**[IDS]**

**Data Mining Project**

**Documentation**

**Authors:**

**Kamil Breński**

**Piotr Luboń**

****

1. **The Problem**

The main problem we are trying to solve is the problem of network based intrusion detection. In other words, we have access to the network traffic flowing in and out of network, using it, can we determine if there is someone that is trying to do something malicious? The intrusion detector learning task is to build a predictive model (i.e. a classifier) capable of distinguishing between “bad” connections, called intrusions or attacks, and “good” normal connections. There is definitely a lot of data we can analyze when using network sniffing tools like the UNIX tool called tcpdump, too much for a human to inspect and analyze by hand, therefore it seems like a perfect place for data mining to be applied.

1. **The Solution**

There exist two main detection approaches when it comes to Intrusion Detection System (IDS). First one uses signature detection which means that we have a database of signatures and we check the network traffic for these signatures. This technique is criticized often since it requires the administrator to constantly update the signature database, and even a properly maintained and updated database does not protect against new attacks that don’t have signatures made for them yet. The second approach uses anomaly detection. This means that we are trying to find “anomalies” or connections that we haven’t seen before. The criticism of this method is that it produces a lot of false positives as traffic for new services and applications might get flagged as anomalies

We have collectively decided that we want to implement this project in python, mostly because with python you don’t have to write large amounts of code to get something done. It also has a large number of libraries that could have been useful for us. We also decided to try one of data mining techniques called decision tree classification in order to divide a large amount of network traffic into the normal group or a group that represents some kind of an attack.

We first started working on generating our own dataset from our own computers. To accomplish that we used a python library called “scapy” which is an interactive packet manipulation library. It can work very similar to the Linux tool called tcpdump discussed earlier with much more flexibility as it is able to decode raw binary packet data into more human readable format. We created a script called sniff2db.py that used scapy to monitor network traffic and insert certain features like source/destination IP address and port, along with state of 8 TCP flags (FIN, SYN, RST, PSH, ACK, URG, ECE, CWR) into an SQLite database. Unfortunately after reading some more white-papers on the subject we found out that using only these 12 extracted features would not be enough in order to do what we wanted to do, and we were forced to abandon this idea and look for a large dataset that had much more features extracted.

The dataset we found and used for training our model is a 708 MB file of comma separated textual data. Each row represents a connection that is composed of various features. We define a connection as a sequence of TCP packets starting and ending at some well-defined times, between which data flows to and from a source IP address to a target IP address under some well-defined protocol. Each connection is described by 40 different features as shown in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Feature name | Description | Type |
| Basic features of each network connection: | | | |
| 0 | duration | Number of seconds the connection is established | continuous |
| 1 | protocol\_type | The name of the protocol used, e.g. tcp, udp, etc. | discrete |
| 2 | service | The network service on the destination, e.g., http, telnet, etc. | discrete |
| 3 | flag | Status of the connection – normal or error | discrete |
| 4 | src\_bytes | The number of data bytes from source to destination | continuous |
| 5 | dst\_bytes | The number of data bytes from destination to source | continuous |
| 6 | land | If source and destination IP addresses and port numbers are equal then, this variable takes value 1 else 0 | discrete |
| 7 | wrong\_fragment | Total number of wrong fragments (checksum error) in this connection | continuous |
| 8 | urgent | Number of packets in this connection with urgent bit set. | continuous |
| Content related features of each connection: | | | |
| 9 | hot | Number of “hot” indicators such as entering a system directory, creating programs and executing programs. | continuous |
| 10 | num\_failed\_logins | Count of failed login attempts | continuous |
| 11 | logged\_in | Status of login session: 1 if logged in, 0 otherwise | discrete |
| 12 | num\_compromised | number of “compromised” conditions | continuous |
| 13 | root\_shell | 1 if root shell is obtained; 0 otherwise | discrete |
| 14 | su\_attempted | 1 if “su root” command attempted; 0 otherwise | discrete |
| 15 | num\_root | Number of operations executed as root in the connection | continuous |
| 16 | num\_file\_creations | Number of file creation operations in the connection | continuous |
| 17 | num\_shells | Number of shell prompts | continuous |
| 18 | num\_access\_files | Number of operations on access control files | continuous |
| 19 | num\_outbound\_cmds | Number of outbound commands in an ftp session | continuous |
| 20 | is\_hot\_login | 1 if the login belongs to the “hot” list i.e. root or admin; else 0 | discrete |
| 21 | is\_guest\_login | 1 if the login is a “guest” login; 0 otherwise | discrete |
| Time related network features of each connection: | | | |
| 22 | count | Number of connections to the same host as the current connection in the past two seconds | continuous |
| 23 | srv\_count | Number of connections to the same service as the current connection in the past two seconds | continuous |
| 24 | serror\_rate | The percentage of connection that have activated flag (3) s0, s1, s2, or s3, among the connections aggregated in count(22) | continuous |
| 25 | srv\_serror\_rate | The percentage of connection that have activated flag (3) s0, s1, s2, or s3, among the connections aggregated in count(23) | continuous |
| 26 | rerror\_rate | The percentage of connection that have activated flag (3) REJ, among the connections aggregated in count(22) | continuous |
| 27 | srv\_rerror\_rate | The percentage of connection that have activated flag (3) REJ, among the connections aggregated in count(23) |  |
| 28 | same\_srv\_rate | The percentage of connection that were to the same service, among the connections aggregated in count(22) | continuous |
| 29 | diff\_srv\_rate | The percentage of connection that were to a different services, among the connections aggregated in count(22) | continuous |
| 30 | srv\_diff\_host\_rate | The percentage of connection that were to a different destination machines, among the connections aggregated in count(23) | continuous |
| Host-based traffic features of each connection: | | | |
| 31 | dst\_host\_count | Number of connections having the same destination host IP address |  |
| 32 | dst\_host\_srv\_count | Number of connections having the same port number |  |
| 33 | dst\_host\_same\_srv  \_rate | The percentage of connections that were  to the same service, among the connections aggregated in dst\_host\_count (31) |  |
| 34 | dst\_host\_diff\_srv  \_rate | The percentage of connections that were  to a different service, among the connections aggregated in dst\_host\_count (31) |  |
| 35 | dst\_host\_same  \_src\_port\_rate | The percentage of connections that were  to the same source port , among the connections aggregated in dst\_host\_srv\_count (32) |  |
| 36 | dst\_host\_srv\_  diff\_host\_rate | The percentage of connections that were  to the different destination machines, among the connections aggregated in dst\_host\_srv\_count (32) |  |
| 37 | dst\_host\_serror  \_rate | The percentage of connections that have activated the flag (4) s0, s1, s2, or s3, among the connections aggregated in dst\_host\_count (31) |  |
| 38 | dst\_host\_srv  \_serror\_rate | The percentage of connections that have activated the flag (4) s0, s1, s2, or s3, among the connections aggregated in dst\_host\_srv\_count (32) |  |
| 39 | dst\_host  \_rerror\_rate | The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst\_host\_count (31) |  |
| 40 | dst\_host\_srv  \_rerror\_rate | The percentage of connections that have activated the flag (4) REJ, among the connections aggregated in dst\_host\_srv\_count (32) |  |
|  |  |  |  |

There are 4 major attack classes and 23 attack types that belong to one of the classes. The attack classes explained below:

1. **DOS –** Denial of service is an attack category, which depletes the victim’s resources thereby making it unable to handle legitimate requests – e.g. SYN flooding. Relevant features include source bytes and percentage of packets with errors.
2. **Probing** – Surveillance and other probing attack’s objective is to gain information about the remote victim e.g. port scanning. Relevant features include duration of connection and source bytes.
3. **U2R** – Unauthorized access to local super user (admin) privilege is an attack type, by which an attacker uses a normal account to login into a victim system and tries to gain root/admin privileges by exploiting some vulnerability in the victim e.g buffer overflow attacks. Relevant features include number of file creations and number of shell prompts invoked.
4. **R2L** –Unauthorized access from a remote machine, the attacker intrudes into a remote machine and gains local access of the victim machine. E.g password guessing. Relevant features include duration of connection, service requested, and host level features like number of failed login attempts.

The mapping of attack class with attack type is shown below:

|  |  |
| --- | --- |
| Attack Class | Attack Type |
| DOS | Back, Land, Neptune, Pod, Smurf, Teardrop |
| Probe | Satan, Ipsweep, Nmap, Portsweep |
| U2L | Buffer\_overflow, Loadmodule, Rootkit, Perl |
| R2L | Guess\_password, Ftp\_write, Imap, Phf, Multihop, Warezmaster, Warezclient, Spy |

1. **Algorithmic Complexity and Correctness Analysis**

After doing some more research we decided to try out the C4.5 algorithm in order to construct our decision tree.

1. **User’s Manual**

LOREM IPSUM

1. **Technical Documentation**

LOREM IPSUM

1. **Tests**

LOREM IPSUM

1. **Conclusion**

LOREM IPSUM

Resources:

http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

<http://www.cl.cam.ac.uk/~awm22/publications/li2007machine.pdf>

https://pdfs.semanticscholar.org/7765/e955db40e0c4634db0f212319999473e8b89.pdf

http://www.ijarcce.com/upload/2015/june-15/IJARCCE%2096.pdf