Data Mining for Intrusion Detection: from Outliers to True Intrusions

Goverdhan Singh[1], Florent Masseglia[1], Cline Fiot[1], Alice Marascu[1], and Pascal Poncelet[2]

[1] INRIA Sophia Antipolis, 2004 route des lucioles - BP 93, FR-06902 Sophia Antipolis, France
Email: First.Last@sophia.inria.fr
[2] LIRMM UMR CNRS 5506,161 Rue Ada, 34392 Montpellier Cedex 5, France
Email: poncelet@lirmm.fr

Abstract. Data mining for intrusion detection can be divided into several sub-topics, among which unsupervised clustering has controversial properties. Unsupervised clustering for intrusion detection aims to i) group behaviors together depending on their similarity and ii) detect groups containing only one (or very few) behaviour. Such isolated behaviours are then considered as deviating from a model of normality and are therefore considered as malicious. Obviously, all atypical behaviours are not attacks or intrusion attempts. Hence, this is the limits of unsupervised clustering for intrusion detection. In this paper, we consider to add a new feature to such isolated behaviours before they can be considered as malicious. This feature is based on their possible repetition from one information system to another.

1 Introduction

Intrusion detection is a very important topic of network security that has received much attention [5, 9, 4, 7] since potential cyber threats are making the organizations vulnerable. *Intrusion Detection Systems (IDS)* are intended to protect information systems against intrusions and attacks and are traditionally based on signatures of known attacks [8, 1]. Therefore, new kinds of attacks regularly have to be added to the signature list. The main drawback is that in case of an emerging attack, based on the recent discovery of a new security hole for instance, the IDS will ignore it since this new attack has not yet been listed in the base of signatures.

Protecting a system against new attacks, while keeping an automatic and adaptive framework is an important topic in this domain. One answer to that problem could rely on data mining. Data mining tools have been used to provide IDS with more adaptive detection of cyber threats [2, 10]. Among those data mining approaches, anomaly detection tries to deduce intrusions from atypical records [4, 3]. The overall principle is generally to build clusters, or classes, of usage and find outliers (*i.e.* events that do not belong to any class or group identifying normal usage). However, the main drawback of detecting intrusions

by means of anomaly (outliers) detection is the high rate of false alarms since an alarm can be triggered because of a new kind of usages that has never been seen before (and is thus considered as abnormal). Considering the large amount of new usage patterns emerging in the Information Systems, even a weak percent of false positive will give a very large amount of spurious alarms that would be overwhelming for the analyst. Therefore, the goal of this paper is to propose an intrusion detection algorithm that is based on the analysis of usage data coming from multiple partners in order to reduce the number of false alarms. Our main idea is that a new usage is likely to be related to the context of the information system on which it occurs (so it should only occur on this system). On the other hand, when a new security hole has been found on a system, the hackers will want to use it in as many information systems as possible. Thus a new anomaly that occurs on two (or more) information systems is probably not a new kind of usage, but rather an intrusion attempt. Let us consider A_x, an anomaly detected in the usage of web site S_1 corresponding to a php request on the staff directory for a new employee: John Doe, who works in room 204, floor 2, in the R&D department. The request will have the following form: `staff.php?FName=John\&LName=Doe\&room=204\&floor=2\&Dpt=RD`. This new request, due to the recent recruitment of John Due in this department, should not be considered as an attack. On the other hand, let us consider A_y, an anomaly that corresponds to a true intrusion. A_y will be based on a security hole of the system (for instance a php vulnerability) and might, for instance, look like: `staff.php?path=../etc/passwd%00`. One can see in this request that the parameters are not related to the data accessed by the php script, but rather to a security hole that has been discovered on the *staff* script. If two or more firms use the same script (say, a directory resquesting script bought to the same software company) then the usage of this security hole will certainly be repeated from one system to another and the request having parameter `../etc/passwd%00` will be the same for all the victims. In this paper, we propose to provide the end-user with a method that takes only one parameter: n, the number of desired alarms. Then, based on the analysis of the usage data coming from the different partners, our algorithm will detect n common outliers they share. Such common outliers are likely to be true attacks and will trigger an alarm.

The paper is organized as follows. In Section 2 we present the motivation of this approach and our general framework. Section 3 presents Cod, our method for detecting outliers and triggering true alarms. Eventually, our method is tested through a set of experiments in Section 4 and Section 5 gives the conclusion.

2 Motivation and General Principle

In this paper we present Cod (Common Outlier Detection) a framework and algorithm intended to detect the outliers shared by at least two partners in a collaborative IDS. Outliers are usually small clusters and our goal is to use outlier lists from different systems (based on a similar clustering, involving the same similarity measure). If an outlier occurs for at least two systems, then

it is considered as an attack. COD is indeed based on the assumption that an intrusion attempt trying to find a weakness of a script will look similar for all the victims of this attack. For clarity of presentation we present our framework on the collaboration of two Web sites, S_1 and S_2 and we consider the requests that have been received by the scripts of each site (cgi, php, sql, etc). Our goal is to perform a clustering on the usage patterns of each site and find the common outliers. However, that would not be enough to meet the second constraint of our objective: to require only one parameter, n, the number of alarms to return. Our similarity measure (presented in section 3.1) will allow normal usage patterns to be grouped together rather than grouped with intrusion patterns. On the other hand, our similarity measure also has to ensure distinguishing an intrusion pattern from normal usage patterns and from other intrusion patterns (since different intrusion patterns will be based on a different security hole and will have very different characteristics). Our algorithm performs successive clustering steps for each site. At each step we check the potentially matching outliers between both sites. The clustering algorithm is agglomerative and depends on the maximum dissimilarity (MD) that has to be respected between two objects.

This work is intended to explore the solutions for monitoring a network in real time. Then, the potential alarms will be triggered at each step of the monitoring (for instance with a frequency of one hour). Depending on the number of true or false alarms, the user might want to adjust n for the next step, until no (or very few) false alarm is returned. Our assumption is that common outliers, sorted by similarity from one site to another, will give the intrusions at the beginning of the list.

3 COD: Common Outlier Detection

The principle of COD is to perform successive clustering steps on usage patterns of different partners sites, until the number of common outliers meets the number of alarms desired by the user. We present in this section an algorithm designed for two information systems. Extending this work to more than two systems would require a central node coordinating the comparisons and triggering the alarms, or a peer-to-peer communication protocol. This is not the goal of this paper. Our objects are the parameters given to script files in the requests received on a Web site. In other words, the access log file is filtered and we only keep lines corresponding to requests with parameters to a script. For each such line, we separate the parameters and for each parameter we create an object. Let us consider, for instance, the following request: `staff.php?FName=John&LName=Doe`. The corresponding objects are o_1 =`John` and o_2 =`Doe`. Once the objects are obtained from the usage data of multiple Web sites, COD is applied and gives their common outliers.

3.1 Main Algorithm

As explained in section 2, COD algorithm will process the usage patterns of both sites step by step. For each step, a clustering result is provided and analyzed for

intrusion detection. First, MD is set to obtain very tight and numerous clusters (very short similarity is allowed between two objects in a cluster). Then, MD is relaxed by an amount of 0.05 step after step in order to increase the size of resulting clusters, decrease their number and lower the number of alarms. When the number of alarms desired by the user is reached, then Cod ends.

Algorithm Cod
Input: U_1 and U_2 the usage patterns of sites S_1 and S_2
and n the number of alarms.
Output: I the set of clusters corresponding
to malicious patterns.

1. Build M, the distance matrix between each pattern ;
2. $\forall p \in M, Neighbours_p \leftarrow$ sorted list of neighbours for p (the first usage pattern in the list of p is the closest to p).
3. $DensityList \leftarrow$ sorted list of patterns by density ;
4. $MD \leftarrow 0$;
5. $MD \leftarrow MD + 0.05$;
6. $C_1 \leftarrow Clustering(U_1, MD)$;
   $C_2 \leftarrow Clustering(U_2, MD)$;
7. $O_1 \leftarrow Outliers(C_1)$; $O_2 \leftarrow Outliers(C_2)$;
8. $I \leftarrow CommonOutliers(O_1, O_2, MD)$;
9. If $|I| \leq n$ then return I ;
10. If $MD = 1$ then return I ; // No common outlier
11. Else return to step 5 ;

End algorithm Cod

3.2 Clustering

Cod Clustering algorithm is based on an agglomerative principle. The goal is to increase the volume of clusters by adding candidate objects, until the Maximum Dissimilarity (MD) is broken (*i.e.* there is one object o_i in the cluster such that the similarity between o_i and the candidate object o_c is greater than MD).
Similarity between objects. We consider each object as a sequence of characters. Our similarity is then based on the longest common subsequence (LCS), as described in definition 1.

Definition 1. *Let s_1 and s_2 be two sequences. Let $LCS(s_1, s_2)$ be the length of the longest common subsequences between s_1 and s_2. The* dissimilarity $d(s_1, s_2)$ *between s_1 and s_2 is defined as follows:* $d(s_1, s_2) = 1 - \frac{2 \times LCS(s_1, s_2)}{|s_1| + |s_2|}$

Example 1. Let us consider two parameters p_1=`intrusion` and p_2=`induction`. The LCS between p_1 and p_2 is L=`inuion`. L has length 6 and the similarity between p_1 and p_2 is $d = 1 - \frac{2 \times L}{|p_1| + |p_2|} = 33.33\%$. Which also means a similarity of 66.66% between both parameters.

Centre of clusters. When an object is inserted into a cluster we maintain the centre of this cluster, since it will be used in the CommonOutliers algorithm. The centre of a cluster C is the LCS between all the objects in C. When object o_i is added to C, its center C_c is updated. The new value of C_c is the LCS between the current value of C_c and o_i.

Algorithm Clustering
Input: U, the usage patterns
and MD, the Maximum Dissimilarity.
Output: C, the set of as large clusters as possible,
respecting MD.

1. $i \leftarrow 0$; $C \leftarrow \emptyset$;
2. $p \leftarrow$ next unclassified pattern in $DensityList$;
3. $i++$; $c_i \leftarrow p$;
4. $C \leftarrow C + c_i$;
5. $q \leftarrow$ next unclassified pattern in $Neighbours_p$;
6. $\forall o \in c_i$ If $d(o, q) > MD$ then return to step 2 ;
7. add q to c_i ;
8. $C_c \leftarrow LCS(C_c, q)$; //C_c is the center of C
9. return to step 5 ;
10. If unclassified patterns remain then return to step 2 ;
11. return C ;

End algorithm Clustering

3.3 Detecting Common Outliers

Our outlier detection principle is described in [6]. Since we want our global algorithm to require only one parameter (the number of alarms), we want to avoid introducing a similarity degree for comparing two lists of outliers. For this comparison, our algorithm uses the centre of outliers. For each pair of outliers, it calculates the similarity between centers of these outliers. If this similarity is below the current MD, then we consider those outliers as similar and add them to the alarm list.

4 Experiments

The goal of this section is to analyze our results (*i.e.* the number of outliers and true intrusions and the kind of intrusions we have detected). Our datasets come from two different research organizations; (*anonymized for submission*). We have analyzed their Web access log files from March 1 to March 31. The first log file represents 1.8 Gb of rough data. In this file, the total number of objects (parameters given to scripts) is 30,454. The second log file represents

1.2 Gb of rough data and the total number of objects is 72,381. COD has been written in Java and C++ on a PC (2.33GHz i686) running Linux with 4Gb of main memory. Parameters that are automatically generated by the scripts have been removed from the datasets since they cannot correspond to attacks (for instance "`publications.php?Category=Books`"). This can be done by listing all the possible generation of parameters in the scripts of a Web site.

4.1 Detection of common outliers

As described in Section 2, COD proceeds by steps and slowly increases the value of MD, which stands for a tolerance value when grouping objects during the clustering process. In our experiments, MD has been increased by steps of 0.05 from 0.05 to 0.5. For each step, we report our measures in table 1. The meaning of each measure is as follows. O_1 (resp. O_2) is the number of outlying objects in site 1 (resp. site 2). $\%_1$ (resp $\%_2$) is the fraction of outlying objects on the number of objects in site 1 (resp. site 2). For instance, when MD is set to 0.3, for site 1 we have 5,607 outlying objects, which represents 18.4% of the total number of objects (*i.e.* 30,454) in site 1. COD is the number of common outliers between both sites and $\%_{FA}$ is the percentage of false alarms within the common outliers. For instance, when MD is set to 0.05, we find 101 alarms among which 5 are false (which represents 4.9%). One first observation is that outliers cannot be directly used to trigger alarms. Obviously, a number as high as 5,607 alarms to check, even for one month, is not realistic. On the other hand, the results of COD show its ability to separate malicious behaviour from normal usage. Our false alarms correspond to normal requests that are common to both sites but rarely occur. For instance, on the references interrogation script of *anonym_lab1*, a user might request papers of "John Doe" and the request will be `publications.php?FName=John\&LName=Doe`. If another user requests papers of "John Rare" on the Web site of *anonym_lab2*), the request will be `biblio.php?FName=John\&LName=Rare` and the parameter "John" will be given as a common outlier and trigger an alarm. As we can see, $\%_{FA}$ is very low (usually we have at most 5 false alarms in our experiments for both Web sites) compared to the thousands of outliers that have been filtered by COD. Another lesson from these experiments is that a low MD implies very small clusters and numerous outliers. These outliers are shared between both sites, among which some are false alarms due to rare but common normal usage. When MD increases, the clustering process gets more agglomerative and alarms are grouped together. Then one alarm can cover several ones of the same kind (*e.g.* the case of easter eggs explained further). At the same time, the number of outliers corresponding to normal usage decreases (since they are also grouped together). Eventually, a too large value of MD implies building clusters that do not really make sense. In this case, outliers will get larger, and the matching criteria will get too tolerant, leading to a large number of matching outliers capturing normal usage. In a streaming environment involving the real data of these experiments, one could decide to keep 70 as the number of desired alarms and watch the

ratio of false alarms. If this ratio decreases, then the end-user should consider increasing the number of desired alarms.

	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5
O_1	13197	10860	8839	7714	6547	5607	5184	4410	3945	3532
$\%_1$	43.3%	35.6%	29%	25.3%	21.5%	18.4%	17%	14.4%	12.9%	11.6%
O_2	35983	27519	24032	20948	18152	14664	12738	11680	10179	8734
$\%_2$	49.6%	37.9%	33.1%	28.9%	25%	20.2%	17.5%	16.1%	14%	12.1%
COD	101	78	74	70	67	71	71	85	89	90
$\%_{FA}$	4.9%	5.12%	4%	2.85%	1.5%	2.8%	2.8%	10.6%	11.2%	16.6%

Table 1. Results on real data

4.2 A sample of our results

None of the attacks found in our experiments have been successful on the considered Web sites. However, our security services and our own investigations allow us to confirm the intrusion attempts that have been discovered by our method:

- **Code Injection:** a recent kind of attack aims to inject code in PHP scripts by giving a URL in the parameters. Here is a samle of such URLs detected by COD:
  - http://myweddingphotos.by.ru/images?
  - http://levispotparty.eclub.lv/images?
  - http://0xg3458.hub.io/pb.php?

  Depending on the PHP settings on the victim's Web server, the injected code allows modifying the site. These URLs are directly, automatically and massively given as parameters to scripts through batches of instructions.
- **Passwords:** another kind of (naive and basic) attack aims to retrieve the password file. This results in outliers containing parameters like `../etc/password` with a varying number of `../` at the beginning of the parameter. This is probably the most frequent attempt. It is generally not dangerous but shows the effectiveness of our method.
- **Easter Eggs:** this is not really an intrusion but if one adds the code ?=PHPE9568F36-D428-11d2-A769-00AA001ACF42 to the end of any URL that is a PHP page, he will see a (funny) picture on most servers. Also on April 1st (April Fool's Day), the picture will replace the PHP logo on any phpinfo() page. This code (as well as two other ones, grouped into the same outlier) has been detected as a common outlier by COD.

5 Conclusion

In this paper, we have proposed i) an unsupervised clustering scheme for isolating atypical behaviours, ii) a parameterless outlier detection method based on

wavelets and iii) a new feature for characterizing intrusions. This new feature is based on the repetition of an intrusion attempt from one system to another. Actually, our experiments show that atypical behaviours cannot be directly used to trigger alarms since most of them correspond to normal requests. On the other hand, this very large number of outliers can be effectively filtered (reducing the amount of atypical behaviours up to 0.21%) in order to find true intrusion attempts (or attacks) if we consider more than one site. Eventually, our method guarantees a very low ratio of false alarms, thus making unsupervised clustering for intrusion detection effective, realistic and feasible.

Acknowledgement

The authors want to thank Laurent Mirtain, the responsible for intrusion detection of Inria Sophia-Antipolis, for his assistance in identifying attacks in our access log files.

References

1. D. Barbara, N. Wu, and S. Jajodia. Detecting novel network intrusions using bayes estimators. In *1st SIAM Conference on Data Mining*, 2001.
2. E. Bloedorn, A. D. Christiansen, W. Hill, C. Skorupka, and L. M. Talbot. Data mining for network intrusion detection: How to get started. Technical report, MITRE, 2001.
3. E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S Stolfo. A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data. *Applications of Data Mining in Computer Security*, 2002.
4. A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *3rd SIAM DM*, 2003.
5. W. Lee and S. J. Stolfo. Data mining approaches for intrusion detection. In *7th conference on USENIX Security Symposium*, 1998.
6. A. Marascu and F. Masseglia. A multi-resolution approach for atypical behaviour mining. In *The 13th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'09)*, Bangkok, Thailand, 2009.
7. A. Patcha and J.-M. Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Comput. Networks*, 51, 2007.
8. M. Roesch. SNORT, 1998.
9. A. Valdes and K. Skinner. Probabilistic alert correlation. In *Recent Advances in Intrusion Detection*, pages 54–68, 2001.
10. N. Wu and J. Zhang. Factor analysis based anomaly detection. In *IEEE Workshop on Information Assurance*, 2003.