Intrusion Detection System in Python

[1]Mrinal Wahal, [2]Dr. Tanupriya Choudhury, [3]Manik Arora
[1,2,3] Amity University, Uttar Pradesh, India
[1] MrinalWahal@gmail.com, [2] tchoudhury@amity.edu, [3]manikarora04@gmail.com

Abstract— **Intrusion detection and prevention systems are an epitome of system security and network security by an extension. This paper essentially explains on how to make a basic intrusion detection system entirely in Python both by using external modules like Scapy or by designing layer 2 raw sockets. With the expanding application of computer networks, various and incredibly enhanced intrusive tools and mechanisms have surfaced which exploit the security of individuals and private organizations. These systems are designed to be as foolproof as possible and provide highest possible level of security from intruding agencies. Except that these systems/mechanisms are NOT completely fool-proof. Essentially these systems are particularly designed to either protect the host or protect an entire network or the mainframe, depending on the design.**

Index Terms — **Client-server systems, Computer hacking, Computer networks, IDS, Network security, NIDS.**

I. Introduction

This paper critically demonstrates a step by step process to develop a basic intrusion detection system in Python using both external and inbuilt modules, hence, drawing one of the most fundamental paradigms to explain the working and design of intrusion detection and prevention systems, along with clarifying how this base is further used to establish much advanced mechanisms for the same purpose. Example: IDPS using Neural Networks. The paper also initially sets the basic explanatory platform by explaining the taxonomy of IDPS and the common divisions of intrusion detection tools usually employed for the purpose. The rest of the portion of this paper is prepared as followed:

- Position of art work.
- Basic blueprint followed by maximum IDS.
- Classes of cyber-attacks.
- Taxonomy of employed tools.
- Model of operation which the system works on.
- Disadvantages and Back falls of designed system.
- Case Studies.

II. State of Ability of Work

Generally, the art of intrusion is aimed particularly towards encroaching or infringing the confidentiality or reliability of the targeted resource. This digital age is without doubt experiencing a rapid expansion of computer networks and therefore, it is only normal to consider and enhance the security with due responsibility. Considering the threats becoming increasingly devastating day by day, IDP techniques only become indispensable for the employed purpose.

Related Works

A the most fundamental level intrusion detection aims to detect computer attacks and needless to mention, overtime these techniques have elevated their precision for detection and prevention of intrusions and network harassments. For best practices, it is suggested to first comprehensively study the types of threats that exists and how these threats can be potentially mounted against individual hosts/networks. These threats may very well attempt to access/manipulate information or/and render the host inaccessible/dead.

A typical Network IDS attempts to detect unauthorized access/intrusions on network media like cables and wireless. And upon the targeted network, detection algorithms/mechanisms are executed to match the behavior with a preset normal behavior. And if the detected behavior turns out to be anything other than the expected normal behavior, the session is flagged as an anomaly and left for future administrators/engineers/programs to utilize it for inspection or review. Snort is particularly useful as a Network IDS [22].

Several models of detection are employed in the process and therefore, these tools/programs can be broadly categorized upon their models. One of the most documented/understood risks involved is the violation of integrity of operations due to intrusion/malfunction. Example: Anomaly detection or Misuse detection. The irregularity detectors institutionally differentiate and builds the expected profiles by distinction from the regular and irregular [1,2]. These, not only create a distinction between the flow of packets or their signatures, but are also advance modified to differentiate between who's allowed access to the information and who isn't, at a user level. Or who's further authorized to use a particular resource on the network.

The way a particular Neural Network based intrusion detection system would work for misuse detection, at the most compartmentalized level, is by first collecting the audit logs or the history of user commands. Then the network is fractionally trained to detect the authorized users on their history of used commands and should the output turn out to be anything else than the original property, it is flagged as an anomaly. Or further micro-operations are executed depending on how advanced or intricately designed the system is [3].

Advanced pattern matching models have also been introduced which employ techniques like Keystroke matching. This is one of the most basic approaches to counter intrusions in which user level keystroke sequences are matched and used to detect

978-1-5386-1719-9/18/$31.00 ©2018 IEEE

intrusions. One major disadvantage of this technique is the general unavailability of user keystrokes in order to initially train the expert system for further detection and increased protection. Another average level approach for misuse detection is State Transition Analysis. In this approach, the system states are first graphed carefully and are in turn assigned to particular Boolean states. These Boolean states are used as judging parameters during transition of system states and are heretofore translated into a potential attack or otherwise. This may not just be limited to a single audit trail but may also be a sequence of defined conditions [4].

It is true, inversely, that the increase in complexity and size of the networks and there, in turn, increased usage, the severity of the intrusions/attacks also elevates. Therefore, it gets gradually hard from any sole cyber security expert or any intrusion detection and prevention system to provide an authentic and credible response to the particular attack. Furthermore, to advance these systems for better protection and operation, various high-level machine learning algorithms and techniques are employed in the process. Neural Networks [5] and Genetic Algorithms [6] are proving to be one of the most efficient mechanisms/processes to achieve the task as evident by various previous works including an evidently reported accuracy of 97.65% [16]. Amongst other highly regarded ways to counter the intrusions are Support Vector Machines [10] and are being heavily incorporated in the development processes.

Often the administrators or engineers realize the need to update the existing intrusion detection systems. This process is often expensive and slow. Therefore, data mining has been proven to be extremely efficient in countering this issue. The central idea for data mining frameworks being employed in IDS is to adopt various auditing programs to collect and design the deep features of every network connection and host session. And then mining frameworks are designed to learnt the pre-existing rules to learn the relationships of/in normal behavior of the target environment. This method is further modified at a working level to be critically used for anomaly detection. And co-operative learning processes are used to design a meta detection model which utilizes the output from several other models [7].

Various hybrid intrusion detection mechanisms have been well employed together for better achievements in much historical experiments. This famously includes using Clustering ID3 Decision Tree Learning Methods and K-Means[12] with an accuracy of about 96.24%, using Decision trees, principal component analysis, SPegasos (Stochastic variant of Piramol estimated sub-gradient solver in SVM) [13] to go much higher at an accuracy of 99.5% and ultimately reaching an ever high of 99.8% using the Naive Bayes, J48 (decision tree), JRip (rule induction) and iBK (nearest neighbour) [14] altogether.

In fuzzy logic based intrusion detection systems, generally, the proposed algorithms are employed to provide higher efficiency than regular data mining algorithms as a target. The general approach followed according to the literature is first classifying the training data (like KDD Cup 99 dataset) followed by defining a strategy for fuzzy rules and then finally ending with a fuzzy decision module which outputs a certain answer employing the specified fuzzy rules on each input and subsequently calculating the output for the same [8,9]. Ultimately, these systems have proved to provide higher reliability than the traditional systems based on regular machine learning algorithms. Intrusion detection, in terms of accuracy, is enhanced by repeatedly fuzzifying the set rules and sending it to the fuzzy system which classifies the test data.

Self-designed hybrid/meta mechanisms have evolved overtime to produce better results and some of these works infamously includes, using Hamming and MAXNET Algorithm with reported accuracy of 95% [15], very well-designed Single-Layer WinnerTake-All Kohonen Mapping approach with reported accuracy of 99.63% [17], using Packet Network Traffic Feature Extraction with false-alarm rate of 0.71% and reported accuracy of 99.29% [18], employing Conditional Random Fields and Layered Approach with false-alarm rate of 17% and accuracy of 98.62% [19] and Kohonen's SelfOrganizing Feature Map (SOM) algorithm with 99.8% recorded accuracy [20].

III. COMMON BLUEPRINT OF OPERATION

Most common intrusion detection mechanisms/methods employed by various programs/systems are as followed:

a. Signature Based Detection [11]: At the most basic level this method employs the monitoring of digital activities like Log Entry analysis and ultimately concludes/detects by searching for similarities in activity patterns. A certain model is equipped which corresponds to a certain threat.

b. Anomaly Based Detection: This method fundamentally employs the analysis using statistics. Specific models are equipped with algorithms using graphs and charts to differentiate between normal and abnormal behavior of the monitored activity (like inflow of data packets). These models are now generally being developed to be artificially intelligent in order to detect abnormal behavior without much human intervention.

c. Stateful Protocol Analysis [21]: This particular technique is employed to monitor the activities and flag these activities on the basis of their protocols being used for request combinations. Further this technique can be enhanced to take necessary action against the flagged request. This is also the method we employ in designing our own intrusion detection system.

IV. CLASSES OF ATTACKS

a. *Denial of Service*: Smurf, Back, teardrop, Neptune, pod, land.
b. *User to Root*: Rootkit, loadmodule, buffer-overflow.
c. *Root to Local*: spy, warezclient, imap, multi-hop, guess-passed, ftp-write.
d. *Probe*: nmap, satan. Portsweep, ipsweep.

V. Taxonomy of IDS Tools

Plethora of mechanisms and designs are fundamentally employed to perfect the art of intrusion detection and prevention, by an extension. Generally, most of these tools are not packed into one, rather separate tools are designed and used in integration. Following figure 1 draws an abstract to help you get a better idea of it.

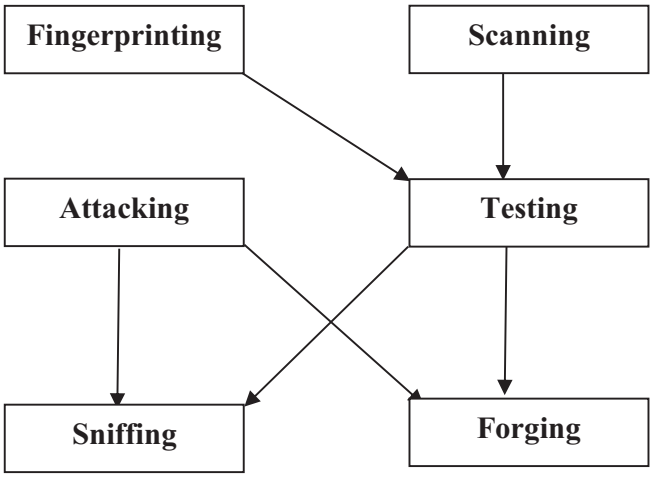

Figure 1 – Types of Intrusion Detection & Prevention Tools.

a. Fingerprinting Tools: Particularly designed to execute predefined unitary tests in order to successfully discriminate an intruding peer.

b. Scanning Tools: Broadly same as fingerprinting tools, except scanning tools perform their predefined unitary tests with certain parameters, which again might vary in a given certain range.

c. Testing Tools; Performs predefined instructions and generally returns an affirmative or non-affirmative (Yes/No) answer. Example: Ping requests.

d. Attacking Tools: In most cases, these tools are employed in preventive measures and they use some property/unexpected values of the protocols.

e. Sniffing Tools: In varied cases sniffing tools primarily capture inflowing or outflowing data packets and possibly, upon instructions, dissects them.

f. Forging Tools: Primarily uses the data packets captures by sniffing tools and forges them in a specified manner in order to corrupt the data. And eventually sends the forged packets on their natural course.

VI. Model of Operation

First, we need to properly look and understand how the data packets look when viewed/captures using a basic "*netstat*" command. Following figure 2 shows the output of the same.

Active Connections

Proto	Local Address	Foreign Address	State
TCP	192.168.1.5:49682	hk2sch130021317:https	ESTABLISHED
TCP	192.168.1.5:49692	13.75.106.[illegible]:https	ESTABLISHED
TCP	192.168.1.5:49706	13.78.93.8: https	CLOSE_WAIT
TCP	192.168.1.5:49710	157.55.130.160:40032	CLOSE_WAIT
TCP	192.168.1.5:49722	91.190.218.53:12350	CLOSE_WAIT
TCP	192.168.1.5:49829	sc-in-f108: imaps	ESTABLISHED
TCP	192.168.1.5:49835	hk2sch130021137: https	ESTABLISHED
TCP	192.168.1.5:50912	sc-in-f109: imaps	ESTABLISHED
TCP	192.168.1.5:50994	del03s07-in-f14: https	ESTABLISHED
TCP	192.168.1.5:51121	a-0011: https	TIME_WAIT
TCP	192.168.1.5:51122	a-0011: https	TIME_WAIT
TCP	192.168.1.5:51124	a-0011: https	TIME_WAIT
TCP	192.168.1.5:51125	a-0011:https	TIME_WAIT
TCP	192.168.1.5:51126	a-0011: https	TIME_WAIT

Figure 2: Output of Netstat Command.

The above figure contains four columns signifying the output of a netstat command run on command prompt of the windows OS. To understand the output, let's break down these columns and understand them separately. The first column easily returns the protocol being used by the connection. In this figure, all connections used the Transfer Control Protocol.

The second column returns the Local Address from where the connection is established from. In this case, it's our own system, therefore, all connections have the source address as our IP address. However, the second part of the local address, after the semicolon, essentially tells us the virtual port number of our machine through which the data goes out/comes in for that particular connection.

The third column shows the foreign address i.e. the non-self system/server to which the connection is established from self (our own system). And the second part of the foreign address column, which is after the semicolon, tells us the protocol used by the foreign/non-self server/system to establish a connection with us. And finally, the last column returns the state of the connection i.e. whether the connection is still in 'connecting' phase or has already been 'established' or is still waiting for it to be established. For example, in the first connection, the parameters are as followed:

```
Protocol: TCP
Local Address: 192.168.1.5
Virtual Port Number: 49682
Foreign Address: hk2sch130021317
Foreign Protocol: https
State: Established
```

At the most fundamental level, motive of this very basic intrusion detection system is to capture the inflowing/outflowing data packets and flag the ones which use suspicious protocols like *ssh, ssl, https, telnet*, *sip, netis, rfb, microsoft-db*– any protocol which can be evidently used for potential cyber-attacks, and further store these captured, and flagged, packets for future review by the respective system/network administrators. Therefore, the algorithm used is portrayed in the following figure 3:

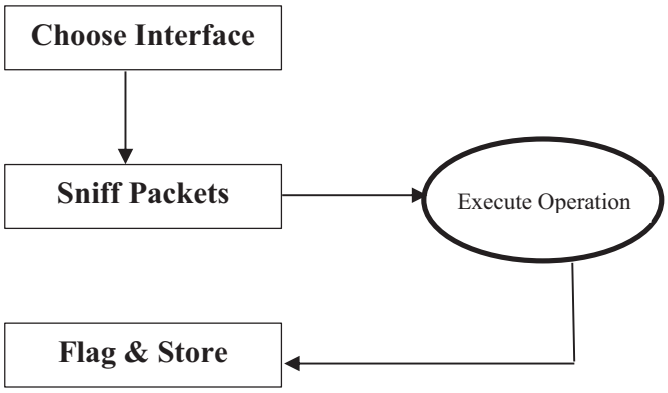

Figure 3: Algorithm of the Program

Step 1: The intrusion detection tool we design will be essentially a sniffer. And since we are using IP/IPv6, we'll be working on layer 3 of the OSI model. Therefore, we'll use the tool Scapy. First import the external files of Scapy in our python script.

```
from scapy.all import *
```

Step 2: We particularly use the inbuilt **sniff ()** function of scapy to sniff the data packets. This function has the following parameters:

a. **Iface** = Interface on which packets are to be sniffed. Leaving this parameter blank, initializes the default interface for use. Example: wlan0
b. **Prn** = Function to be executed upon each captured packet.
c. **lfilter** = Filtering port number/protocol to be applied while capturing packets. Example: "port 80" as a filter will make the sniffer only capture packets going through port 80.
d. **Count =** No. of packets to be captured.
e. **Store** = No. of packets to store in the memory.

We use the following command in order to capture packets on our default interface.

```
sniff (
prn= lambda x: wrpcap ("captured_pkts.pcap",
x, append = True), lfilter = "ssh",
count = 2
)
```

In the above command, the *prn* argument essentially uses the *lambda* function of python to write all the captured packets (in this case 2) filtered for "*ssh*" protocol to a pcap file (.pcap format) "*captured_pkts.pcap*" using the inbuilt scapy function *wrpcap* which particularly signifies - *write to pcap.*

Step 3: In the final step we use the *summary ()* function to read the summaries of all the captured packets stored in the pcap file. Syntax is as followed:

```
for packet in rdpcap("captured_pkts.pcap"):
        print packet.summary()
```

The *rdpcap* () function essentially allows us to read previously stored pcap files.

Step 4: Finally, we flag the captured packets for certain protocols as '*potentially dangerous'* and leave them for system/network administrators to review later.

VII. Drawbacks

1. *Very Basic:* Considering the very output of the program and the institutional level it is designed at the program is not very powerful as compared to pre-existing high ended intrusion detection systems.

2. *Works on Layer 3:* This system particularly works on layer 3 of OSI model since we have to use TCP/IP, IPv6. And therefore, remains inactive or error-full in case executed on layers other than layer 3.

   Resolution: Solution to this problem, is to design *raw sockets* in python using the *socket* library and then operate the program on layer 2.

3. *High CPU Usage:* If the system is executed on layer 3, then in case of capturing large packets, it consumes very high CPU power and leaves minimum power for other running programs on the server/system. Example: Capturing packets when downloading heavy data like ISO files of an operating system.

VIII. Case Study

Initially during the first phase of development the idea of elevating the convincing capability of this paper, valid case study evidence was decided to be added for both small scale deployment and large-scale deployment. Venues/environment of the deployment is to be duly kept into consideration while referring to this case study. The entire network's both in case of small scale and large-scale deployments systems/nodes on the network were first analyzed and judged without any pre-installed intrusion detection systems.

Small Scale Deployment – Gamma Testing

For small scale deployment, a small laboratory of about 10 systems/nodes was finalized and utilized. The network originally having small number of utilized IP addresses was *not sub-networked*. The original project designed in Python 2.7 consisting of a stand-alone executable script was installed remotely on each system and the data packets were initiated for transfer. During the most initial phase of gamma testing, the self-designed executable python script was used to monitor data packets captures from basic netstat commands. And the results we critically fascinating. The program gave a rough accuracy of about 80%, however, with a few tweaking in the program to suit the needs of the environment, the accuracy showed a spike till 88%. Objectives clearly achieved during the gamma testing phase were as followed:

a. Protocol flagging.
b. Manual pattern/signature analysis.
c. Effective prompting of flagged connections/packets.

Large Scale Deployment – Beta Testing

For the purpose of beta testing/large scale deployment, a larger networking laboratory of around 50 systems was utilized. Similarly, as before, original project designed in Python 2.7 consisting of a stand-alone executable script was installed remotely on each system and the data packets were initiated for transfer. Instead of using basic self-designed script to monitor the connections/packets through terminal commands, as compared to Gamma testing phase, more advanced sniffing/monitoring software like Wireshark were brought to use this time. During the initial non-tweaked testing, the self-designed intrusion detection system gave a surprising accuracy of properly flagging packets/connections with suspicious protocols of 76%. However, with certain advancements and tweaking to support the surrounding environment, the accuracy was spiked to 82%. Objectives achieved near the end of a successful Beta testing phase were as followed:

a. Successful protocol flagging at large scale
b. Increased number of filters.
c. Optimized data retention for flagged packets.

Conclusion And Future Works

As explained, this self-designed basic intrusion detection system is an appropriate example to understand the basics of cyber intrusions and their concerned preventive measures. The principle property of the designed system lies in its very operation at the most fundamental and institutional level of operation. And therefore, this concept can be heretofore used as a building block for researches in the similar field of interest/expertise, as evident by the provided case studies, for both Gamma testing and Beta testing. For future, we expect to thoroughly design an advanced neural network based artificially intelligent intrusion detection and prevention system.

References

[1] Carla E. Brodley, Terran lane, Temporal Sequence Learning and Data Reduction for anomaly Detection, Vol. 2, No. 3, pp. 295- 331,August 1999.

[2] "An Intrusion-Detection Model," Denning D., IEEE Transactions on Software Engineering, Vol. SE-13, No. 2, pp.222-232, 1987.

[3] Lin M-J., Ryan J., Miikkulainen R. (1998) "Intrusion Detection with Neural Networks," Advances in Neural Information Processing Systems, Vol. 10, and Cambridge, MA: MIT Press.

[4] Spafford E. H., Kumar S., "An Application of Pattern Matching in Intrusion Detection," Technical Report CSD-TR-94-013. Purdue University, 1994.

[5] "Artificial neural networks for misuse detection," Cannady J., Proceedings of the '98 National Information System Security Conference (NISSC'98), Arlington: Virginia Press, pp. 443-456, 1998.

[6] Huang Hao, Yu Y, "An ensemble approach to intrusion detection based on improved multi-objective genetic algorithm," Journal of Software, Vol.18, No.6, June 2007, pp.1369-1378.

[7] S. J. Stolfo, W.K. Lee, "A data mining framework for building intrusion detection model," Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA: IEEE Computer Society Press, pp. 120-132, 1999.

[8] Jan Platos, Vaclav Snasel, Pavel Kromer, Ajith Abraham," Fuzzy Classification by Evolutionary Algorithms," IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp.313 - 318, 2011.

[9] S. M. Bridges, J. Luo, "Mining fuzzy association rules and fuzzy frequency episodes for intrusion detection," International Journal of Intelligent Systems, pp. 687-703, 2000.

[10] Moon J, Shon T, Seo J, "SVM approach with a genetic algorithm for network intrusion detection," Proceedings of the 20th International Symposium on Computer and Information Sciences (ISCIS 05), Berlin: Springer Verlag, pp. 224-233, 2005.

[11] Youssef Senhaji, Hicham Medromi, Architecture System Team Hassan II University of Casablanca ENSEM Casablanca, Morocco, in their paper on "Network Security: Hybrid IDPS", August 2015.

[12] Shekhar R. Gaddam, Kiran S. Balagani, Vir V. Phoha , "K-Means+ID3: A Novel Method for Supervised Anomaly Detection by Cascading K-Means Clustering and ID3 Decision Tree Learning Methods,"IEEE Transactions On Knowledge And Data Engineering, Vol. 19, No. 3, March 2007.

[13] Mrutyunjaya Panda, Manas Ranjan Patra, Ajith Abraham "A Hybrid Intelligent Approach for Network Intrusion Detection," International Conference on Communication Technology and System Design, Procedia Engineering, vol. 30, pp.1-9,2012.

[14] Iwan Syarif, Adam Prugel-Bennett, Ed Zaluska, Gary Wills, "Application of bagging, boosting and stacking to intrusion detection, "Proceedings of the 8th international conference on Machine Learning and Data Mining in Pattern Recognition,pp.593-602,2012.

[15] Muna M, Monica Mehrotra, Taher Jawhar, "Anomaly Intrusion Detection System using Hamming Network Approach," International Journal of Computer Science & Communication,Vol. 1, No. 1, pp. 165-169, January-June 2010.

[16] Hussein A. Abbass, Kamran Shafi, “Evaluation of an Adaptive Genetic-Based Signature Extraction System for Network Intrusion Detection, “Pattern Analysis and Applications, November 2011.

[17] Qiuming A. Zhu, Suseela T. Sarasamma, Julie Huff “Hierarchical Kohonenen Net for Anomaly Detection in Network Security,” IEEE Transactions on Systems, Man, And Cybernetics-Part B: Cybernetics, Vol. 35, No. 2, April 2005.

[18] Monowar H Bhuyan, Prasanta Gogoi, J K Kalita, D K Bhattacharyya,"Packet and Flow Based Network Intrusion Dataset," Contemporary Computing Communications in Computer and Information Science, vol.306, pp.322-334, 2012.

[19] Baikunth Nath, Kapil Kumar Gupta, Ramamohanarao Kotagiri,"Layered Approach Using Conditional Random Fields for Intrusion Detection,"IEEE Transactions On Dependable And Secure Computing, Vol. 7, No. 1, January-March 2010.

[20] A. Nur Zincir-Heywood, H. Gunes Kayacik, Malcolm I. Heywood,” A Hierarchical SOM based Intrusion Detection System, “Engineering Applications of Artificial Intelligence,Vol.20,no.4, pp.439-451,June 2007.

[21] Indu Kashyap, Lata, “Study and Analysis of Network based Intrusion Detection System” in International Journal of Advanced Research in Computer and Communication Engineering Vol. 2, Issue 5, May 2013.

[22] “Intrusion Detection Systems with Snort”, Rafeeq Ur Rehman, 2003 Pearson Education, Inc. Publishing as Prentice Hall PTR Upper Saddle River, New Jersey 07458 pp 2-3,7-8