Refinements In Zeek Intrusion Detection System

Asheesh Tiwari
Department of CEA,
GLA University,
Mathura, India
asheesh.tiwari@gla.ac.in

Shivam Saraswat
Department of CEA,
GLA University,
Mathura, India
shivam.saraswat_csf18@gla.ac.in

Utkarsh Dixit
Department of CEA,
GLA University,
Mathura, India
utkarsh.dixit_csf18@gla.ac.in

Sagar Pandey
Department of CEA,
GLA University,
Mathura, India
sagar.pandey_csf18@gla.ac.in

2022 8th International Conference on Advanced Computing and Communication Systems (ICACCS) | 978-1-6654-0816-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICACCS54159.2022.9785047

***Abstract*—Introduced as a free and open-source network analysis framework and designed to function as a network security monitor (NSM), Zeek can also be used as a network intrusion detection system (NIDS) when combined with additional live network events analysis. But it lacks some features that should have come by default built-in. So to fill this gap, some features have been added to it, so that Zeek IDS becomes more useful to system administrators. This tool is being used by Fortune 500 companies, universities, and governments, and it helps them detect malicious activities. So they can prevent them from suffering a major loss that could have been caused by malicious activities. In this paper, we have troubleshot and resolved some of the problems with the Zeek network monitoring tool. Zeek's logging capability is enhanced by adding the City and Country name of IPs to the logs, filtering the noise of a particular domain, filtering the noise of a particular port, and separating Local and Remote Connection Logs into separate files.**

Keywords—Network Security Monitor (NSM), Zeek,Hybrid IDS, IPS, Network Intrusion Detection System (NIDS), Logging

I. INTRODUCTION

In most organizations' networks, infiltration happens often. Hackers and crackers can infiltrate computer networks; these individuals may employ computer viruses, spam, and denial of service to disrupt network operations. According to [1], all of these intrusions might happen from either inside or outside the organization's network. One of the most difficult approaches for ensuring the security of information systems and networks is the deployment of Intrusion Detection Systems (IDSs).

Detection of such attacks, known as network intrusion detection, is a relatively recenttopic of security research. These systems are classified into two types: those that run "standalone" by directly and passively monitoring network traffic using a packet filter, and those that rely on audit information obtained from the hosts on the network they are attempting to safeguard. Furthermore, hybrid systems that mix these two techniques are becoming increasingly popular.

In this paper, we have focused on adding some features to the Zeek IDS that enhance the logging activities of Zeek IDS. Though monitors contain less information than systems with access to audit trails, they have the important benefit of being able to be introduced to a network without needing any modifications to the hosts. This benefit is huge for our uses — monitoring a pool of several thousand different, diversely regulated hosts.

II. INTRUSION DETECTION SYSTEM (IDS)

An intrusion detection system (IDS) is a software or a physical appliance that monitors network traffic to detect undesired behavior such as unlawful and malicious traffic, traffic that breaches security rules, and traffic that violates authorized usage norms [2].

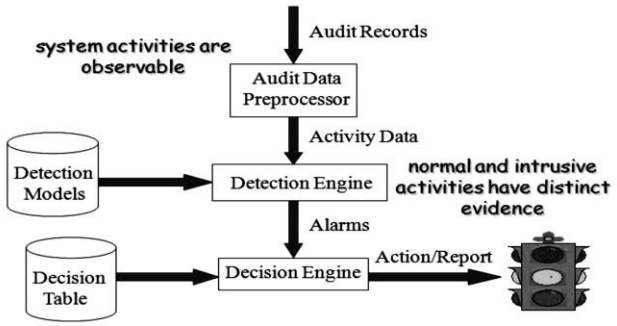

Fig 1: Activities in IDS

A. Detection Techniques

There are two types of threat detection ways: signature-based and anomaly-based detection.

i. Signature-based IDS

A signature-based intrusion detection system (IDS) searches for rules or patterns of recognized harmful traffic. When a signature match is identified, your administrator receives an alert. These warnings can identify known malware, server attacks, and network scanning.

978-1-6654-0816-5/22/$31.00 ©2022 IEEE

ii. Anomaly-based IDS

An anomaly-based intrusion detection system (IDS) detects computer intrusions by monitoring system activity and classifying it as normal or odd. It consists of a statistical model of ordinary network traffic, which includes the bandwidth used, the protocols used for the traffic, and the network's ports and devices. It analyses network traffic in real-time and compares it to the statistical model. The administrator is notified in the event of any abnormality or inconsistency. One advantage of this method is that it can identify fresh and unique threats.

B. Types of Alarms Generated by Intrusion Detection Systems

Alarms generated by IDSs fall into four categories: *true-positive, true-negative, false-positive,* and *false-negative*. To begin, a *true positive* happens whenever the detection engine of an intrusion detection system generates a warning based on the correct detection of a potential threat. The *true negative* is when a detection engine doesn't raise an alarm for normal traffic despite the possibility of danger, and this occurs during a normal network traffic flow. In the case of a *false positive*, a detection engine generates an alert for a non-harmful occurrence, leading to a false alarm. The fourth, and possibly most dangerous, condition is a *false negative*, which occurs when a detection engine fails to alert or identify something, allowing it to enter the network undetected.

As a result, IDSs may mistake normal activity as harmful, resulting in a *false positive (FP)*, or harmful activity as normal, resulting in a *false negative (FN)*. Many security issues can arise as a result of false positives and false negatives; for example, false negatives might result in unlawful or anomalous activity on the Internet or in computer systems/information systems. Furthermore, when false positives are numerous, they can easily obscure serious threats and so overload the IDSs that are in charge of computer system security. It should be emphasized that when actual assaults occur, true positives (real alarms) are deeply buried inside false positives, making them simple to ignore by the security operator [3].

III. ZEEK IDS

In 1995, Vern Paxson began work on Bro IDS, the dominant open-source NSE-based solution. Bro was designed for size, customization, and performance as a Linux/BSD-based software package. Because of an active open-source community, the support of Corelight (a premium product and support firm founded by Paxson and other prominent contributors), and large clients who use it, its capabilities have continued to improve and expand. In 2018, Bro was renamed, Zeek.

Zeek is a hybrid (signature and anomaly-based) intrusion detection system. Its analysis engine will convert collected traffic into a series of events. The Policy Script Interpreter is the system's powerhouse. It has its language (Bro-Script) and can perform a wide range of incredibly powerful and diverse operations.

Every packet mirrored to it from a sector of interest is simply parsed by the architecture. Zeek then uses many simultaneous event engines to create a wide range of events to indicate the presence of any discernible form of transaction. Regardless of who is involved or what the objective is, a file transfer will produce an event that will send all the necessary data about the transaction. These events provide the context for the logs that Zeek creates across a wide range of protocols and data item kinds. These same events are also valuable in the Zeek architecture's Event Scripting tier, where operators may utilize them and the information they pass to trigger signatures or handoff data to scripts for correlation, alerting, or external integration. The operators can endow Zeek's policy-neutral capabilities with the ability to watch and act depending on the site's scripted policy under this Event Scripting layer.

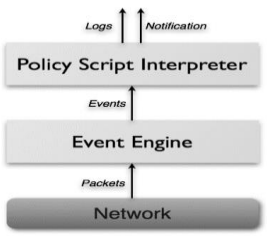

Fig 2: Zeek's Internal Architecture

Zeek's basic architecture also enables it to scale effectively to large environments, thanks to a management daemon named zeekctl, which handles many "workers." Each worker is in charge of the event engines, which process events and parse flows given to them by a load-balancing front-end solution (for example, PF_RING or cPacket). Their logging and events can be sent to a centralized Manager or Logger process for correlation, consolidated analytics, and reporting [4].

Zeek is divided into two key components in terms of architecture. The system's event engine (or core) divides the incoming packet stream into a set of higher-level events. These events represent network activity in a policy-agnostic manner, stating what was seen but not why or whether it is relevant. Every HTTP request, for example, creates a corresponding *http_request* event, which specifies the IP addresses and ports involved, the URI getting requested, and the HTTP version being used. However, the event does not disclose any other information, such as if the URI is associated with a known malware site [5].

IV. LITERATURE REVIEW OVERVIEW ON ZEEK IDS

Johnson, K. described Snort as a signature-based IDS. Because of the way the Snort engine is designed, a single rule may be applied to several network protocols. Protocol and

content matching analysis are performed by the Snort program.It is also often used to actively thwart or passively detect a wide range of attacks, as well as to effectively conduct probes such as buffer overflows, stealth port scans, web application attacks, and operating system fingerprinting attempts. The Snort application is primarily used to prevent intrusions by controlling attacks as they happen [6].

Some important features of the Snort software include its ability to run on any operating system, examine protocols, examine packet states, and reassemble packets. Snort is a single-threaded program, which means it can only use one CPU core at a time. Furthermore, Snort includes a graphical user interface that displays numerous components for examining data. Snort rules are simple to specify by basic users, yet powerful enough to identify a wide range of hostile or suspicious network activity. Snort can do one of three basic actions when a packet fits a defined rule pattern: log, alarm, or pass.

Snort IDS necessitates the use of an external data packet-sniffing library. Stacks are frequently in charge of re-gathering packets and enabling application visibility into the bundle's payload in network operating systems. A packet-sniffer will expect to have access to raw packets that have not been changed to differentiate them and will be able to detect the usual activities. Snort is most often used with Libpcap as its packet-sniffing library. Snort IDS is particularly adaptable since Libpcap works across a broad range of operating systems.

Amira described CPU utilization as a significant component in determining IDS/IPS performance. Packet dropping happens when the number of packets exceeds the processor's capability and resources. In addition, the number of possible rules for an IDS raises the number of processing jobs required [7].

OISF described Suricata as a signature-based IDS, similar to Snort. Suricata, a signature-based intrusion detection system (IDS), is an open-source network threat detection engine that is quick and robust. It can identify intrusions in real-time, monitor network security, and analyze offline packet capture (pcap). Suricata inspects network traffic and detects complex threats by employing powerful and broad rules and signature language.

Suricata is multi-threaded, so it may run numerous threads by utilizing all the CPU cores available on a computer. Suricata can capture and log not just packets but also Transport Layer Security/Secure Sockets Layer (TLS/SSL) certificates, HTTP requests, and DNS queries. As network capacity rises, multi-threading as an advanced feature of the Suricata detection engine is required (Nielsen, 2010). Suricata should be able to handle traffic at speeds ranging from 100 to 200 Mbps before hitting the limit of one CPU and dropping packets to balance. Suricata was built from the start to employ multi-processors, according to the Open Information Security Foundation (OISF). The Suricata intrusion-detection system features many threads inside the same detection engine, allowing it to partition processing duties and harness processing activity among various threads for the same detection engine [8].

Moore, G., proposed that the processing capability of a CPU would double every eighteen months when compared to single-core systems. Multi-threaded processing can benefit from such a forecast [9].

Mehra, P. described Zeek as a Hybrid IDS. Zeek, as an IDS, features an analysis engine that turns collected traffic into a sequence of events. The events mentioned may be a user logon to FTP, a connection to a webpage, or any other online action by a user. As an open-source IDS, it supports Linux, FreeBSD, and macOS platforms. It is a network-based intrusion detection system that looks for unusual activities in network traffic. Zeek IDS identifies intrusions by monitoring network traffic to extract its application-level structure, then running event-oriented analyzers to match network activity with threat-related patterns.Zeek's detection engine can be told to produce a log entry, warn the operator in real-time, perform an operating system instruction that can result in the termination of a connection, or prevent malicious host activity if it detects any suspicious behavior. Forensics can benefit greatly by providing extensive log data.

Zeek IDS is capable of detecting intrusions at high speeds (Gigabits per second (Gbps)) and in large volumes of traffic. This intrusion detection system employs the packet-filtering concept, allowing it to achieve efficiency while operating on commercially accessible PC hardware, allowing it to act as a cost-effective way of monitoring a site's Internet connection. Multiple analyzers are included in Zeek IDS, including protocol decoders for a variety of network protocols and a signature matching engine that interacts with network events. It includes a scripting language that allows users to construct event handlers in their environment-specific policies [10].

Instead of discrete processing engines, processors, and decoders, this IDS uses a script interpreter. It makes use of datapackets obtained from the network through common packet collection libraries such as libpcap. Its engine, in addition to managing large throughput, provides clustering options for extremely high-throughput situations, allowing it to deal with huge-volume data packets. Zeek IDS can also handle multi-threading operations, albeit it is not an essential component of it [11].

According to Paxson, V., Zeek later included file extraction and correlation features similar to those found in Suricata [12]. There is file hash extraction support and correlation with this approach, allowing for automatic file extraction and alerts utilizing bespoke file hashes or publicly available hash data sources.

There is a need to focus on increasing per-core processing efficiency, which will result in increased efficiency in clustered or dispersed networks to match the consistent rise in network throughput [13].

Using Zeek IDS with its scripting possibilities makes managing threats, logging, and even after-detection tasks easier. Snort and Suricata, with their in-line method, provide options for controlling traffic by banning those that match signature criteria. Zeek not only logs and blocks non-matching signatures but also offers alternatives for delivering email messages and automatically truncates network connections. Zeek's scripting policy gives special solutions to cater to rate-limit flows by correlating specified policies as well as the operating system and application response [14].

V. PROPOSED WORK

The Zeek IDS is a really good Hybrid IDS to work with. But it lacks some features that should have come by default built-in. So to fill this gap, we added some features to it, so that Zeek becomes more useful to system administrators.

These features have been added using Bro Script by Zeek. We added these codes to the *zeek/scripts/policy/custom-scripts* directory of Zeek IDS.

We added City and Country names to the logs, filtered noise of a particular domain, filtered noise of a particular port, and separated Local and Remote Connection Logs in separate files.

After adding these codes, we had to restart the Zeek to test it out.

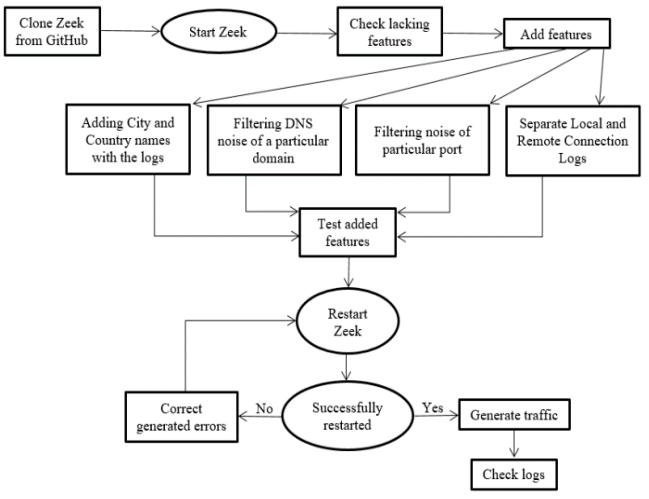

Fig 3: Process Flow of Refinements in Zeek IDS

A. Result Analysis

1. Adding City and Country name to each connection record.

The first feature was to add City and Country names to each connection record of IP addresses.

By default, Zeek doesn't make logs with City and Country name as shown below (showing example with *104.112.106.143*). For this, we added the following code –

```
redef record Conn::Info += {
    orig_cc: string &log &optional;
    country: string &log &optional;
    region: string &log &optional;
};

event connection_state_remove(c: connection)
    {
    local orig_loc = lookup_location(c$id$orig_h);
    if ( orig_loc?$country_code )
        c$conn$orig_cc = orig_loc$country_code;

    local resp_loc = lookup_location(c$id$resp_h);
    if ( resp_loc?$country_code )
        c$conn$country = resp_loc$country_code;
        c$conn$region = resp_loc$city;
    }
```

Fig 4: Code

Restarted Zeek andgeneratedsome network traffic.Then checked the logs to confirm that logs now contain City and Country names.

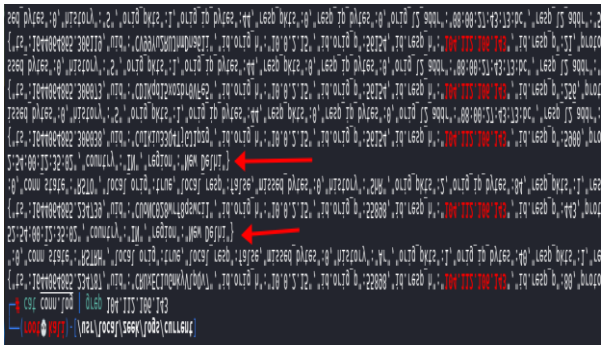

Fig 5: Checking connection logs

2. Filtering DNS noise of a particular domain.

The second feature was to filter the DNS noise of a particular domain.

By default, Zeek makes logs of every DNS request in the *dns.log* file. We made Zeek to filter DNS requests of a particular domain (showing an example with *cybersapien.tech*). For this, we added the following code –

```
module LogFilter;

event zeek_init()
{
    Log::remove_default_filter(DNS::LOG);
    Log::add_filter(DNS::LOG, [$name = "dns-noise",
                $path_func(id: Log::ID, path: string, rec: DNS::Info) = {
                    return (rec?$query && /.mozilla.(org|net|com)$|.github.(io|com)$|cybersapien.tech$/ in rec$query) ? "dns-noise" : "dns";
                }]);
}
```

Figure 6: Code

Restarted Zeek anda DNS requestwas madeusing ping.

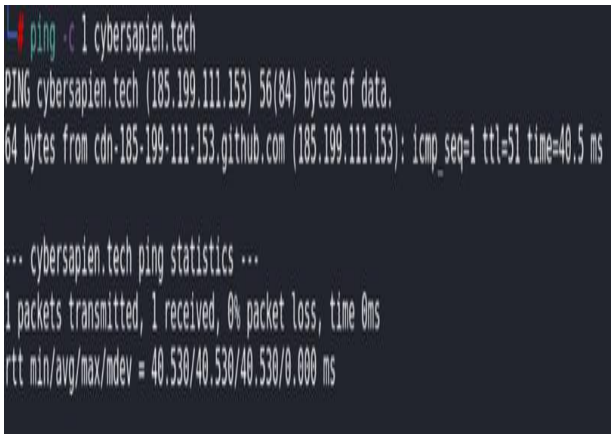

Fig 7: Ping request

Now, cybersapien.tech is captured as DNS noise–

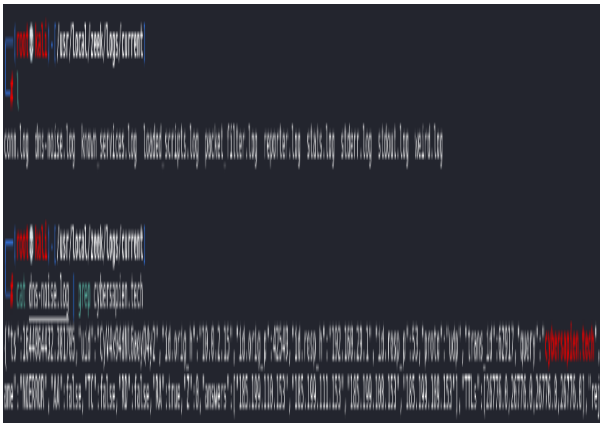

Fig 8: Checking DNS-noise logs

3. Filtering noise of particular port.

The third feature was to filter the noise of a particular port.

By default, Zeek makes logs of every port request. We made Zeek to filter requests of a particular port (for example, port 80).So to filter it from logs, we added the following code –

```
module LogFilter;

const ignore_ports_resp: set[port] = {80/udp, 80/tcp, 123/udp, 137/udp, 161/udp, 514/udp, 514/tcp, 5355/udp, 5666/tcp, 8443/tcp} &redef;
const ignore_services: set[string] = {"dns"} &redef;

event zeek_init()
{
        Log::remove_default_filter(Conn::LOG);
        Log::add_filter(Conn::LOG, [$name = "conn-noise",
                        $path_func(id: Log::ID, path: string, rec: Conn::Info) = {

                                return (rec$id$resp_p in ignore_ports_resp) ? "conn-noise" : "conn";
                        }]);
}
```

Fig 9: Code

In the above code, we are filtering ports *80, 123, 137, 161, 514, 5355, 5666,* and *8443*.

Restarted Zeek and some network traffic is generated using Nmap.

Now, we can see that logs for the above ports are filtered as connection noise –

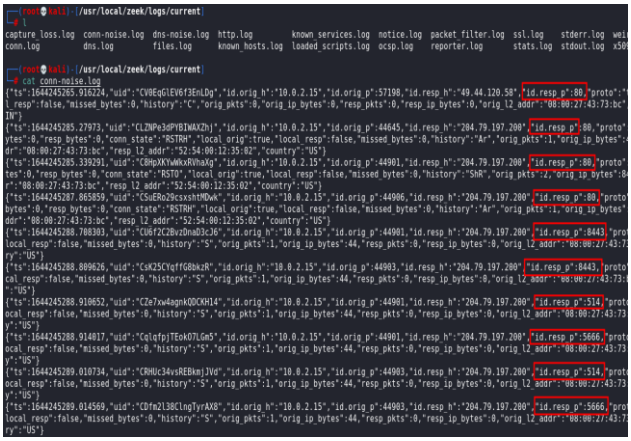

Fig 10: Checking connection-noise logs

4. Separate Local and Remote Connection Logs.

The fourth feature was to separate the Local and Remote Connection Logs.

By default, Zeek makes connection logs in the same file (*conn.log*). We made Zeek make connection logs in two separate files (*conn-local.log* and *conn-remote.log*) along with *conn.log*.

For doing this, we added the following code –

```
function myfunc(id: Log::ID, path: string, rec: Conn::Info) : string
    {
        # Return "conn-local" if originator is a local IP, otherwise
        # return "conn-remote".
        local r = Site::is_local_addr(rec$id$orig_h) ? "local" : "remote";
        return fmt("%s-%s", path, r);
    }

event zeek_init()
    {
    local filter: Log::Filter = [$name="conn-split",
            $path_func=myfunc, $include=set("ts", "id.orig_h")];
    Log::add_filter(Conn::LOG, filter);
    }
```

Fig 11: Code

Restarted Zeek and after this, some network traffic is generated with the help of Nmap.

Now, we can see two more connection log files are generated (local and remote).

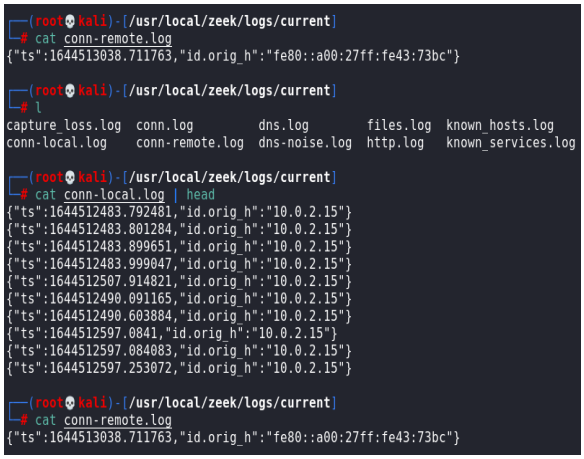

Figure 12: Separate files are created

CONCLUSIONS

In this paper, we compared Zeek IDS with other IDS available on the market. With this, we got to know that Zeek is a Hybrid IDS, which means it can do detection based on signature as well as an anomaly. After this, we checked where Zeek was lagging, and according to that, we added some of the features to the Zeek IDS. For adding features, we used Bro Script (Zeek's scripting language). Then we tested out the added features. With these features, Zeek can be more useful to the administrators using it.

REFERENCES

[1]. Ross, S. (2007). IS Security Matters. Information Systems Control Journal, 6, 14-19.

[2]. Liao, H. J., Lin, C. H. R., Lin, Y. C., & Tung, K. Y. (2013). Intrusion detection system: A comprehensive review. Journal of Network and Computer Applications, 36(1), 16-24.

[3]. Sourour, M., Adel, B. & Tarek, A. (2009). Environmental awareness intrusion detection and prevention system toward reducing false positives and false negatives. In Proceedings of IEEE Symposium on Computational Intelligence in Cyber Security (CICS '09) (pp. 107 –114). Oslo: IEEE.

[4]. Mcphee, M. (2020). Methods to Employ Zeek in Detecting MITRE ATT&CK Techniques.

[5]. Zeek Docs -https://docs.zeek.org/en/v3.0.14/intro/index.html

[6]. Johnson, K. (2008). Basic Analysis and Security Engine, Tech. Report.

[7]. Amira, S.A., Salama, M., Hassanien, A.E., Hanafi, S.E. & Tolba, M.F. (2013). "Multi-layer hybrid machine learning techniques for anomalies detection and classification approach". In. 13th International Conference on Hybrid Intelligent Systems (HIS) (pp. 215-220). IEEE.

[8]. Open Information Security Foundation (OISF). (2011, 04 05). www.openinfosecfoundation.org. Retrieved 12 10, 2017, from OISF Foundation website: https://www.openinfosecfoundation

[9]. Moore, G. (1965). Cramming more components onto integrated circuits. IEEE, 11 (3), 114-117.

[10]. Zeek-ids (2008). Zeek-ids Technical Report. California: International Computer Science Institute.

[11]. Mehra, P. (2012). A brief study and comparison of Snort and Bro Open Source Network Intrusion Detection System. International Journal of Advanced Research in Computer and Communication Engineering, 1 (6), 384-389.

[12]. Paxson, V. (1999). "Bro: a system for detecting network intruders in real-time,". Journal of Computer Networks, 31, 2435-2463.

[13]. Zeek-ids (2014). http://www.zeek.org. Retrieved 8 22, 2017, from zeek.org: http://www.zeek.org

[14]. Gerber, J. (2010, 08 26). http://blog.securitymonks.com. Retrieved 12 14, 2017, from securitymonks.com: http://www.securitymonks.com