss, with a slight exception of thmmy's minimum and mean fprs in the 100,000 and 110,000 region; in this region, both rates are suddenly rising a bit. So, for vergina and www ee on all train sizes the mean fpr is close to 10% and the minimum fpr is less than 10%; for thmmy, the corresponding rates are 20% and 10% respectively. But as we can see from Fig. 4 more specifically, fpr cl2opt is quite high for many different values of tss, which means that it is quite possible to score fprs close to its minimum for those sizes; a manifestation of this fact is shown in the following paragraphs, that describe the selection of the baseline training set for each dataset used in the subsequent experiments. Moreover, with the exception of thmmy whose mean tprs are a bit lower than 90%, the other two server's mean tprs are above 95%, for both methods of sample selection. Hence, despite thmmy's small divergence, we can certainly conclude that our model is definitely descriptive, as far as intrusion detection is concerned, for many different values of tss and for both methods of feature selection.

Throughout the rest of our experiments, we have selected one specific training set for each server that performs well in order to form a baseline and be able to compare and discuss further results against it. As we see in Figs. 2 and 3 this does not affect generality, since good results can be achieved for almost all different sizes and method of selection on each server. What is interesting is that dr minimums (Fig. 5) are very close to dr means, which means that when someone trains their IDS, they do not need to worry about drs, since they never fall lower than the minimum which is already high enough. On the other hand, what they should be worried about is achieving an acceptable low fpr and the desired dr is in essence guaranteed. Hence, the way we selected our baseline training sets can be used as the default method of choosing the appropriate training set for any server wishing to use \( \partial^2 \text{DSS} \). We start by choosing a test set larger than the maximum desired tss (e.g. 150,000) and train an SVM one-class classifier using training sets of increasing sizes that start from the first element of nus; the training set size increases with a step of 10,000 feature vectors on each run. We stop our runs when a low false positive rate (<10%) has been achieved. As one may infer from Table 6, www ee's training set was selected from the first run, thmmy's from the third and vergina's from the fourth. Results (Table 6) show detection rates higher than 95% for all datasets and corresponding false positive rates that do not exceed 8.15%. One class SVM was trained with a gamma value of 0.1, an RBF kernel and 0.001 tolerance.

In order to rule out the possibility that only one feature, or a small combination of our features is highly informative by itself and is the primary reason why the machine learning novelty detector performed that well, Experiment set 2 was carried out.

5.4. Experiment set 2: feature effectiveness

Given the fact that the dimensionality of our feature space is relatively low, instead of using a sub-optimal feature selection method to validate our features, we simply decided to run experiments for all \( 2^{12} - 1 \) possible combinations of features, using the SVM one class classifier and the baseline training sets of Experiment set 1.

Fig. 6 shows a histogram of how frequently a specific feature is found in the set of feature combinations that scored the highest drs while scoring the lowest fprs at the same time. For vergina and www ee feature combinations that scored drs higher than 90% and fprs lower than 10% have been selected, while for thmmy 1000 feature combinations have been selected that scored the highest drs while having corresponding fprs lower than 10%. The intersection of those three sets that contain the combinations of features scoring the best for each server, denoted C, is comprised of 176 elements, from a total of 4095. In Fig. 6, integers on the x axis represent a different feature, while the y axis depicts the percentage of the elements of C that contain the specific feature.

From Fig. 6 we can see that all features are participating in the highest scoring combinations of features, at least to an extent. x axis represents the feature number, using the numbering of Tables 3 and 4. The feature that is found in the fewest elements (22.1%) of C is \( x_5 (Pr(mff)) \) and the one that is found in all elements is \( x_7 \) (parent contains configuration files). The rest of the features are found in at least 40% of C's elements. The lowest percentage is sufficiently close to the rest, thus one may argue that all of the selected features participate in the combinations that score

### Table 6

<table>
<thead>
<tr>
<th>Dataset</th>
<th>dr</th>
<th>fpr</th>
<th>tss</th>
</tr>
</thead>
<tbody>
<tr>
<td>vergina</td>
<td>98.21%</td>
<td>8.39%</td>
<td>40,000</td>
</tr>
<tr>
<td>www_e_e</td>
<td>95.84%</td>
<td>6.07%</td>
<td>10,000</td>
</tr>
<tr>
<td>thmmy</td>
<td>88.11%</td>
<td>6.69%</td>
<td>30,000</td>
</tr>
</tbody>
</table>
the highest results, at least to an extent, and therefore all are needed in order to form a good classifier.

Fig. 7, on the other hand, shows the mean detection rate and false positive rate for the elements of C that contain 1–12 features. One can clearly identify that the fpr is decreasing monotonically as more features are present in the feature vector while the dr is initially decreasing but after at least four features are present in the feature vector it starts rising again. We see a small divergence for thmmy, where dr starts falling a bit after 10 features are present, but as we have explained before, due to its hardened configuration, almost half of the malicious commands return an error which is confusing the detection engine, so we cannot expect thmmy’s behaviour to always agree with the norm as we cannot always explain the norm from thmmy’s point of view. On the other hand, the difference in behaviour of fpr to dr can be easily explained: When the feature space is comprised of fewer features (less than 3 in Fig. 7), the detection mechanism cannot be trained well and fpr is very high while dr is very high as well; this means that most monitored activity, malicious or not, is recognised as attack. When more features are added to the feature space, the IDS starts learning benign data more accurately, and so the dr initially decreases, but then rises again while the fpr keeps decreasing until all features are present and the feature space is more informative.

5.5. Experiment set 3: feature space validation

Results from Experiment set 1 indicate that the model implemented can perform well on all datasets. Results from Experiment set 2 depict that our feature selection process does not lead to superfluous features. What remains to be answered is whether (a) the selected features are mapping the input to a highly discriminative space – thus creating a relatively easy anomaly detection problem which more than one machine learning algorithms are able to solve – or if the machine learning problem is hard, yet SVM managed to solve it, and (b) if this feature space performs better than alternatives found in the related literature.

To gain insight regarding question (a), a second novelty detection algorithm was chosen, namely GMM with the configuration explained in Section 5.1. We have used the baseline training sets for each server as explained in Experiment set 1.

We ran GMM for all our datasets, and as we can see from Table 7, the results clearly validate the effectiveness of the proposed IDS. Fprs are significantly lower than the corresponding ones on SVM (Table 6) and for some datasets (thmmy, www_ee) the fp rate is close to or even lower than 1%; the lower fprs of GMM compared to those of SVM should not necessarily surprise us, as stated in [22]. Moreover, with the exception of the thmmy dataset, drs are very close to those of Table 6. It should be mentioned that by raising the log-probability density threshold, one can raise the detection rate with a trade-off in false positive rate and vice versa, but we have chosen to pick the log-probability threshold without borrowing knowledge from mts in order to keep our IDS unsupervised.

Furthermore, to answer the comparison question (b) we implemented the algorithm used by FWRAP, PAD [35], and ran it with both feature sets – ours and FWRAP’s – on all three servers, using the same training sets and evaluation sets that have been described earlier. A detailed explanation of how the existing datasets have been transformed to match PAD’s needs has been presented in Section 5.2.3. With the results of these experiments, the ROC curves of Figs. 8 and 9 have been calculated, that belong to PAD running with our feature set and FWRAP’s respectively. As a comparison metrics we have used the computed areas under each ROC curve, which as Figs. 8 and 9 depict, PAD’s area running with our feature set is greater than PAD’s area running with FWRAP’s feature set for all three datasets, indicating that our feature set makes PAD perform better than FWRAP’s feature set does.

What is equally interesting is that these results, apart from showing that our feature set is more descriptive with respect to intrusion detection than FWRAP’s, implicitly strengthen the results referring to question (a) as well. And that is because PAD is a very simple decision making algorithm that performs well when using our feature set, and therefore another simple one-class classifier using our feature set can perform well on all datasets, proving once more that our feature space is highly discriminative.

5.6. Experiment set 4: FI2DS vs FWRAP

Up until this point we have shown that FI2DS can be used as a file system IDS that scores high detection rates and low false positive

<table>
<thead>
<tr>
<th>Dataset</th>
<th>dr</th>
<th>fpr</th>
<th>tss</th>
</tr>
</thead>
<tbody>
<tr>
<td>vergina</td>
<td>98.02</td>
<td>1.98</td>
<td>40,000</td>
</tr>
<tr>
<td>www_ee</td>
<td>95.85</td>
<td>0.65</td>
<td>10,000</td>
</tr>
<tr>
<td>thmmy</td>
<td>83.38</td>
<td>1.1</td>
<td>30,000</td>
</tr>
</tbody>
</table>
Table 8
Detection rates (dr), false positive rates (fpr) and training set sizes (tss) for FWRAP on all datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>dr</th>
<th>fpr</th>
<th>tss</th>
</tr>
</thead>
<tbody>
<tr>
<td>vergina</td>
<td>99.81%</td>
<td>20.12%</td>
<td>40,000</td>
</tr>
<tr>
<td>www_ee</td>
<td>98.71%</td>
<td>13.33%</td>
<td>10,000</td>
</tr>
<tr>
<td>thmmy</td>
<td>96.33%</td>
<td>14.34%</td>
<td>30,000</td>
</tr>
</tbody>
</table>

rates. Moreover, as we have explained in Section 5.2, there are no publicly available datasets that can be used with F2DS and therefore no direct comparison of F2DS against some other alternate IDS can be made through a score comparison on it. This comparison issue has been addressed by implementing the FWRAP IDS, training and evaluating it with all our datasets and calculating its detection rates, false positive rates (Table 8) and ROC curves (Fig. 9). Furthermore, we calculated the ROC curves for GMM (Fig. 10) on all servers with the same datasets in order to be able to directly compare F2DS with FWRAP.

As one can notice from Table 8, FWRAP achieves high detection rates, but also relatively high false positive rates, especially in comparison with the associated results of F2DS running with SVM (Table 6) and GMM (Table 7). There was no threshold we could find
Table 9

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Window size</th>
<th>Threshold</th>
<th>dr</th>
<th>fpr</th>
</tr>
</thead>
<tbody>
<tr>
<td>vergina</td>
<td>6</td>
<td>51–60%</td>
<td>95.92%</td>
<td>0.081%</td>
</tr>
<tr>
<td>www, ee</td>
<td>6</td>
<td>51–40%</td>
<td>95.74%</td>
<td>0.00093%</td>
</tr>
<tr>
<td>thmmy</td>
<td>6</td>
<td>51–60%</td>
<td>91.30%</td>
<td>0.02%</td>
</tr>
</tbody>
</table>

for FWRAP that would make the false positive rates fall any lower, and hence the peculiar endings of the related ROC curves (generated by scipy [11]). Moreover, from Figs. 9 and 10 it is apparent that FFD
dSO ROC area for each server is greater than the corresponding FWRAP area, which clearly indicates that FFD
dS0 performs better than FWRAP.

5.7. Experiment set 5: moving window evaluation

On all previous experiments, detection and false positive rates have been calculated for each record separately. This implies that if an attack is constituted of 800 records that 760 of them are labeled malicious, the aforementioned mechanism fires 760 different alarms. Since this behaviour is not practical at all for live systems, we added a moving window mechanism on FFD
dS0 that fires an alarm whenever the percentage of malicious records it contains exceeds a certain threshold. In this method, each record is still screened for its malicious intent, but the alarm-firing process is based on some aggregation. We have deliberately chosen to use the simplest aggregate metric (percentage of malicious records to total number of records in window) so as to see how the simplest moving window implementation would affect our detection mechanism.

As far as the datasets that have been used in this experiment is concerned, our ts have remained the same (baseline ts from Experiment set 1) but our mts and nts have changed to meet the experiment’s demands. We created an mts that is comprised of malicious and innocuous regions of audit records, where each malicious region contains a malicious command’s audit records and each innocuous region contains 1000 audit records collected from nus. nts, on the other hand, comprises the hole nus excluding the records used for ts. The moving window slides through these new mts and nts and once it recognizes an attack it fires an alarm.

In this experiment set, moving windows sizes (mws) ranging from 2 to 40 and various percentage thresholds (pt) ranging from 10% to 60% have been tested and a total of 48 different combinations have been found that on all datasets scored tprs >90% with corresponding fprs <10−3. From these 48 combinations, 10 scored the best results on all three datasets, and these are presented in Table 9. We used GMM as our detection mechanism using the configuration of Section 5.1 due to GMM’s lower fprs compared to SVM’s and PAD’s running with our feature set; fprS play a very significant role, especially in systems like ours where there are lots of feature vectors in our sensor’s input, thus forcing us to keep fprs as low as possible, without significant decline in corresponding drs.

As we see in Table 9, fprs have fallen 2–4 orders of magnitude, even though we did not use any false positive reduction technique (e.g. training a second classifier to learn false positives, etc.) maintaining high drs. This time, thmmy’s dataset scores significantly higher than it did on our previous examples, vindicating our decision to not exclude the commands that failed to execute from its mts.

6. Discussion

Throughout this series of experiments FFD
dS0 has been evaluated against various tests to be assessed as an anomaly detection IDS. In these experiments, we have initially shown that the feature set presented in this paper can be utilized successfully for intrusion detection when the detection engine is based on one-class SVM, yielding high detection rates and low false positive rates. Additionally, we have shown that all members of our feature vector are to some degree necessary during the intrusion detection process, so none of them could be left out. Furthermore, we have argued that SVM can be replaced with less complex algorithms, like GMM and PAD, without deteriorating the detection performance of FFD
dS0, indicating that our feature space is highly discriminative with respect to intrusion detection.

For comparison purposes, FFD
dS0 has been evaluated against FWRAP, an alternate file system based IDS of the relevant literature that introduces a different set of features (see Section 5.2.3) and a different decision engine [35]. The two IDS’s have been compared based on detection rates, false positive rates and ROC areas, where FFD
dS0 outperformed FWRAP in all comparisons and the comparison results will be validated for statistical significance. Furthermore, a ROC area comparison has been held between PAD running with our feature set and PAD running with FWRAP’s feature set to directly compare the two feature sets and the results of this comparison were in favour of our approach.

To further strengthen our comparison findings and test if our results are statistically significant, we performed a t-test (hypothesis test) that was based on the ROC areas that were calculated during 1000 runs of FFD
dS0 running with GMM, FWRAP and FFD
dS0 running with PAD on each server using the same train sizes and evaluation sizes as already mentioned. On each run, a new randomly selected contiguous training set and a new randomly selected evaluation set of appropriate size was created, and these sets were used by all three IDS’s to calculate the corresponding ROC areas.

By t-testing the results of PAD and FWRAP using the following null and alternate hypothesis:

\[
H_0 : \mu_{PAD} = \mu_{FWRAP} \\
H_1 : \mu_{PAD} > \mu_{FWRAP}
\]

we received the results depicted in Table 10. As one can deduce with high certainty (level of significance \(0 < p < 3.47 \times 10^{-194}\)), the expected ROC area of FFD
dS0 running with PAD is greater than the respective expected ROC area of FWRAP, meaning that our feature set is more descriptive with respect to intrusion detection than the feature set used by FWRAP.

Moreover, by t-testing the results of FFD
dS0 and FWRAP using the following null and alternate hypothesis and evaluating the results of Table 11:

\[
H_0 : \mu_{FFD} = \mu_{FWRAP} \\
H_1 : \mu_{FFD} > \mu_{FWRAP}
\]

we can conclude with high confidence \((1.37 \times 10^{-6}) < p < 1.46 \times 10^{-81}\) that FFD
dS0 outperforms FWRAP on all datasets, since the

Table 10

<table>
<thead>
<tr>
<th>Dataset</th>
<th>t-statistic</th>
<th>p-value</th>
<th>tss</th>
</tr>
</thead>
<tbody>
<tr>
<td>vergina</td>
<td>81.20</td>
<td>0.0</td>
<td>40,000</td>
</tr>
<tr>
<td>www, ee</td>
<td>37.66</td>
<td>3.47 × 10^{-194}</td>
<td>10,000</td>
</tr>
<tr>
<td>thmmy</td>
<td>35.60</td>
<td>3.29 × 10^{-100}</td>
<td>30,000</td>
</tr>
</tbody>
</table>

Table 11

<table>
<thead>
<tr>
<th>Dataset</th>
<th>t-statistic</th>
<th>p-value</th>
<th>tss</th>
</tr>
</thead>
<tbody>
<tr>
<td>vergina</td>
<td>51.15</td>
<td>1.46 × 10^{-281}</td>
<td>40,000</td>
</tr>
<tr>
<td>www, ee</td>
<td>46.56</td>
<td>7.99 × 10^{-231}</td>
<td>10,000</td>
</tr>
<tr>
<td>thmmy</td>
<td>18.06</td>
<td>1.37 × 10^{-623}</td>
<td>30,000</td>
</tr>
</tbody>
</table>
expected ROC area of $F^2DS$ is highly likely larger than the expected ROC area of $FWRAP$.

Finally, a real life implementation of $F^2DS$ was presented that uses a sliding window mechanism that achieves high detection rates with low corresponding false positive rates in all our servers, that in the best case (see Table 9) reached a 95.74% detection rate with an associated 0.00093% false positive rate.

7. Conclusion and further work

We have presented $F^2DS$, a File system Intrusion Detection System that performs anomaly detection based on file system BSM audit records. An audit sensor gathers incoming audit records and a pre-processing step filters them to obtain those that are relevant to the monitored daemon. A feature extraction step computes a set of features from each audit record and a machine learning anomaly detection model determines whether each record corresponds to normal behaviour or an alarm should be raised.

Two categories of features are extracted, frequency and binary. Frequency features allow us to compute the probability of access of files and directories with specific modes based on previously collected audit data. Binary features define qualitative characteristics of the accessed files and directories, such as whether they correspond to executable files, configuration files, library files etc. For anomaly detection, two different unsupervised approaches are employed in $F^2DS$, namely one-class SVMs with RBF kernel and full covariance matrix GMMs.

To experimentally validate our approach we generated three datasets from high traffic real-world web servers, hosting a total of 35 web sites. The results of our experiments have been compared with those of an alternate IDS ($FWRAP$) and have been proven to be better.

$F^2DS$ uses a moving window to assess audit records and react on an attack. Results of our experiments show that the proposed approach is highly effective for intrusion detection on web servers, reaching high detection rates and low false positives rates. Furthermore, good results were obtained with different schemes, dictating that the proposed features are highly discriminative for $F^2DS$, so that even relatively simple machine learning algorithms perform well. At the same time, analysis with all possible feature subsets shows that all of the proposed features are useful, so feature redundancy is negligible.

$F^2DS$ is the first IDS to use only the path attribute of BSM audit records for anomaly detection and the first one to introduce the specific feature space. Experimental results show that our approach is highly promising and encourage further research work in this direction. Examples include evaluation of $F^2DS$ for other daemons beyond Web servers, evaluation on data collected from real attack environments (War Games), false positive analysis and reduction, cluster analysis of the proposed feature space and application of more sophisticated moving window algorithms on the decision engine.

Acknowledgments

We would like to thank the Department of Electrical and Computer of the Aristotle’s University of Thessaloniki, Greece and their sys-admin team for allowing us to collect data from their servers and Miltos Allamanis for his directions on how to use GMMs for our novelty detection problem.

References

[33] P. Stanton, W. Yurick, L. Brumbaugh, Fabs: file and block surveillance system for determining anomalous disk accesses, in: Information Assurance Workshop,


