Intrusion Detection Systems
Active Authentication

Ana Maria Martinez Gomez - anamaria@martinezgomez.name

June 1, 2020

1 Data gathering

I used osquery daemon with the configuration in the osquery.conf file. In this file the features to be collected are specified as well as other configuration options. Osquery allows to write a sensor that is multiplatform. My sensor should work in MacOS and any Linux distribution without changes, although not in Windows, as many of the tables I selected are not available for Windows. As we will see, there are some Linux specifics, that affected some decisions during its design.

With osquery we can query the status of the computer using SQL queries, which allows to easily define custom and complex features. To log the features, explained in the next section, snapshot queries are used. A snapshot is the state in a concrete point in time. I do not need every single event and doing it this way, the log file is smaller, as I log one query per feature every minute instead of one per event. It also makes aggregating the data simpler.

Another thing worthwhile mentioning is that I am only monitoring the file activity in my home directory and the `tmp` directory. This because osquery uses inotify in Linux to subscribe to file changes at the kernel level for performance, limiting the files that can be monitored. Increasing the limit affects the kernel memory, so that's why I decided to only monitor those two directories, which I thought may be enough. The exclusion of the path `/home/ana/uni`, which is a symbolic link to a subdirectory, avoids loops. I consider excluding other files such as `.swp` files or files like `.bash_history`, as they are not files I create manually. I decided it to not exclude them, as if it is noise, it is noise which is always present and if not, they could be representative of the computer activity even if the feature not exactly represents what originally intended.

2 Features

I tried to use similar features to the one suggested in [1], although I modified or replaced some of them because it was not possible or not easy to implement them in Linux and/or osquery. I concretely focused on the features with higher Fisher discriminant in [1], as they are expected to be more useful in discriminanting different behavior patterns.

2.1 Processes created

As suggested in [1], I logged the number of processes created every minute. This can be done with a simple query in osquery after enabling process auditing.

2.2 Number of processes

I didn't find a way in osquery to log the number of deleted processes, as suggested by [1]. As I was already logging the created processes, I thought that adding the total number of processes when raising the query would be a similar combination of features. Note that that when aggregating the data, instead of adding all the query results in the window, as I do with most of the features, I perform the average. This could make that when using big windows sizes, anomalous values get loss. Because of that, when aggregating the data, I add as a feature the standard deviation of the number of processes as well.

2.3 Number of listening ports

I also didn't find an easy way to log the number of ports opened and closed as suggested in [1]. Instead I logged the number of processes with listening ports. As with the number of processes, when aggregating the data I added both the average and standard deviation as features.

2.4 File changes

As suggested in [1], I logged the number of file changes every minute. This can be done with a simple query in osquery with File Integrity Monitoring.

2.5 Documents touched

Logging documents in Linux is tricky, as Linux does not rely on filename extensions, as it happens with Windows. Linux uses the information in the file header to determine the file type and not the extension, but file headers can not be filtered with osquery. However, extensions are often use as a convention and to make interacting with other operating systems easier. That's why I decided to keep this feature, adding `.odt` and `.ods` to the extensions mentioned in the paper. These extensions are used by LibreOffice, the open source office suite commonly used in Linux. The function `documents_query` in script-generate-queries.rb was used to generate this query.

In [1], about both file changes and files touched are mentioned. As the execution of the `touch` command updates the change time and both things would be the same, I think that what is meant with touched is accessed. Knowing which files have been accessed in Linux is not possible in a reliable way. The mount option `relatime` is the kernel default since Linux 2.6.30 (from 2009) and modifying this would have performance implications. With this option, accessed time is only updated if the previous access time was more than 1 day old or earlier than the current modify or change time. That may be the reason why osquery doesn't provide events for files accessed. So for example, executing the following instructions will be completely undetected by osquery:

```
cat /home/ana/a.txt
sudo cp /home/ana/a.txt /home     # /home is not logged
```

Consequently, I am only logging documents updated or whose attributes are updated.

2.6 Files moved

There is no way in Linux to know which files are renamed, as suggested in [1]. I logged the number of file events where `action` is `MOVE_FROM` or `MOVE_FROM`, as it seems reasonable that is an equivalent feature.

2.7 Decoys activated

I followed a similar way as the one used in [1] to create the decoys. To determine the location to place the decoys, I got the top 10 directories containing the most files in the first level using the following command (taken from [2]):

```
find /home/ana -xdev -type d -exec sh -c '
echo "$(find "$0" | grep "^$0/[^/]*$" | wc -l) $0"' {} \; | sort -rn | head -10
```

Those are the results:

```
12186 /home/ana/.cache/chromium/Default/Cache
9354 /home/ana/.cache/thumbnails/normal
8483 /home/ana/.cache/mozilla/firefox/n6fu3vn8.default/cache2/entries
4648 /home/ana/.local/share/okular/docdata
2755 /home/ana/.cache/chromium/Default/Code Cache/js
1338 /home/ana/.pyenv/versions/3.5.3/lib/python3.5/test/__pycache__
657 /home/ana/github/osem/tmp/cache/assets/sprockets/v3.0
541 /home/ana/.pyenv/versions/3.5.3/lib/python3.5/test
532 /home/ana/.bundle/cache/compact_index/rubygems.org.443.29b0360b937aa4d161703e6160654e47/info
507 /home/ana/.pyenv/versions/3.5.3/lib/python3.5/__pycache__
```

Creating a decoy in those directories won't interfere with my activity, as they are cache and library directories where I do not copy anything manually, with the exception of the following:
`/home/ana/github/osem/tmp/cache/assets/sprockets/v3.0`.
`/home/ana/github/osem` is a git repository. Creating a decoy there will without no doubt interfere with my activity, as I could commit it by accident, and I will most likely delete it fast. Consequently, I only considered the other 9 directories and I added the top directory containing the most files in the first level in the `home/ana/Documents` directory, which is `/home/ana/Documents/uni/1-semester/analysis_of_algorithms/lectures`. For every of the directories, I found the last modified file with the following command and created a decoy with a similar name:

```
ls -pc | grep -v / | head -1
```

Note that [1] also suggests to use the directories with the most recently accessed documents. As explained before, both documents extensions and accessed times are not reliable. Thus, I didn't implement that part. Last, I created the decoy `/home/ana/.password`. Malware commonly looks for hidden files with passwords, so I thought that could be a good decoy.

Remember that, as explained in the Documents touched section, accessing decoys doesn't trigger this feature, as we don't have a way to log accessed files in Linux.

2.8 Shell commands

There were other features in [1] I couldn't log such as the number of window touches. Because of that I decided to add a more original and personalized feature. I registered all the executed bash commands in any of my consoles by modifying my bash configuration. After a week, I used the script script-command-line.rb to figure out the most used bash commands. Those were the results:

```
1366 command executed. Most common ones:
  0. sudo - 499
  1. la - 104
  2. osqueryi - 93
  3. vim - 92
  4. cd - 86
  5. git - 51
  6. open - 48
  7. cp - 38
  8. driveup - 24
  9. irb - 13
```

I didn't consider `osqueryi`, as the reason I called it that many times during that week is that I was setting `osquery` for the project up and I do not expect to use it again during the data collection. I collected how many commands of each type `0,1,3-9` and how many different from those one had been executed. Note that similarly as it happened with the number of processes and the number of listening ports, I do not log the number of commands executed in a time of period, but the number of commands since I have started logging at a concrete point in time (I delete the `.bash_history` file before starting osqueryd). When parsing/aggregating the data, I removed the number of commands in the previous windows, getting the number of commands executed in the current window, which is what I used as features.

Worthwhile mentioning is that I had problems to get the history of the `ana` user, as osqueryd is started as root. I am not sure if this is an osqueryd bug. To fix it, I replace `/root/.bash_history` by a symbolic link to `/home/ana/.bash_history`, making that both users share the bash history.

3 Collected data

I have collected data for 4 days to use it as training data for the ML model and for an extra day to evaluate the false positives rate. I allowed a "trusted" adversary to use my computer during 50 minutes to collect data to determine the intrusion detection rate. The adversary is a technical user with extensive knowledge about my operative system. In the 50 minutes, he compiled emacs, stole my `personal documents` folder, several key files (`.ssh`, `.gpg`, etc.) and files containing passwords in plain text (such as the password for openSUSE Build Service) and copied them using scp (ssh). In other words, he stole the majority of the most important files in my computer. As a curiosity, no decoys were activated neither by me or the adversary.

The collected data can be found in the logs directory, however the paths and ports fields have been removed for privacy concerns. This was done using the following vim command: `%s/,"\(ports\|paths\)":",\{-}"/`

4 ML model and result

are_you_you.py parses the logs and prints false positives and intrusion detection rates using several ML algorithms and different windows sizes. For every of the algorithms, the model is first trained with the data of the first 4 days and the false positives calculated by predicting the fifth day with this model. Afterwards, the model is re-trained with all my data (the five days) and this new model is used to calculate the intrusion detection rate of the adversary data. I calculate both false positives and intrusion detection rates 10 times using 10 different seeds (from 1 to 10) and average them to get a deterministic program, which I can use to compare the different models and the effectiveness features. Those are the results of the program:

```
Windows size: 2
  Using GaussianMixture
    False positives rate 4.39
    Intrusion detection rate: 94.23
  Using IsolationForest
    False positives rate: 4.17
    Intrusion detection rate: 72.69
  Using EllipticEnvelope contamination = 0.01
    False positives rate: 1.34
    Intrusion detection rate: 45.77
  Using EllipticEnvelope contamination = 0.3
    False positives rate: 8.5
    Intrusion detection rate: 94.23

Windows size: 5
  Using GaussianMixture
    False positives rate 2.46
    Intrusion detection rate: 88.0
  Using IsolationForest
    False positives rate: 5.95
    Intrusion detection rate: 87.0
  Using EllipticEnvelope contamination = 0.01
    False positives rate: 1.11
    Intrusion detection rate: 40.0
  Using EllipticEnvelope contamination = 0.3
    False positives rate: 7.46
    Intrusion detection rate: 99.0
```

```
Windows size: 10
  Using GaussianMixture
    False positives rate 7.14
    Intrusion detection rate: 100.0
  Using IsolationForest
    False positives rate: 6.67
    Intrusion detection rate: 100.0
  Using EllipticEnvelope contamination = 0.01
    False positives rate: 1.59
    Intrusion detection rate: 60.0
  Using EllipticEnvelope contamination = 0.3
    False positives rate: 8.73
    Intrusion detection rate: 100.0
```

As you can observed in the results, the selected features work really well, getting very good results in all the models, which I have to admit was a bit surprising. I also thought that many of the features would not be reliable in small windows. But even for a Window of 2 minutes we get intrusion rates of at least 72% for all models with small false positives rates. Bigger windows increase the intrusion detection rates, but generally also the false positives rates.

Regarding the algorithms, I used one clustering algorithm, the Gaussian Mixture Model, with 10 clusters. This model is suggested in [1]. It is the one which works the best in small windows, proving high detection rates, with acceptable false positives rates Note that reducing the number of clusters generates much worse results. In particular, using 2 clusters (expecting a cluster for `me` and another for `different than me`) doesn't work well. I also used two anomaly detection algorithms: Isolation Forest and Elliptic Envelope. Elliptic Envelope has a contamination parameter, which allows to decide if we prefer a higher intrusion rate at the cost of having a higher false positives rate. You can observe this in the results for the value `0.01` and `0.3`. With respect to the seeds, different seeds produce different results, which I could have use to look for a convenient seeds for this case. For example, with the seed `38`, GaussianMixture false positives rate is only `0.64`, with an intrusion detection rate of `84.62`.

Last, let's speak about how good every feature is working. I figured this out by disabling every of them individually to see how the results change. I did this just by changing the return condition in the `data_for` function. It seems like all the features contribute (even if it is slightly) to have a better prediction, with the exception of the number of decoys activated and files moved. That the decoys don't help is not surprising as no decoys were activated. Removing these two features gives us much better result in the Gausian Mixture model (there difference is not so big in the anomaly detection algorithms):

```
Windows size: 2
  False positives rate 0.7
  Intrusion detection rate: 81.92
```

```
Windows size: 5
  False positives rate 0.79
  Intrusion detection rate: 90.0

Windows size: 10
  False positives rate 1.43
  Intrusion detection rate: 100.0
```

Especially important for all the models are the shell commands features (number of every of the 9 most used commands and others). To the extent that only using this features, we get such good results using the Elliptic Envelope model with window size `2` and contamination factor `0.2` (this doesn't work with all the models):

```
False positives rate: 1.5
Intrusion detection rate: 70.77
```

All in all, we can conclude that the model can determine if I am who is using the computer.

5 Acknowledgments

Apart from the instructors of my Intrusion Detection Systems course, I would like to thank the following persons for their help with the project:

- Victor Gallego for his tips about machine learning.
- Alexander Graul for using my computer to generate adversary data.

References

[1] Jonathan Voris, Yingbo Song, Malek Ben Salem, Shlomo Hershkop, and Salvatore Stolfo. Active authentication using file system decoys and user behavior modeling: results of a large scale study. *Computers & Security*, 87:101412, 2019.

[2] Find the top 50 directories containing the most files/directories in their first level? `https://unix.stackexchange.com/questions/122854`.