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:

1. Use a machine learning based approach to create a model that learns normal network traffic.
2. Learn 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:

1. "Anomalous Payload-based Worm Detection and Signature Generation", Ke Wang and Salva- tore J.Stolfo, RAID2004
2. "Polymorphic Blending Attacks", Prahlad Fogla, Monirul Sharif, Roberto Perdisci, Oleg Kolesnikov, Wenke Lee, Usenix Security 2006
3. "True positive (true detections) and False positive (false alarms)"

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 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. 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. You will have to modify the port numbers in the read pcap.py (commented in the sourcecode) according to the protocol.
    - 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. 1. It computes the mahalanobis distance between each test payload and the model (of the same length), and 2. It labels the payload: If the mahalanobis distance is below the threshold, then it accept the payload as normal traffic. Otherwise, it reject the packet as attack traffic.
    - To run PAYL on testing mode: python wrapper.py [FILE.pcap]

```
$ python wrapper.py
Attack data not provided, training and testing model based on pcap files in data
  /folder alone.
To provide attack data, run as: python wrapper.py <attack-data-filename>
----------------------------------------------------------------------
Training
Testing
Total Number of testing samples: 7616
Percentage of True positives: XX.XX
Exiting now
```

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.
- Modify the IP addresses/port numbers (also commented in the python files) in the source code according to the traffic you are working with.
- 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.

Task B:

- Download your unique attack payload: To download your unique attack payload, visit the following url: http://www.prism.gatech.edu/~vseshadri30/Pcap/einstein7.pcap and replace "einstein7" with your GTID.

- Use PAYL in testing mode. Feed the training data that you used before, use the same pair of parameters that you found from Task A and provide the attack trace.
- Verify that your attack trace gets rejected - in other words that it doesn't doesn't fit the model.
- You should run as follows and observe the following output:

```
$ python wrapper.py attack-trace-test
Attack data provided, as command line argument attack-trace-test
-------------------------------------------------------------
Training
Testing
Total Number of testing samples: 7616
Percentage of True positives: XX.XX
----------------------------------------------------
Analysing attack data, of length1
No, calculated distance of ZZZ is greater than the threshold of YYY. It doesn't fit the model.
Total number of True Negatives: 100.0
Total number of False Positives: 0.0
Number of samples with same length as attack payload: 1
```

- Finally, use the artificial payload of the protocol provided. Test the artificial payload against your model(use testing mode as explained above). This packet should be accepted by your model. You should get an output that says "It fits the model".

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 he 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.
    - **Code provided:** Please look at the Polymorphic blend directory. All files (including attack payload) for this task should be in this directory.
    - **How to run the code:** Run *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. According to the paper the substitution table has to be an array of length 256. 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. You are free to create this table with one-to-one or one-to-many mapping as per your choice.
- **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.
- **Deliverables:** Please deliver your code for the substitution and the padding, and the output of your code. Please see section deliverables.

- **Test your output.** Test your output (below noted as 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:

```
$ python wrapper.py output
Attack data provided , as command line argument Output
————————————————————————————————————————————————————————
Training
Testing
Total Number of testing samples: 7616
Percentage of True positives: XX.XX
————————————————————————————————————————————
Analysing attack data, of length1
Yes, calculated distance of YYYY is lesser than the threshold of XXXX. It fits
the model.
Total number of True Negatives: 0.0
Total number of False Positives: 100.0
```

Deliverables & Rubric:

- **Task A: 35 points** Please report the protocol that you used and the parameters that you found in a file named parameters. Please report a decimal with 2 digit accuracy for each parameter.
  Format:
  |Protocol:HTTP|
  |Threshold:1.23|
  |SmoothingFactor:1.24|
  |TruePositiveRate:80.95|

- **Task B: 5 points** Please append a new line in parameters with the score of the attack payload after completing Task B.

Format:
|Distance:2000|

- **Task C: 60 points**
    - **Code: 40 points.** Please submit the code for substitution.py, substitution table.txt and padding.py.
    - **Output: 20 points.** Please submit your output of Task C generated as a new file after running task1.py.

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         # Or whatever your current gcc version is
sudo apt-get install gcc-multilib
```

Next, then create a Makefile with following:

```
a.out: shellcode.o payload.o
  gcc -g3 -m32 shellcode.o payload.o -o a.out
shellcode.o: shellcode.S
  gcc -g3 -c shellcode.S -m32 -o shellcode.o
payload.o: payload.bin
  objcopy -I binary -O elf32-i386 -B i386 payload.bin payload.o
```

Now, modify the hardcoded attack payload length at line no. 10 of shellcode.S with the length of your malicious attack payload. It should be an integer value equal to or the next multiple of 4 of your attack payload length. You can also get this number from task1.py and seeing what len(adjusted attack body) is. Without this, the code won't point to the correct xor table location.

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.

Sample Substitution Table:

Below is the one-line output generated using "print substitution table" in python. Your substitution table.txt should look like this:

```
{’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)]}
```

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