

Flow-Based Detection of Unknown Network Attacks

Luís Sacramento

Instituto Superior Técnico

Abstract A telecommunications company like Vodafone Portugal has a complex infrastructure to support your business. This infrastructure contains a multitude of devices and applications. These entities communicate using numerous protocols and generate a huge amount of traffic with complex features. Search malicious or anomalous behavior in the middle of this traffic is a complex task, especially if there is no precise definition of what are these behaviors. The goal of this thesis is to propose an approach to do so and create a tool that implements this approach. The dissertation is based on two basic concepts: flows (flows) and machine learning (machine learning). The concept of flux (flow) was originally defined by Cisco to summarize traffic information is therefore appropriate for this case. A flow is defined in terms of parameters such as the subnet IP packet source, destination subnet to the transport protocol, port of origin and port of destination. Today there are several versions of the concept, such as NetFlow from Cisco, the sFlow (defined by a consortium of companies) and IPFIX (Internet standard). Compare flows therefore appears to be an affordable way to compare complex traffic. On the other hand, unsupervised learning (unsupervised machine learning) allows you to group similar behaviors without having to previously instruct the system about what those behaviors. This approach is therefore a good starting point to discover anomalies without defining what they are. The approach developed will combine these two concepts to group traffic into classes, which then must be analyzed using context information to understand if they are abnormal or not. Examples of context information is information about malicious or send SPAM ASs. A second phase will be set a classifier based on supervised learning (supervised machine learning) to classify the subsequent traffic.

Keywords: Intrusion Detection System, Machine Learning, NetFlow

1 Introduction

Flow definition

IDSs overview

Benefits and challenges

Scope in context

Solution overview

Document structure

2 Related Work

In this section, an overview of the theoretical background is given, and also will be discussed some of the existing major contributions in this field. It will be divided in four subsections:

1. Network Flow Analysis Tools
2. Flow-based Network Monitoring
3. Flow-based Applications in Intrusion Detection Systems
4. Machine-Learning-aided Intrusion Detection Systems

2.1 Network Flow Analysis Tools

As stated previously, network flows allow for a different approach in analysing, monitoring and securing a network. While deep packet inspection allows for *signature-based* approaches, making it easier to detect some kinds of attacks, its not scalable for high speed networks, e.g. 10 Gbit/s [18] - the packets' payloads can't be analysed in real time, for these kinds of speeds. Also, nowadays most packets exchanged have their payload encrypted, making it even more difficult to inspect, even if it was possible to process those many packets in real time.

In 1996, Cisco developed the first flow-based monitoring tool: NetFlow. This consist in a built-in in their routers, and is used to collect and export flow records. As the years went by, new versions arrived and were more and more complete. The latter version - version 9 [2] - also includes security issues, which were not being addressed in the previous ones. **TODO - NAO E NADA DE SECURITY, MAS SIM INTEGRAÇÃO COM OUTROS PROTOCOLOS TAIS COMO MPLS**

So, firstly, as it was previously mentioned, a flow is defined as being an unidirectional sequence of packets passing through an observation point (in this case, an observation point would be a Cisco NetFlow-enable router) that satisfies a set of common features in a given period of time. These features must be defined *a priori*, in order for the device to perform the matching of features.

Being this technology built-in in network devices, it allows to filter all the traffic passing through that device. For example, by deploying this in a border router, all of the traffic going in and out of that network will be filtered by NetFlow. Upon the arriving of an IP packet, the network device looks at that packet's fields in order to find any matching feature with those previously defined. In case the packet's features do match, then an entry is created in the *flow cache* for that flow. Note that a flow may contain several packets, and many different flows can be collected.

As this cache cannot be kept indefinitely, the authors have defined certain policies for its management. And, according to those policies, when of them is satisfied, the entries belonging to that cache will be exported to another device. So, a packet will be exportend when: (i) the end of a flow if detected; (ii) a flow is inactive (i.e. when, for a given timeout, there are no longer packets belonging

to a flow); (iii) belonging to a long-lasting flow, the timeout is reached; (iv) the device is in need of resources, e.g. internal memory.

The device to which the flow records are exported is referred to as the *Collector*, and is physically located in another device. In order to achieve maximum efficiency latency-wise, these records are encapsulated in an UDP datagram. However, it provides also the possibility of exporting the data through SCTP (Stream Control Transport Protocol), if we want to operate in a connection-aware environment.

However, this technology does not have in consideration any security issue - no confidentiality, integrity or authentication is guaranteed. This was designed as such due to efficiency and scalability issues - the deployment of such technology in large networks would not be able to provide real time measures. Instead, it assumed that the exporters (and also collectors) are deployed within a private, restricted and controlled network, rather than a public network, in which anyone could sniff these records, or even forge them.

Apart from NetFlow, many other vendors have their own implementation for flow collection and exporting. Examples are NetStream, Juniper, etc.

Due to the heterogeneous nature of these technologies from each of the vendors existing in this market, the IETF (Internet Engineering Task Force) joined forces to create the first existing standard in flow collection and exportation, thus allowing for the clients to easily deploy their flow-based applications, called IPFIX (IP Flow Information eXport). The aforementioned common properties are referred to as *flow keys*, and these can be, for example a tuple such as:

(IP_source, IP_destination, port_source, port_destination, typeOfService)

In NetFlow, the flow exportation was done by encapsulating the records into UDP or SCTP datagrams. IPFIX provides, in addition to these protocols, the option of exporting flow records through TCP. Also, it fills the security gap in Cisco's implementation: IPFIX provides both (dubio, TODO) confidentiality, integrity, and authentication. These three properties are guaranteed by using secure channels - either through TLS (Transport Layer Security), if the records are exported through TCP, or through DTLS (Datagram Transport Layer Security) if the records are to be exported through UDP or SCTP datagrams. Furthermore, X.509 Certificates are also used, in order to reinforce the latter property.

[13] -> NfSen (network monitoring visualization tool)

[] -> Talk about SiLK (System for Internet-Level Knowledge)

[12] -> include here?

2.2 Flow-based Network Monitoring

Amongst many other applications, such as the monitoring of applications, hosts, security, account and billing, the analysis of network flows has been widely deployed for network monitoring. In this subsection we refer some of the existing work done in this area.

- [12] -> detection of bandwidth misuse -> done reading, TODO
- [10] -> framework for accurate characterization of traffic
- [11] -> network QoS monitoring
- [19] -> Sukhov et al. (2009), based on the analysis of active flows on a network, they managed to develop a system that allows to troubleshoot link problems in the backbone of a network.
- [17] -> effects of DDoS on network flow monitoring applications -> done reading, TODO

2.3 Flow-based Applications in Intrusion Detection Systems

2.3.1 Port Scanning A *port scan* is defined as the act of consistently probing a target hosted, by sending a large amount of generally small packets.

These kinds of attacks can be very easily addressed in the study of network flows, since produce a great amount of flows, when probing the victims. [14] Divides this attack in three distinct categories:

- *Horizontal scan*, in which a single host scans multiple ports in a single machine
- *Vertical scan*, in which a single host scans one single port in multiple machine
- *Block scan*, a combination of the two scans above - a single host scanning multiple ports in multiple machines

Whatever category the attack falls in, this will create an anomaly in the normal network traffic pattern, and many kinds of different flows can be observed.

Most of these attacks are investigated by observing a flow characteristic that registers the most significant difference when compared to normal traffic: the unusually high number of in and outgoing connection in a host, and this is due to the fact that the scans are done by probing many different ports and machine, and therefore generating a lot of new flows.

A type of attack that falls into this category is the SSH attack. This is a particularly interesting attack in the field of study of port scan attacks, as it consists, in a first phase, of performing a port scan on the victim, in order to gain remote access to them. Once the attacker gains access, it can do whatever it wants with the victim, and also to others that might belong to the same network. Therefore, SSH attacks can be potential harmful not only to hosts individually, but also to the network it is connected to.

However, the detection of these attacks, as they rely mainly on *port scan* attacks, can be addressed by performing an analysis of the network traffic at a flow-level. [7] developed a flow-based Intrusion Detection System called *SSHCure*, which allows for real-time detection of brute-force SSH network attacks.

Their solution was based on the observation made by [Hidden Markov Model modeling of SSH brute-force attacks.] and [14]. They observed that the behavior of the attacks over time, in terms of flows, follows a pattern of its evolution, and it can be identified in three distinct phases, as described below:

1. Scanning phase, in which the attacker performs a *port scan* for a certain IP address block, in order to find running SSH daemons in a host (SSH daemons use TCP port 22)
2. Brute-force phase, in which the attackers tries to login to a certain number of hosts, by means of a dictionary attack - various combinations of usernames and passwords
3. Die-off phase, in which after successfully login into the victim host, the traffic volume is drastically reduced, leaving only residual traffic

Based on these three phases, two metric of evaluation were used - *PPFs* (*Packets-per-flow*) and *Minimum number of flow records*.

During the first phase, the first anomalies will start to manifest. As this phase consists in performing scans from one single host, there will be generated many small flows originated from the attacker to the many (or single) targets.

[9] -> by analysing flow-based traffic patterns, are able to identify both network and port scans

2.3.2 Denial of Service A *Denial of Service (DoS)*, is the attempt of an attacker to make a certain server unable to reply, by flooding it with several requests and thus making draining all of its resources.

Despite being an incresing trend in Intrusion Detection Systems, the flow-based approaches are vulnerable to applications that generate a large number of flows, and such is the case of DDoS attacks [17]. (??)

[8] -> detection of novel DoS attacks

[6] -> DoS detection on high-speed networks

[15] -> survey for DoS DDoS detection techniques

2.3.3 Worms A *worm* consists on a harmful software that, unlike the well-know case of a virus (which will be discussed further ahead), has the capability to autonomously explore software vulnerabilities, thus making it capable of replicating itself throughout a network.

This specific attack is usally divided in two distinct phases: (i) a scanning phase, in which the attacker probes several machines in order to find a vulnerability and then proceed to spread the infection; (ii) the transfer phase, in which the attacker proceeds to send the harmful code and infect the victim. As the first phase is a well know case (as described above, in the *port scanning* phase), the discovery of the scan can be crucial in identifying the attacker. Due to the fact that most of the network traffic relies on secure connections, and therefore the payload of travelling packets in most of the times is encrypted (e.g. TCP traffic), a major emphasis is given to the detection of the first phase, as is would be very difficult to detect malicious content of packets in a flow-based analysis.

From using protocol graphs [3], to extending port scan detection approaches, to assigning hosts to sets of classes, many can be the strategies to indentify *worms* in a network, based on the analysis of network flows.

[3] developed a system capable both of detecting Hit-List *worms* and identifying bots, using protocol graphs. A protocol graph is a representation of traffic logs for one specific , in which its vertices (or nodes, as sometimes referred in literature) are representations of IP addresses and the edges are the connections existing between those two entities. This system's detection approach is based on graph size and largest connected component properties. The first is the total number of connected vertices, and the latter is the number of vertices in the largest connected component of the graph, i.e. the vertex with the most connections overall.

In their research, only four protocols were considered:

- **HTTP**, identified by observing a connection in which one of the peers uses port 80 (either destination or source), and represents the majority of the traffic observed
- **SMTP**, identified by observing connections where one of the peers uses port 25, and is the second most active protocol producing traffic in the network
- **FTP**, which can be identified by observing the usage of port 20 by one of the entities in a connection
- **Oracle**, identified when one of the peers is using port 1521. As this protocol requires a login and password to authenticate an user, its expected that users connected to a smaller number of servers

In order to construct the graph, flow records were extracted from Cisco NetFlow-enabled routers, in a large intercontinental network. These records were collected throughout a period of 5 days. As stated by the authors, many *worm* detection systems rely on the detection of an unusual high number of frequent connections between peers, and track them by inspecting connection evidences such as *half-open* TCP connections. So, to avoid this issue, the attackers can use *hit lists*, which consists in a list of previously identified servers. Using this, they need not to contact random servers across the network and can focus on these known ones, making it harder for the systems to detect this scanning phase.

[5] -> grouping traffic in classes, allowing to identify email and scanning worms, without previous knowledge of the scanning strategy.

[4] -> detection of worms by analysing the correlation between network flows and honeypot logs.

2.3.4 Botnets Basically, the problem is solved when the botmaster is tracked. However, the defense against botnets is still in a very early phase, and is still not very well developed.

[22] -> tracking IRC communication channels, that are widely used by botmasters.

Whilst on the detection of scans or DoS, the systems provide real-time detection, in botnets the detection requires a long period of observation to identify a botmaster.

2.4 Machine-Learning-aided Intrusion Detection Systems

A increasing trending in intrusion detection systems is the use of machine learning techniques [16,18]. Machine learning can be defined as a collection of methods that aim to build an intelligent and autonomous system, based on the observation of patterns in a given environment. As expected, such method has been used in an enormous number of different applications, in many different fields of science, such as natural language processing, speech recognition, bioinformatics, spam detection, network intrusion, among many others.

In the field of network intrusion detection, machine learning has been able to classify network traffic, identify anomalous patterns and potentially harmful users. There are mainly two different approaches [Citar o paper Intrusion Detection with Unlabeled Data Using Clustering] when it comes to designing such systems:

- *Misuse-based detection* (also known as *knowledge* or *signature-based detection*), in which signatures (known patterns that correspond to attacks or threats) are compared to captured traffic from the network, in order to detect intrusions, as defined by H.-J. Liao *et al.* [Citar paper Intrusion detection system: A comprehensive review]. The captured traffic will then be labeled as being normal or anomalous, for future training of the algorithm [Citar Intrusion Detection with unlabeled Data Using Clustering]
- *Anomaly-based detection* (or *behavior-based*), in which normal traffic patterns are differentiated from anomalous ones. It focuses its attention on finding patterns that would not be expected from the user's behavior. Unlike *misuse-based* IDSs, these patterns are unknown to the system [16]

2.4.1 Unsupervised Learning [21]

[20]

[1]

One of the first to address unsupervised learning in intrusion detection systems was Eskin et al. (insert date - unknown for now). They present a new technique, which they entitle *unsupervised anomaly detection*. This algorithm allow to train the system with a set a dataset of completely unlabeled data, providing the chance of detecting unknown attacks to the network, which would not be possible when training the system with labeled data - in this case, the system is only able to recognize those labeled intrusions; and also, the manual classification of data can be very hard and tiresome. They used the famous KDD99 dataset for the training of the system. The features were extracted from connection record the raw data gathered throughout the simulated intrusions present in the dataset. This included features such as the basic components of a TCP connection (duration time, protocol type, etc.), some others that were not so trivially obtained (number of file creation operations, number of failed login attempts, etc.), and some other features captured in a small two-second time windows (number of connections to the same host as the current connection in this timespan, percentage of connections that have *SYN* and *REJ* errors, etc.). All summed up, there was a total of 49 features. Also, the dataset was filtered, so that there would only exist a percentage of 1 to 1.5% of attacks vs. 98.5 to 99% normal traffic instances. This is done because of the need of the system to learn to distinguish intrusion instances from the normal ones, and the original dataset was composed mainly of intrusions.

Their solution was based on two assumptions:

1. The number of normal instances greatly outnumbered from the number of intrusions
2. The intrusions themselves are qualitatively different from the normal instances

The system clusters the collected data through an algorithm that computes a distance-based metric. However, because of the different distributions that each feature vector may have, these have to be normalized in order to apply the same metric to all of the vectors. After computing the clustering algorithm, these new clusters can now be classified as being normal traffic instances or an intrusion. The first assumption implies that small clusters correspond to the intrusion instances, as opposed to the bigger clusters that represent normal instances; the second assumption implies that normal and intrusion instances will not be under the same clusters because of their qualitative differences.

In order to measure the performance of the system, used the following metrics:

- *Detection rate*, which represents that ratio of intrusions detected by the system, by the intrusions present in the dataset
- *False positives*, which is the ratio of the total number of intrusions that were incorrectly detected by the system, by the total number of normal instances

In this solution, there is an inevitable *tradeoff* between these two indicators, as one scales with the other. However, they managed to obtain a fairly reasonable ratio of *detection rate* and *false positives*.

This paper presents mainly two advantages to the traditional intrusion detection systems. The first, is that it does not require any kind of manual classification, and the second is that the system is capable of detect intrusion that were previously unknown.

3 Architecture

4 Evaluation

5 Conclusion and Outlook

References

1. Casas, P., Mazel, J., Owezarski, P.: Unsupervised Network Intrusion Detection Systems: Detecting the Unknown without Knowledge. *Computer Communications* 35(7), 772–783 (2012), <http://dx.doi.org/10.1016/j.comcom.2012.01.016>
2. Claise, B.: Cisco Systems NetFlow Services Export Version 9 Status. The Internet Society (2004)
3. Collins, Reiter, M.: Hit-List Worm Detection and Bot Identification in Large Networks Using Protocol Graphs. *Raid* 4637, 276–295 (2007), http://dx.doi.org/10.1007/978-3-540-74320-0_{_}15
4. Dressler, F., Jaegers, W., German, R.: Flow-Based Worm Detection Using Correlated Honeypot Logs. *Communication in Distributed Systems (KiVS)*, 2007 ITG-GI Conference pp. 1–6 (2007)
5. Dubendorfer, T., Plattner, B.: Host Behaviour Based Early Detection of Worm Outbreaks in Internet Backbones. *14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE'05)* pp. 166–171 (2005), <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1566204>
6. Gao, Y., Li, Z., Chen, Y.: A DoS Resilient Flow-Level Intrusion Detection Approach for High-Speed Networks. *IEEE International Conference on Distributed Computing Systems* pp. 39–39 (2006), http://ieeexplore.ieee.org/xpls/abs_{_}all.jsp?arnumber=1648826
7. Hellemons, L., Hendriks, L., Hofstede, R., Sperotto, A., Sadre, R., Pras, A.: SSHCure: A flow-based SSH intrusion detection system. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7279 LNCS, 86–97 (2012)
8. Ke-xin, Y., Jian-qi, Z.: A Novel DoS Detection Mechanism. *Mechatronic Science, Electric Engineering and Computer (MEC)*, 2011 International Conference on pp. 296–298 (2011)
9. Kim, M.S.K.M.S., Kong, H.J.K.H.J., Hong, S.C.H.S.C., Chung, S.H.C.S.H., Hong, J.: A Flow-based Method For Abnormal Network Traffic Detection. *2004 IEEE/IFIP Network Operations and Management Symposium (IEEE Cat. No.04CH37507)* 1, 1–14 (2004)

10. Kundu, S.R., Pal, S., Basu, K., Das, S.K.: An architectural framework for accurate characterization of network traffic. *Parallel and Distributed Systems, IEEE Transactions on* 20(1), 111–123 (2009)
11. Li, Y.: Study of the monitoring model for securities trading network quality of service. In: *Information Science and Engineering (ICISE), 2010 2nd International Conference on*. pp. 1–4. IEEE (2010)
12. Mansmann, F., Fischer, F., Keim, D.A., Pietzko, S., Waldvogel, M.: Interactive analysis of netflows for misuse detection in large ip networks (2009)
13. Minarik, P.: Netflow data visualization based on graphs. *Visualization for Computer Security, Lecture Notes in Computer Science* 5210, 144–151 (2008), <http://www.springerlink.com/index/8X551377N4243026.pdf>
14. Northcutt, S.:
15. Peng, T., Leckie, C., Ramamohanarao, K.: Survey of network-based defense mechanisms countering the DoS and DDoS problems. *ACM Computing Surveys* 39(1), 3–es (2007), <http://portal.acm.org/citation.cfm?doid=1216370.1216373>
16. Pitts, F., Press, A.: A Survey Of Network Flow Applications 36, 178–189 (2014)
17. Sadre, R., Sperotto, a., Pras, a.: The Effects of DDoS Attacks on Flow Monitoring Applications. *IEEE Network Operations and Management Symposium (NOMS'12)* pp. 269–277 (2012)
18. Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A., Stiller, B.: An Overview Of IP Flow-Based Intrusion Detection. *IEEE Communications Surveys and Tutorials* 12(3), 343–356 (2010)
19. Sukhov, A.M., Sidelnikov, D.I., Galtsev, A., Platonov, A.P., Strizhov, M.V.: Active flows in diagnostic of troubleshooting on backbone links. *arXiv preprint arXiv:0911.2619* (2009)
20. Winter, P., Hermann, E., Zeilinger, M.: Inductive Intrusion Detection in Flow-Based Network Data Using One-Class Support Vector Machines. *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on* pp. 1–5 (2011)
21. Xiang, J., Westerlund, M., Sovilj, D., Pulkkis, G.: Using Extreme Learning Machine for Intrusion Detection in a Big Data Environment. *AISeC'14* pp. 73–82 (2014)
22. Zeng, Y., Hu, X., Shin, K.G.: Detection of Botnets Using Combined Host-and Network-Level Information. *IEEE/IFIP International Conference on Dependable Systems and Networks* pp. 291–300 (2010), papers3://publication/uuid/5BB744AC-A55A-40A7-80F3-170B64CAE01F