

TPR - Ransomware

Pedro Almeida - 89205

Pedro Mateus - 88858

Problem identification

Ransomware is a public affair. It's a type of malicious software that encrypts a user's computer or device, sending the user a blatant message telling them their system has been compromised.


The inspiration for this project came from an attack made against the company Garmin in July 2020 in which the hackers seized control of all the files on the system and demanded to be paid a ransom of \$10 million to decrypt the compromised data.

Link to the news: <https://www.mitnicksecurity.com/blog/2020-garmin-ransomware-attack>

Our implementation

Through a docker image of owncloud we created a directory that automatically synchronizes the files with the cloud.

In order to simulate normal use, we made a 30 minutes capture on wireshark using the following command:

A terminal window with a dark background. The prompt is 'Pedralmeida' followed by a green arrow icon and a tilde '~'. The command entered is 'tshark -i 10 -f "net 10.10.2.0/31" -s 64 -w normal_use.pcap|'.

```
Pedralmeida ~ tshark -i 10 -f "net 10.10.2.0/31" -s 64 -w normal_use.pcap|
```

During the time of the capture we inserted files both in the local directory and in the cloud directory, we also edited and deleted files.

For the attack capture we ran a script that cyphers and deletes files inside the directory. The scripts also uses some random sleeps to make the script seem more “machine like”.

Features - What worked and what didn't

In the previous presentation we had decided to use features like Minimum, Maximum and Standard Deviation (for time independent features).

After implementing the Minimum and Maximum features we noticed that they were not helpful as both of these values were the same for the attack and normal use.

For the final implementation we used the following features:

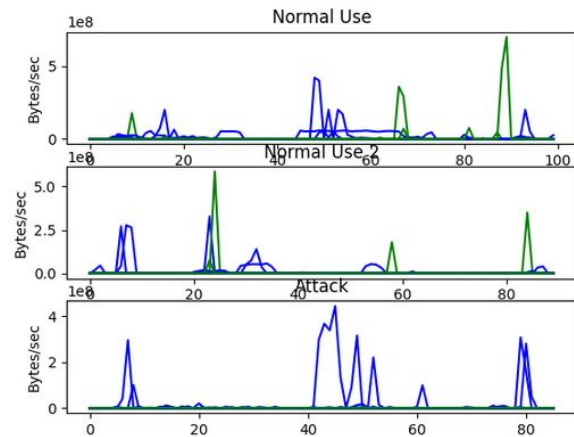
- Time Independent Features
 - Standard deviation
 - Mean
 - Kurtosis
 - Percentile
- Time Dependent Features
 - Mean
 - Variance
 - Standard Deviation

System Training - Machine Learning

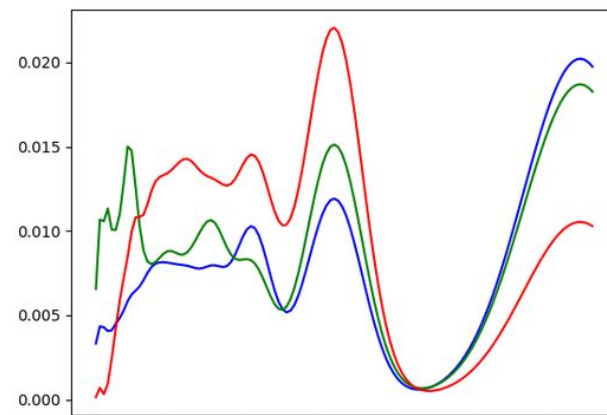
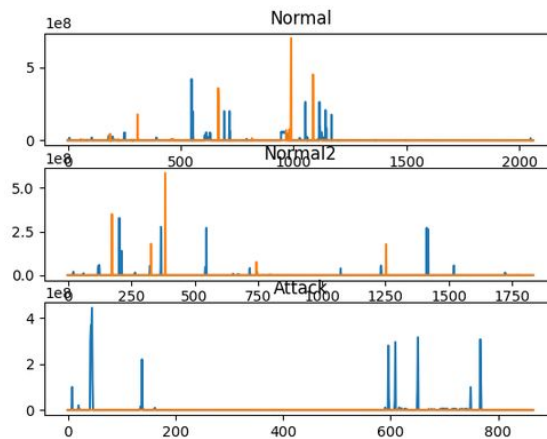
Since the objective is to perform the detection of anomalies, the system was trained with a dataset with only the “normal” data.

The “attack” data was only used when testing the system.

ATTACK 1

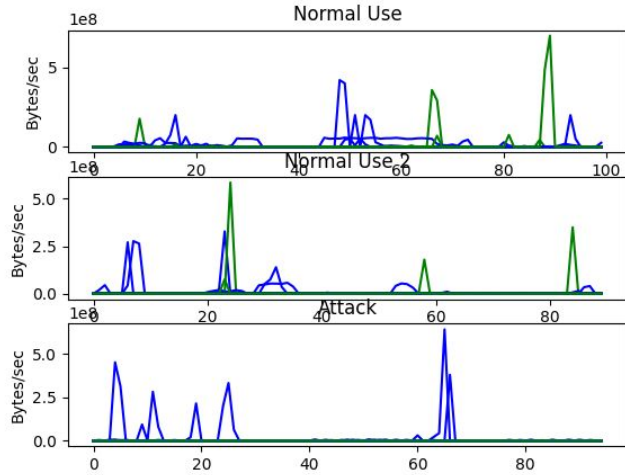


Upload/Download -
Observation Windows

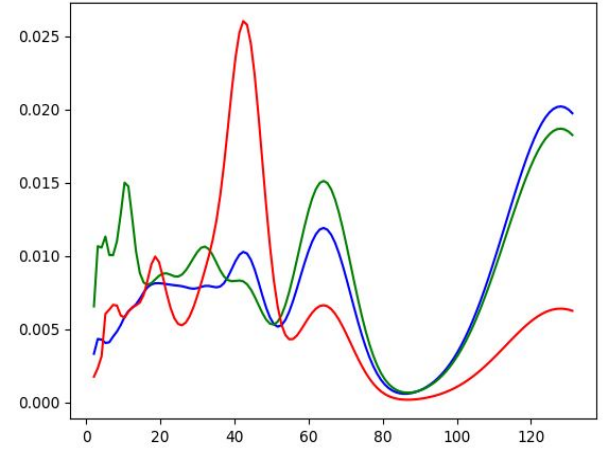


Periodicity

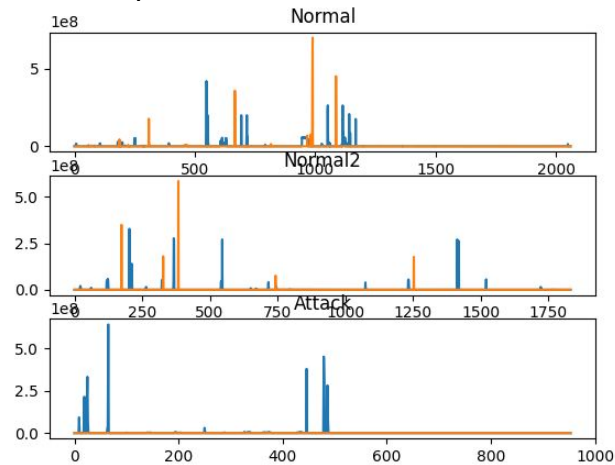
ATTACK 2



Upload/Download -
Observation Windows



Periodicity



Results - Data Sets

For normal use we captured two sets of data, the first one has very few moments of silence and therefore, a lot of activity. The second set of data has its activity more spread out with more moments of silence.

As expected the second set of data presents much better results, resulting in a much lower number of misconceptions in detecting the normal use.

The same procedure was made for the attack, the first attack we made had a reduced number of silence periods making it easier to detect anomalies. The second attack has longer periods of silence making the attack seem more “human like” and therefore, harder to detect anomalies.... or is it?

Results - Attack 1 (fewer periods of silence)

```
-- Anomaly Detection based on One Class Support Vector Machines --
```

Obs: 0	(Normal): Kernel Linear->OK	Kernel RBF->OK	Kernel Poly->Anomaly
Obs: 1	(Normal): Kernel Linear->OK	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 2	(Normal): Kernel Linear->Anomaly	Kernel RBF->Anomaly	Kernel Poly->Anomaly
Obs: 3	(Normal): Kernel Linear->Anomaly	Kernel RBF->Anomaly	Kernel Poly->Anomaly
Obs: 4	(Normal): Kernel Linear->Anomaly	Kernel RBF->Anomaly	Kernel Poly->Anomaly
Obs: 5	(Normal): Kernel Linear->Anomaly	Kernel RBF->Anomaly	Kernel Poly->Anomaly
Obs: 6	(Normal): Kernel Linear->Anomaly	Kernel RBF->Anomaly	Kernel Poly->Anomaly
Obs: 7	(Normal): Kernel Linear->Anomaly	Kernel RBF->Anomaly	Kernel Poly->Anomaly
Obs: 8	(Normal): Kernel Linear->Anomaly	Kernel RBF->Anomaly	Kernel Poly->Anomaly
Obs: 9	(Normal): Kernel Linear->Anomaly	Kernel RBF->OK	Kernel Poly->Anomaly
Obs: 10	(Normal2): Kernel Linear->OK	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 11	(Normal2): Kernel Linear->OK	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 12	(Normal2): Kernel Linear->OK	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 13	(Normal2): Kernel Linear->OK	Kernel RBF->OK	Kernel Poly->OK
Obs: 14	(Normal2): Kernel Linear->OK	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 15	(Normal2): Kernel Linear->OK	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 16	(Normal2): Kernel Linear->OK	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 17	(Normal2): Kernel Linear->OK	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 18	(Normal2): Kernel Linear->OK	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 19	(Normal2): Kernel Linear->OK	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 20	(Attack): Kernel Linear->Anomaly	Kernel RBF->Anomaly	Kernel Poly->Anomaly
Obs: 21	(Attack): Kernel Linear->Anomaly	Kernel RBF->Anomaly	Kernel Poly->Anomaly
Obs: 22	(Attack): Kernel Linear->Anomaly	Kernel RBF->Anomaly	Kernel Poly->Anomaly
Obs: 23	(Attack): Kernel Linear->OK	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 24	(Attack): Kernel Linear->Anomaly	Kernel RBF->Anomaly	Kernel Poly->Anomaly
Obs: 25	(Attack): Kernel Linear->Anomaly	Kernel RBF->Anomaly	Kernel Poly->Anomaly

```
linear %: 69.23076923076923
rbf %: 46.15384615384615
poly %: 65.38461538461539
```

```
-- Anomaly Detection based on One Class Support Vector Machines (PCA Features) --
```

Obs: 0	(Normal): Kernel Linear->OK	Kernel RBF->OK	Kernel Poly->Anomaly
Obs: 1	(Normal): Kernel Linear->OK	Kernel RBF->OK	Kernel Poly->OK
Obs: 2	(Normal): Kernel Linear->Anomaly	Kernel RBF->OK	Kernel Poly->Anomaly
Obs: 3	(Normal): Kernel Linear->Anomaly	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 4	(Normal): Kernel Linear->Anomaly	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 5	(Normal): Kernel Linear->Anomaly	Kernel RBF->OK	Kernel Poly->Anomaly
Obs: 6	(Normal): Kernel Linear->Anomaly	Kernel RBF->OK	Kernel Poly->OK
Obs: 7	(Normal): Kernel Linear->OK	Kernel RBF->OK	Kernel Poly->Anomaly
Obs: 8	(Normal): Kernel Linear->Anomaly	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 9	(Normal): Kernel Linear->OK	Kernel RBF->OK	Kernel Poly->Anomaly
Obs: 10	(Normal2): Kernel Linear->OK	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 11	(Normal2): Kernel Linear->OK	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 12	(Normal2): Kernel Linear->OK	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 13	(Normal2): Kernel Linear->OK	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 14	(Normal2): Kernel Linear->OK	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 15	(Normal2): Kernel Linear->OK	Kernel RBF->OK	Kernel Poly->Anomaly
Obs: 16	(Normal2): Kernel Linear->OK	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 17	(Normal2): Kernel Linear->OK	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 18	(Normal2): Kernel Linear->OK	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 19	(Normal2): Kernel Linear->OK	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 20	(Attack): Kernel Linear->OK	Kernel RBF->OK	Kernel Poly->Anomaly
Obs: 21	(Attack): Kernel Linear->Anomaly	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 22	(Attack): Kernel Linear->Anomaly	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 23	(Attack): Kernel Linear->Anomaly	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 24	(Attack): Kernel Linear->Anomaly	Kernel RBF->Anomaly	Kernel Poly->OK
Obs: 25	(Attack): Kernel Linear->Anomaly	Kernel RBF->Anomaly	Kernel Poly->OK

```
linear %: 61.53846153846154
rbf %: 34.61538461538461
poly %: 61.53846153846154
```


Conclusion

From our analysis whilst making test runs we concluded that bigger observation windows provided better results for both attacks.

From the results shown previously we can conclude that the RBF algorithm is the worst.

Without PCA features both the Linear and the Poly algorithm provided the same results with very satisfactory percentages of hits.

With PCA features, the Linear algorithm provided the best results in attack 1 (reduced number of silence periods) and the Poly algorithm provided the best results in attack 2 (longer periods of silence).