RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

A NEW APPROACH TO DETECT P2P TRAFFIC BASED ON SIGNATURES ANALYSIS

UMA NOVA ABORDAGEM PARA DETECTAR TRÁFEGO P2P COM BASE NA ANÁLISE DE ASSINATURAS

UN NUEVO ENFOQUE PARA DETECTAR EL TRÁFICO P2P BASADO EN EL ANÁLISIS DE FIRMAS

Ammar Mazri[1], Merouane Mehdi[1]

e534994
https://doi.org/10.47820/recima21.v5i3.4994

RECEIVED: 01/18/2024 APPROVED: 02/18/2024 PUBLISHED: 03/04/2024

ABSTRACT
In recent years, peer-to-peer (P2P) networks have gained more popularity in the form of file-sharing applications, such as uTorrent and eMule, that use BitTorrent and eDonkey protocols. With such popularity comes security risks and external attacks; the latter is often associated with information hacking. In this paper, we will introduce a new way to monitor and detect the use of each of the P2P applications within the corporate network. Based on the inspection of traffic packets in order to extract digital signatures of these applications using the open-source packet analysis program "Wireshark," in addition to using the well-known Snort intrusion detection system (IDS) with a number of adequate and new rules, this solution can allow us to receive powerful warning messages that detect the presence of P2P applications inside the network. We implemented our rules in Snort IDS. Over a period of time, this solution allowed us to achieve 96% effectiveness in detecting the presence of P2P applications.

KEYWORDS: Peer-to-Peer (P2P). Digital signatures E-Donkey. µtorrent. EMule. BitTorrent. Snort IDS.

RESUMO
Nos últimos anos, as redes *peer-to-peer* (P2P) ganharam mais popularidade na forma de aplicativos de compartilhamento de arquivos, como uTorrent e eMule, que usam protocolos BitTorrent e eDonkey. Com essa popularidade vem os riscos de segurança e ataques externos; o último é frequentemente associado a *hackers* de informações. Neste artigo, apresentaremos uma nova maneira de monitorar e detectar o uso de cada um dos aplicativos P2P dentro da rede corporativa. Com base na inspeção de pacotes de tráfego para extrair assinaturas digitais desses aplicativos usando o programa de análise de pacotes de código aberto *"Wireshark",* além de usar o conhecido sistema de detecção de intrusão Snort (IDS) com várias regras adequadas e novas, esta solução pode nos permitir receber mensagens de aviso poderosas que detectam a presença de aplicativos P2P dentro da rede. Implementamos nossas regras no Snort IDS. Ao longo de um período de tempo, essa solução nos permitiu alcançar 96% de eficácia na detecção da presença de aplicações P2P.

PALAVRAS-CHAVE: Peer-to-Peer (P2P). Assinaturas digitais E-Donkey. µtorrent. Emule. BitTorrent. Snort IDS.

RESUMEN
En los últimos años, las redes peer-to-peer (P2P) han ganado más popularidad en forma de aplicaciones para compartir archivos, como uTorrent y eMule, que utilizan protocolos BitTorrent y eDonkey. Con tal popularidad vienen los riesgos de seguridad y los ataques externos; este último se asocia a menudo con la piratería de información. En este artículo, presentaremos una nueva forma de monitorear y detectar el uso de cada una de las aplicaciones P2P dentro de la red corporativa. Basado en la inspección de paquetes de tráfico para extraer firmas digitales de estas aplicaciones usando el programa de análisis de paquetes de código abierto "Wireshark", además de usar el conocido sistema de detección de intrusión Snort (IDS) con una serie de reglas adecuadas y nuevas, esta solución puede permitirnos recibir poderosos mensajes de advertencia que detectan la presencia de aplicaciones P2P dentro de la red. Implementamos nuestras reglas en Snort IDS. Durante un

[1] DIC, Laboratory, Electronics Department, University Blida, Algeria.

período de tiempo, esta solución nos permitió lograr una efectividad del 96% en la detección de la presencia de aplicaciones P2P.

PALABRAS CLAVE: Peer-to-Peer (P2P). Firmas digitales E-Donkey. µtorrent. EMule. BitTorrent. Snort IDS.

INTRODUCTION

Over the past few years, P2P applications have gained immense popularity in the online realm. These applications allow users to share files and exchange data with other P2P users worldwide. However, this widespread accessibility also introduces potential risks, as it exposes data not only to legitimate users but also to potential intruders. Among the well-known P2P applications widely used today are µTorrent and eMule. Due to their nature as file transfer tools, these applications are susceptible to exploits that can result in the exposure of sensitive data, network overload, and the distribution of malicious software like spyware, bots, and viruses. Malicious actors with nefarious intent can exploit vulnerabilities in the protocols used by P2P applications to target peer-to-peer networks. The objective of this thesis is to devise an effective strategy for detecting the usage of P2P applications within a network and mitigating the associated risk factors involved in their utilization.

In recent years, P2P applications have become very popular in the internet world. Anyone can install a P2P application, which allows us sharing files and exchange data with all other users (P2P) around the world. This process makes the data available to others as well as intruders. Examples of some well-known applications for P2P used in our time that have great popularity there are µtorrent and emule, since these applications are transfer tools, they are vulnerable to exploits that involve exposure of sensitive data, network overloading, and the distribution of malware that includes spyware, bots, and viruses. Criminals with malicious intentions can attack peer-to-peer networks by exploiting vulnerabilities in the protocols used by P2P applications in the network. Our goal of this thesis is to develop an effective plan to detect the presence of using P2P applications within the network and reduce the risk factors involved in the use of these P2P applications.

For this purpose, in order to monitor the network and detect any uses of P2P applications and minimize the risks generated by them. In our work, we will depend on the intrusion detection system "Snort" and a deep analytical study of P2P protocols. The work is divided into the following steps:

Step 1: Deep analysis of traffic resulting from the use of P2P applications by the analyzer program "Wireshark". Wireshark is used for traffic analysis. This type of tool tries to capture network packets and tries to display this belt data in as much detail as possible [1].

Step 2: Extract signatures of protocols P2P (BitTorrent and E-donkey)

Step 3: Implantation the extracted digital signatures of the P2P protocols into the intrusion detection system "Snort".

Step 4: Discuss the results of monitoring the recorded network traffic to confirm the effectiveness of this work.

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

BACKGROUND AND RELATED WORK

1 BACKGROUND

In this section we provide some background on peer-to-peer networks and describe some applications of P2P. Peer-to-peer networks (P2P) are a type of decentralized network architecture that allows users to share files between them without going through a server. There are two types of P2P architecture: unstructured and structured [2]. In a structured system, peers are organized to search other peers more efficiently, but in an unstructured system, peers are randomly connected to certain other peer subsets [3]. There are three models of unstructured P2P network architecture [4]. P2P applications have become attracting millions of users after their appearance and are very popular in the Internet World. However, Peer-to-peer applications also introduce security risks that may put your information, your computer or your network in danger [5]. these applications still pose a threat to user privacy. Because they are considered very effective in distributing viruses, bots to launch DDOS attacks, spyware, malware, trojans, etc., by sharing fake files or other ways.

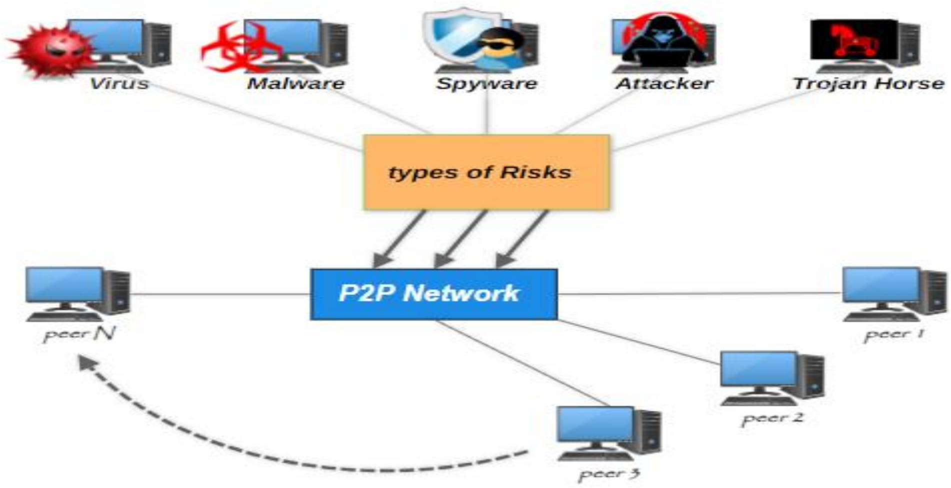


Figure: The risk of using P2P applications

Among the P2P file sharing applications that are very popular (µtorrent [6] and emule [7]). emule is a popular file sharing application which is based on the eDonkey and Kademlia protocol [8]. µTorrent is a very popular file sharing application that implements the BitTorrent protocol on the internet, and consumes a large bandwidth, which affects the service within the corporate network and causes problems of the same denial of Service (DOS) [9]. We always need a new way to detect the use of these applications within the network of the company and the ability to control or prevent them based on intrusion detection systems.

In addition to intrusion detection systems that monitor the traffic passing through the network and examine the payload of each packet, help us detect P2P applications, based on the signature of the protocols used by p2p applications. There are many open-source intrusion detection systems available, for example snort [10]. It monitors each load of the package and raises alerts when a

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

predefined signature is matched. Only, we always need to constantly track the development of P2P applications and extract new signatures.

2 RELATED WORK

2.1 Packet inspection (P2P Traffic analysis)

In-depth analysis of the packets by use Wireshark will be performed for each previous operating state to extract digital signatures of protocols P2P, the latter will allow us to identify the use of each protocol. The figure 1 shows the working architecture used for analysis part.

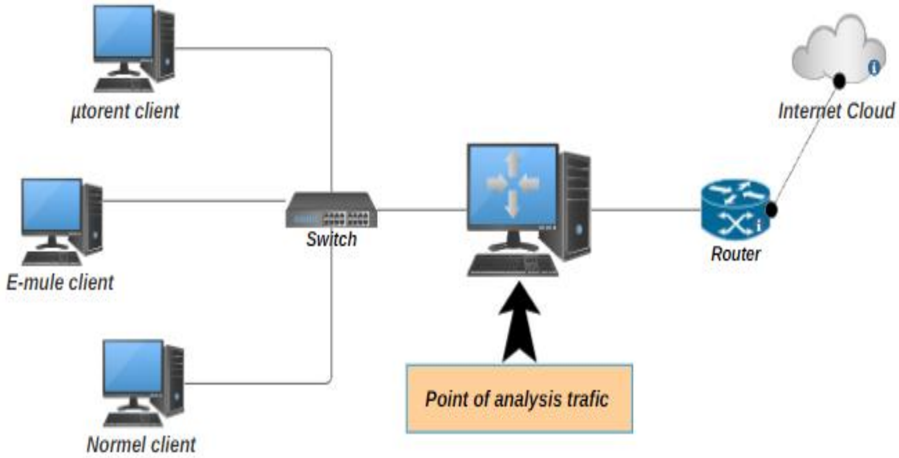


Figure 1: Analysis of traffic by Wireshark

A. Traffic of Application emule

1- After launching emule client and capture packets that sent and received while contacting between servers and other peers by Wireshark as shown in the figure

No.	Time	Source	Destination	Protocol	Length	Info
77	32.466787	10.0.0.3	59.126.251.46	eDonkey	167	eDonkey TCP: Hello
82	33.127163	59.126.251.46	10.0.0.3	eDonkey	170	eDonkey TCP: Hello Answer
84	33.282754	10.0.0.3	59.126.251.46	eDonkey	65	eMule Extensions TCP: Second Identification State
88	33.844041	59.126.251.46	10.0.0.3	eDonkey	88	eMule Extensions TCP: Hello, eMule Extensions TCP: Second Identification State
91	33.900927	10.0.0.3	59.126.251.46	eDonkey	68	eMule Extensions TCP: Unknown
94	34.355990	59.126.251.46	10.0.0.3	eDonkey	137	eMule Extensions TCP: Public Key
95	34.410249	10.0.0.3	59.126.251.46	eDonkey	162	eMule Extensions TCP: Hello, eMule Extensions TCP: Public Key
99	35.025006	10.0.0.3	59.126.251.46	eDonkey	109	eMule Extensions TCP: Signature
100	35.277777	59.126.251.46	10.0.0.3	eDonkey	109	eMule Extensions TCP: Signature
122	49.566192	10.0.0.3	91.208.184.143	eDonkey	48	eDonkey UDP: Server Status Request
126	55.221325	10.0.0.3	212.83.184.152	eDonkey	48	eDonkey UDP: Server Status Request
137	60.737926	10.0.0.3	183.136.232.234	eDonkey	48	eDonkey UDP: Server Status Request
147	66.255303	10.0.0.3	62.210.28.77	eDonkey	48	eDonkey UDP: Server Status Request
172	70.676874	10.0.0.3	47.37.145.12	eDonkey	48	eDonkey UDP: Server Status Request

Figure 2: E-Donkey packets captured during connection establishment

We notice there are multiple of UDP and TCP packets. In the first, emule operating step, which allows us establishment connection to download files. If we want to identify and explain all messages of E-Donkey protocol, we will need to study and analyze these requests in detail.

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

According to figure (2), the establishment of the connection with the server, "***59.126.251.46***" begins with a "***Hello Client***" query. The contents of this request are detailed in the figure 3 below:

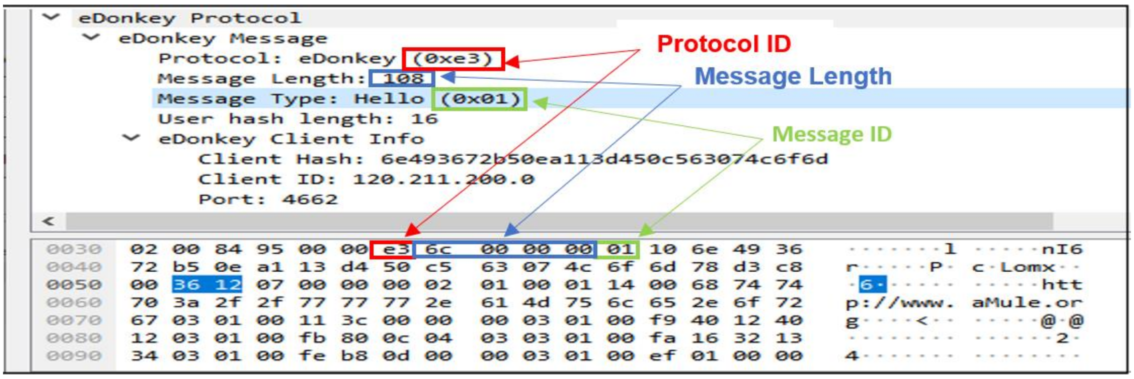


Figure 3: Capture the packet (E-Donkey-Hello)

The server responds to this request by "***Hello Answer***", the contents of this query are detailed in the figure 4:

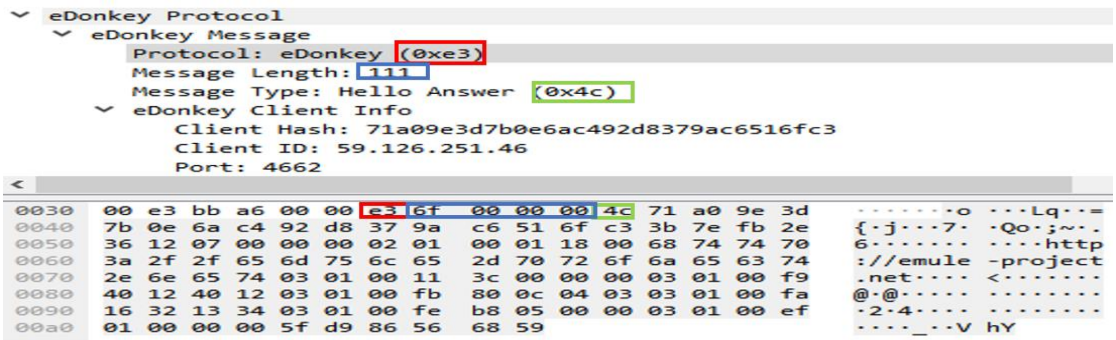


Figure 4: Capture the Paquet (server hello answer)

After connecting, the E-DONKEY server and the client starts the exchange of additional parameters that relate to the identification and sharing options.

The client sends «***the second identification state***» to ensure communication with the server. Subsequently, the server responds with a query that contains the two previous information in «***Hello & second identification state***».

The server offers a «***Public -key***» to the client by request, to index the client's ID at the server. The client responds with a query containing the confirmation of this key.

To complete the identification, the client proposes to the server a special signature that will identify the downloads or sharing, by a signature request. The server responds with a request containing the signature which will later identify all the tasks of this client.

The client automatically establishes UDP connections with the servers in this network to constantly make status updates, using the «***Server status request***» query.

2- Search of file: Once you create a connection with E-Donkey Server. Messages between the server and the client are passed using UDP.

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

No.	Time	Source	Destination	Protocol	Length	Info
102	32.101501	10.0.0.3	80.208.228.241	eDonkey	62	eDonkey UDP: Reask File Ping
103	32.461742	80.208.228.241	10.0.0.3	eDonkey	914	eDonkey UDP: Search File Results
104	32.461850	80.208.228.241	10.0.0.3	eDonkey	899	eDonkey UDP: Search File Results
105	32.466836	80.208.228.241	10.0.0.3	eDonkey	859	eDonkey UDP: Search File Results
106	32.496497	80.208.228.241	10.0.0.3	eDonkey	925	eDonkey UDP: Search File Results
107	32.767910	80.208.228.241	10.0.0.3	eDonkey	843	eDonkey UDP: Search File Results

Figure 5: E-Donkey packets captured during a search emule

The client sends a request to the server "***80.208.228.241***", of the «***Reask File Ping***» type to locate the file and the server responds with a query «***Search file results***». The emule client uses the «***search File***» query to request information about searching, this query is broadcast to all servers, only the primary server of the E-Donkey network responds to ***LOWID***.

3- The file download using emule is based entirely on the KADEMLIA protocol, the latter operates according to a UDP-based mechanism for downloading even if the TCP ports are blocked. The main queries of this protocol are represented in the following figure 6:

No.	Time	Source	Destination	Protocol	Length	Info
7	4.641334	10.0.0.3	1.85.248.135	eDonkey	64	Kademlia UDP: KADEMLIA2_HELLO_REQ
10	5.095082	1.85.248.135	10.0.0.3	eDonkey	80	Kademlia UDP: KADEMLIA2_HELLO_RES
11	5.469505	10.0.0.3	1.85.248.135	eDonkey	77	Kademlia UDP: KADEMLIA2_REQ
12	6.015294	1.85.248.135	10.0.0.3	eDonkey	111	Kademlia UDP: KADEMLIA2_RES
98	31.254452	10.0.0.3	220.190.11.227	eDonkey	64	Kademlia UDP: KADEMLIA2_HELLO_REQ
190	88.574234	10.0.0.3	58.253.40.192	eDonkey	64	Kademlia UDP: KADEMLIA2_HELLO_REQ
214	98.658998	10.0.0.3	117.62.48.69	eDonkey	77	Kademlia UDP: KADEMLIA2_REQ
457	108.050690	10.0.0.3	106.84.201.217	eDonkey	77	Kademlia UDP: KADEMLIA2_REQ

Figure 6: E-Donkey packets captured when downloading a file

According to figure 6, it can be seen that the functioning of the KADEMLIA protocol is based on four main types of queries: ***KADEMLIA_hello_REQ***, ***KADEMLIA_hello_RES***, ***KADEMLIA2_REQ***, ***KADEMLIA2_RES***.

B. Traffic of Application µTorrent

1- When we need to get the Metadata of file from the sites that offers torrents. We notice through the HTTP requests that are exchanged while connecting for obtaining the ".torrent" file, that are showing in figure 7.

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

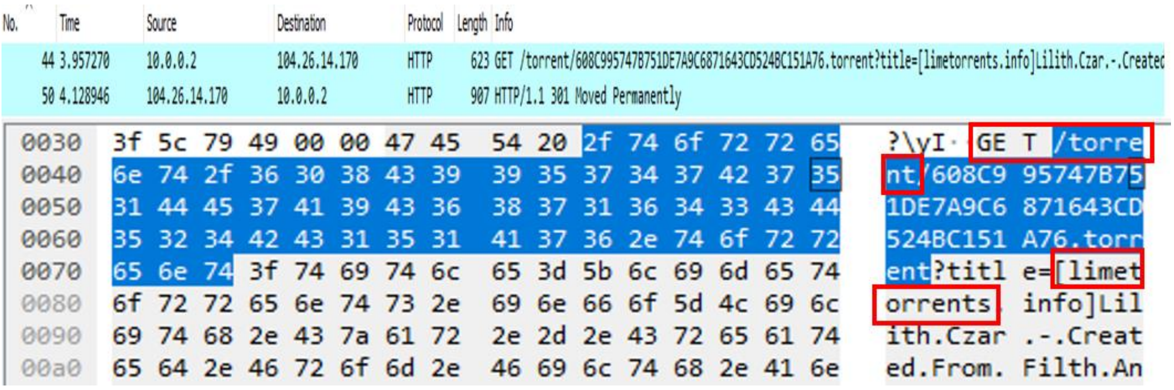
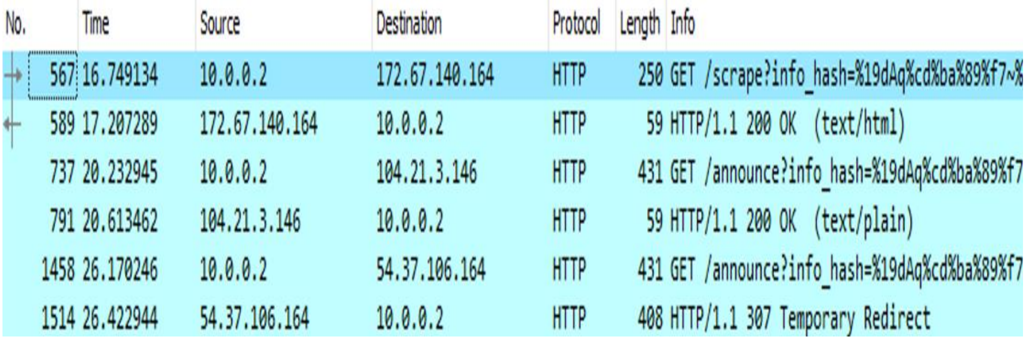


Figure 7: Content of HTTP GET TORRENT file

The client requests information from the web server "***104.26.14.170***" about the ".torrent" file, this request is sent once the user wants to download the "***.torrent file***".

2- Once the metadata file is running on the BitTorrent client, the client is starts connecting with the tracker. The packets captured in this step are shown in figure (8):



No.	Time	Source	Destination	Protocol	Length	Info
567	16.749134	10.0.0.2	172.67.140.164	HTTP	250	GET /scrape?info_hash=%19dAq%cd%ba%89%f7~%
589	17.207289	172.67.140.164	10.0.0.2	HTTP	59	HTTP/1.1 200 OK (text/html)
737	20.232945	10.0.0.2	104.21.3.146	HTTP	431	GET /announce?info_hash=%19dAq%cd%ba%89%f7
791	20.613462	104.21.3.146	10.0.0.2	HTTP	59	HTTP/1.1 200 OK (text/plain)
1458	26.170246	10.0.0.2	54.37.106.164	HTTP	431	GET /announce?info_hash=%19dAq%cd%ba%89%f7
1514	26.422944	54.37.106.164	10.0.0.2	HTTP	408	HTTP/1.1 307 Temporary Redirect

Figure 8: Peer-Tracker main requests - HTTP GET

The client contacts a server, it called tracker. In this time, client contacts the tracker «***172.67.140.164***» by sends *http GET SCRAPE* «***Get /scarpe? Info_hash***» query that contains the file ID «***info_hash***» to get information about the file that want to download it. In addition to that, the client also sends *http GET ANOUNCE* «***Get /announce? Info_hash***» query with the same ID «**info_hash**» to all available trackers [9].

Once one of these trackers respond, let's take a tracker «***104.21.3.146***», it responds by message the request «***http/1.1 200 OK***» with the text/plain. This response contains a list of peers that allows the client to establish connections with peers that shares the file. Only the HTTP protocol that supports the establishment of the connection between peers and tracker.

3- Once a client connection with peers, we notice the first message it sends by client is BitTorrent Handshake.

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

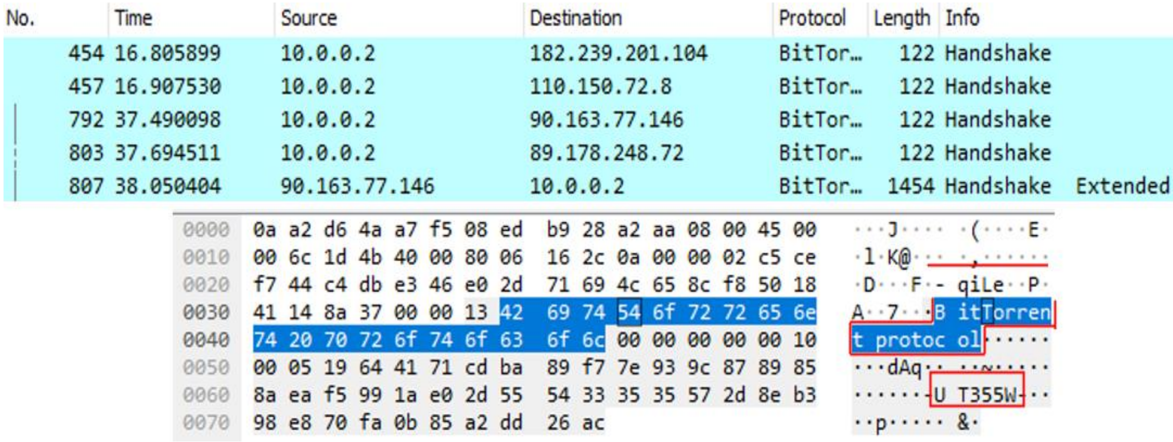


Figure 9: BitTorrent HANDSHAKE

The HANDSHAKE query contains information indicate to the BitTorrent protocol and client (Peer ID). The answer of this query is «***HANDSAKE EXTENDED***». Then, the client joins the download SWARM, and starts exchange with other peers concerning port and pieces available. Among the extensions of the Bittorrent protocol, BitTorrent uses a "***distributed hash table***" (DHT) for storing peer contact information for "***trackerless***" torrents. In effect, each peer becomes a tracker called node. Each node has a globally unique identifier known as the "***node ID***" Node IDs are chosen at random from the same 160-bit space as BitTorrent infohashes. BitTorrent clients include a DHT node, which is used to contact other nodes in the DHT to get the location of peers to download from using the BitTorrent protocol. The protocol is implemented over UDP. Figure 10 clearly illustrates this appearance. In the sniffer traces associated with protocol DHT that represented in "***get_peers1***" query [9]. Get_peers query has two arguments, represented by the chain «***d1:ad2:id20***»containing the node ID of the querying node and containing information of torrent by «***info_hash20*** » [11].

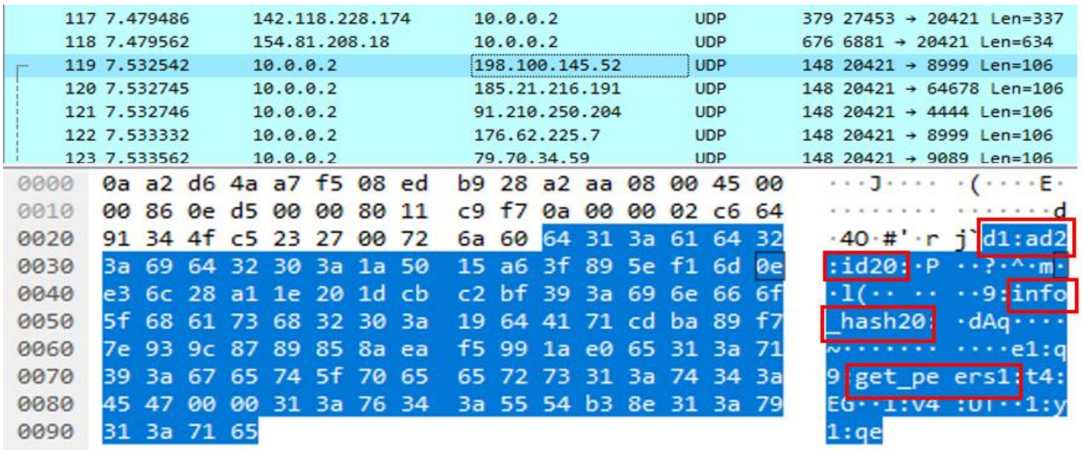


Figure 10: DHT –Get_Peers

4-Encrypted mode on application μTorrent (Forced)

The signatures of the BitTorrent protocol found until now are all in clear mode, once the encrypted mode enabled the traffic analysis becomes as Figure (11) shows us:

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

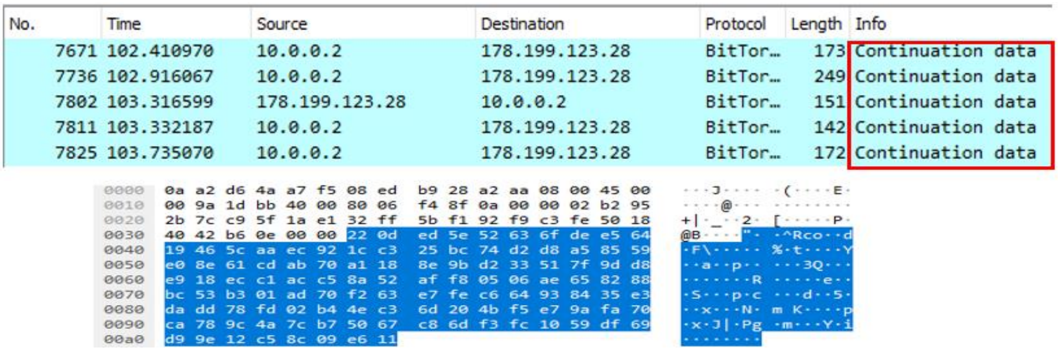


Figure 11: BitTorrent HANDSHAKE – Crypto

All Peer-to-Peer exchanges that are based on BitTorrent will be encrypted. Protocol encryption is a strengthening to privacy and confidentiality. In addition, traffic makes more difficult to determine by parties.

2.2 Signatures extracted from the analysis traffic

A. Signatures of protocol E-Donkey

Our digital of signatures Extraction Structure Based on three Fields in the Captured E-Donkey Packet:

1. Protocol: A protocol ID with a byte - 0xE3 for e-donkey, 0xc5 for emule, 0xE4 for Kademlia.

2. Size: The number of bytes between the protocol ID and the message type of this protocol.

3. Type : A unique byte - a unique message ID.

Through our analysis, the Table of Signatures of the E-Donkey protocol shown below is deducted (Table 1):

N	TYPE OF MSG	Length	Identifier	Protocol transport
0	eDonkey HELLO	XX XX XX XX	E3 01	TCP
1	eDonkey Hello answer	XX XX XX XX	E3 4C	TCP
2	emule extensions Second identification state	XX XX XX XX	C5 87	TCP
3	emule extensions public key	XX XX XX XX	C5 85	TCP
4	emule extensions Hello answer	XX XX XX XX	C5 4C	TCP
5	emule extensions Signature	XX XX XX XX	C5 86	TCP
6	eDonkey Get server info		E3 A2	UDP
7	eDonkey Server statut		E3 97	UDP
8	eDonkey Server Statut req		E3 96	UDP
9	KADEMLIA-KADEMLIA2_HELLO_REQ		E4 11	UDP
10	KADEMLIA-KADEMLIA2_HELLO_RES		E4 19	UDP
11	KADEMLIA-KADEMLIA2_REQ-FIND NODE		E4 21 0B	UDP
12	KADEMLIA-KADEMLIA2_REQ-FIND VALUE		E4 21 02	UDP

13	KADEMLIA-KADEMLIA2_RES		E4 29	UDP
14	KADEMLIA-KADEMLIA2_REQ		E4 21	UDP
15	KADEMLIA-KADEMLIA_FINDBUDDY_REQ		E4 51	UDP
16	eDonkey Get Sources		E3 9a	UDP
17	eDonkey-Search File		E3 98	UDP
18	eDonkey-Search File Results		E3 99	UDP
19	eDonkey-Reask File Ping	Length	E3 90	UDP

Table 1: E-donkey Protocol Digital signatures

B. Signatures of Protocol BitTorrent

We have extracted from using the Torrent application from the beginning of installation until the download, several frequent signatures. Bit Torrent's signature analysis results are summarized in the table 2:

N	Nom	Contents	Protocol transport
1	GET TORRENT	GET /torrent	TCP
2	GET SCARPE	GET /scrape?info_hash	TCP
		User-Agent: uTorrent	TCP
3	GET/ ANOUNCE	GET /announce? info_hash	TCP
		User-Agent: uTorrent	TCP
4	HADSHAKE	BitTorrent protocol	TCP
5	HANDSHAKE EXTENDED	ut_metadata	TCP
		metadata_size	TCP
7	DHT Peer	d1:ad2:id20	UDP
		info_hash20	UDP
		get_peers1	UDP
8	DHT Ping Tracker	/Announce	UDP

Table 2: BitTorrent Protocol Digital signatures

2.3 Create rules of extracted signatures

The structure of creating the detection rules of these two protocols is based completely on the transport protocol (TCP and UDP) in the first place. As previously mentioned, each protocol is identified by a different method.

A. The rules that refer to App emule

We will give SID number to identify Snort rules signature of E-donkey (**1000000 + order of signature in the table**) **e.g.:**

- 1000000 >> indicate first signature from E-donkey
- 1000001 >> indicate second signature from E-donkey

- Rule N° 01: E-Donkey-Hello

Alert tcp any any -> any any (msg:" Possibility of using Emule: eDonkey-Hello "; content:" |E3|"; depth:1; content:" |01|"; depth:1; distance:4; sid:1000000; rev:1;)

- Rule N° 02: E-Donkey-Hello answer

Alert tcp any any -> any any (msg Possibility of using emule: mule extensions-Second identification state "; content:"|C5|"; depth:1; content:"|87|"; depth:1;distance:4; sid:1000002; rev:1;)

- Rule N° 03: emule Extensions-Second identification state

Alert tcp any any -> any any (msg: "Possibility of using emule: eDonkey-Hello answer "; content:" |E3|"; depth:1; content:" |4C|"; depth:1; distance:4; sid:1000001; rev:1;)

- The same process for remaining alerts.

B. The rules that refer to App µTorrent:

- 2000001 >> indicate first signature from BitTorrent
- 2000002 >> indicate second signature from BitTorrent

- Rule N° 01: handshake

Alert tcp any any -> any any (msg: "Possibility of using uTorrent: handshake "; content:"BitTorrent protocol"; sid: 200001 ; rev:1;)

- Rule N° 02: handshake extension

Alert tcp any any -> any any (msg: "Possibility of using uTorrent: handshake extended "; content:" ut_metadata"; content:" metadata_size"; sid: 2000002; rev:1;)

- The same process for remaining alerts.

IMPLEMENTATION AND DISCUSSION

IMPLEMENTATION

After extracting the digital signatures of P2P protocols to detect the usage of P2P applications and protect the network, we employed the Snort system as our chosen intrusion detection system.

Snort operates based on the extracted signatures of protocols. To evaluate the effectiveness of our approach, we conducted tests on a simulated network that simulates a typical company network. The network setup is illustrated in the figure below.

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

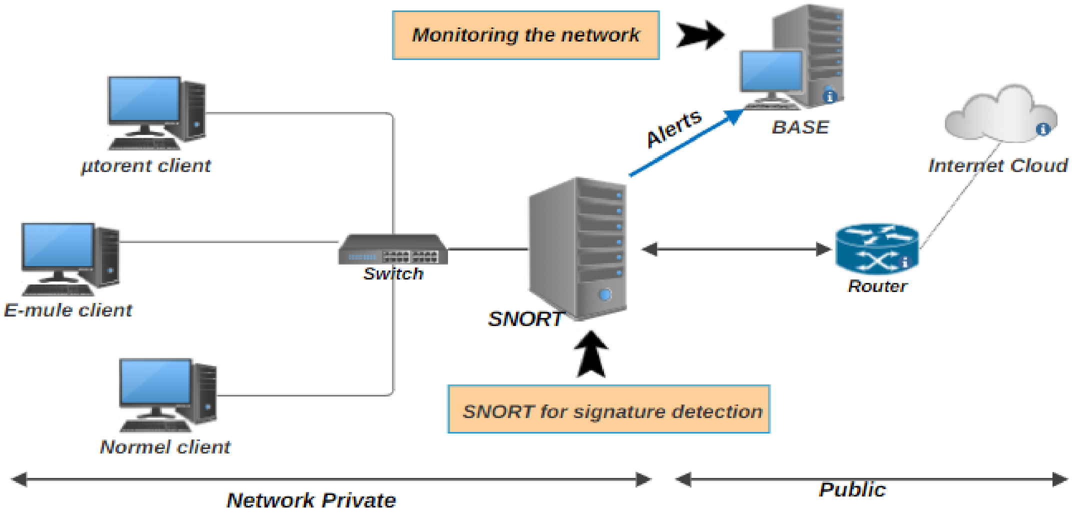


Figure 12: Work Architecture Laboratory – with P2P Apps

In our implementation, the extracted signatures are written into a file named "local.rules.txt" within the Snort system. Once the signatures are in place, we launch the Snort system to begin the intrusion detection process. Additionally, we utilize the graphical interface BASE "Basic Analysis and Security Engine" to facilitate the reading of alerts and real-time monitoring of the network status. Figure 13 illustrates the graphical interface used for this purpose.

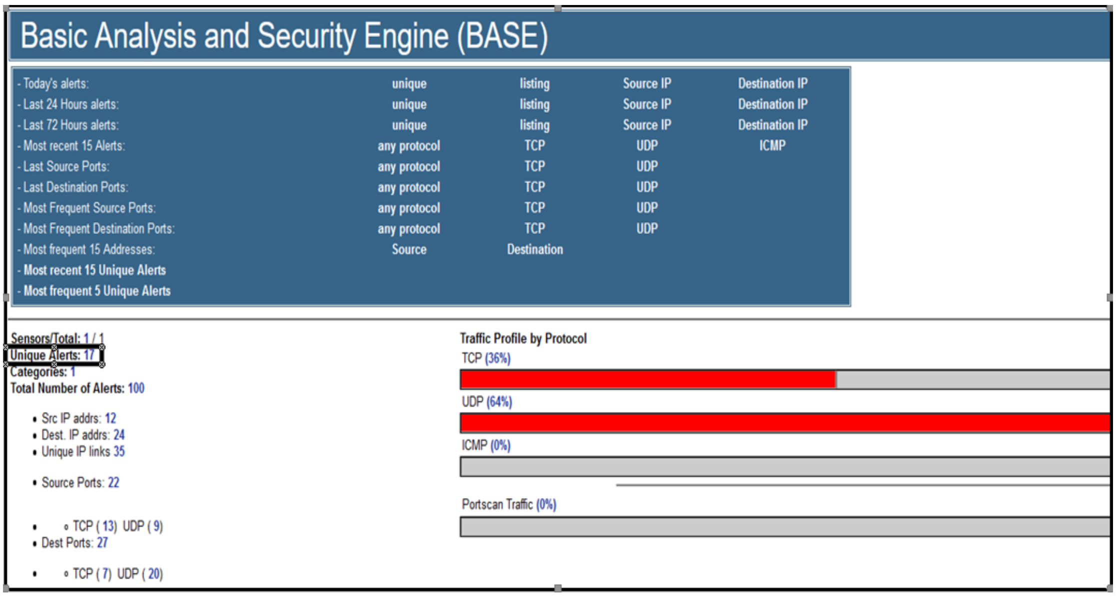


Figure 13: Results generated by Snort on Basic Analysis and Security Engine

DISCUSSION

1. Once the client uses emule, Snort launches UDP / TCP alerts. These alerts are represented in the following figure:

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

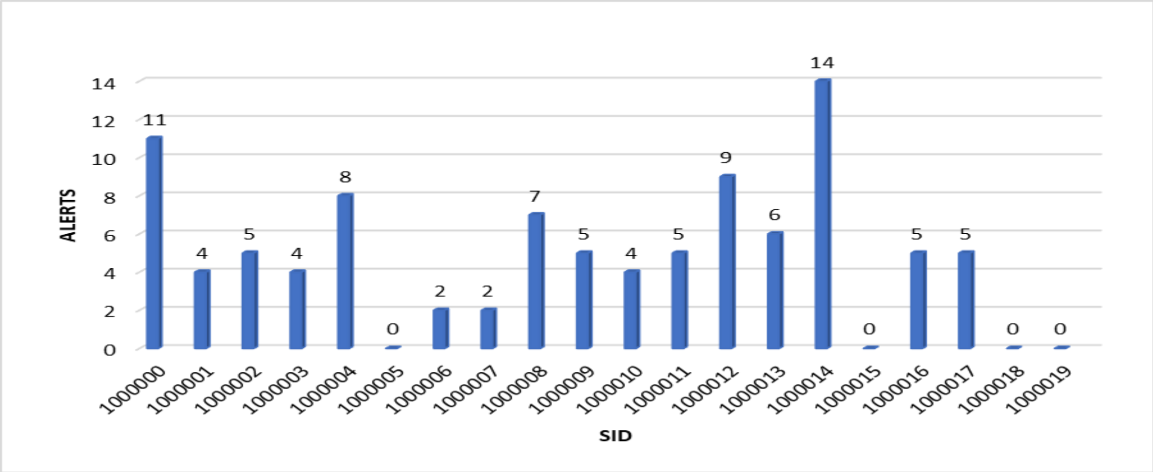


Figure 14: Graphic Columns of E-Donkey Detection Results

According to figure (14) of alerts, we note the presence of the Kademlia protocol. Numerous captured queries are KAD requests, the emule client is based primarily on KAD for searching and downloading files. The absence of four rules can be noted, which are based on identifiers of additional e-donkey operating messages.

2. Once the client µtorrent launched a download of torrent, all discovered alerts are displayed in the following figure 15:

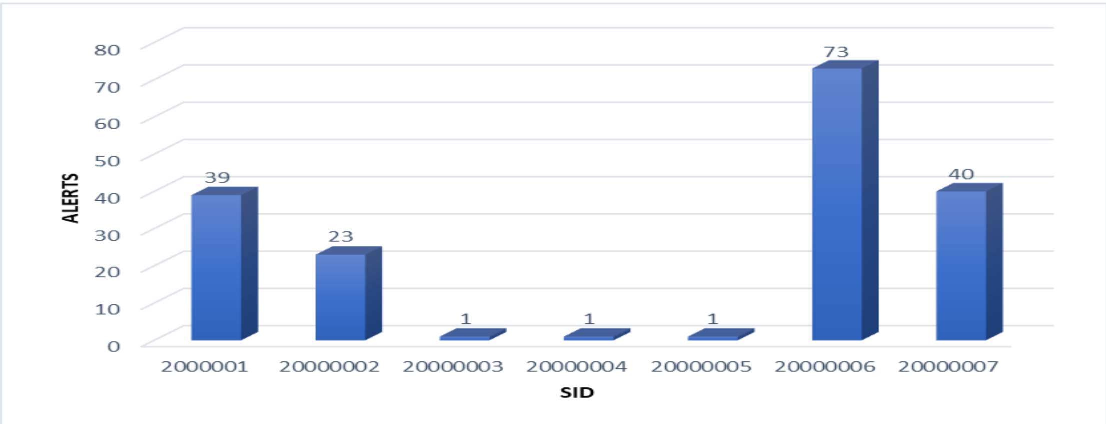


Figure 15: Graphic columns of µTorrent detection results

It should also be noted that the number of alerts generated by the "DHT Peer" rule is very large, this is completely normal because this rule generates an alert on each request send to peers. The operation of the BitTorrent protocol is based completely on the "BitTorrent Handshake" request.

3. Results indicate the disclosure of the use of the µtorrent application in the coded mode by SNORT, as it releases 100 alerts that match five rules and the absence of only two.

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

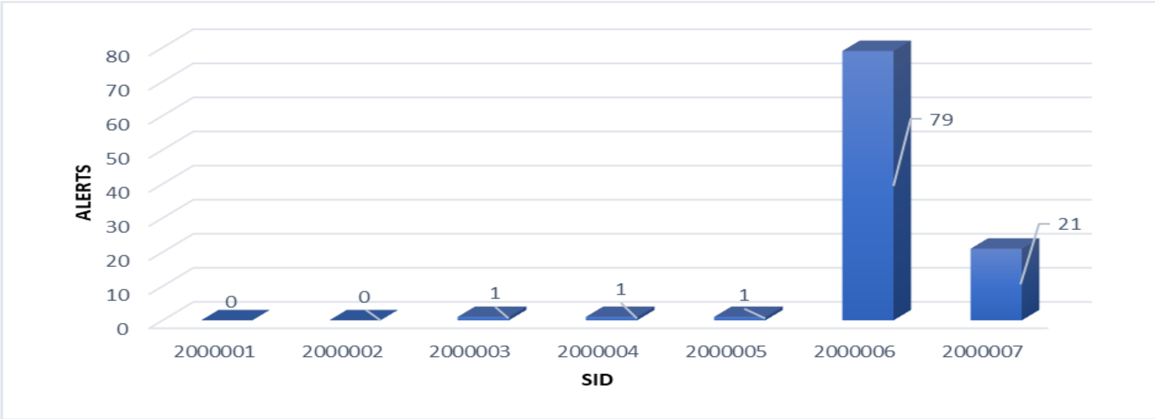


Figure 16: Graphic columns of detection results μTorrent encrypted

We can clearly note the absence of the alerts triggered by the two "Handshake" and "Handshake Extended" rules of the BitTorrent protocol, due to encryption.

We conclude from all these results, taken at a specific time interval, that the percentage of detection of P2P applications by this method was approximately 96%.

RELIABILITY TEST

To assess the reliability of our rules, we focused on the aspect of false positives, which refers to events that generate an alert indicating the use of P2P applications when there is actually no such usage. To ensure the accuracy of our rules, we conducted a test in which no P2P applications were used within the network.

During this test, we monitored the network traffic in real-time for a duration of 4 days. We ran the Snort system to detect any alerts related to P2P protocols. The results of this experiment were tracked and analyzed using the primary interface called "BASE." Figure 17 presents the recorded results of this experiment.

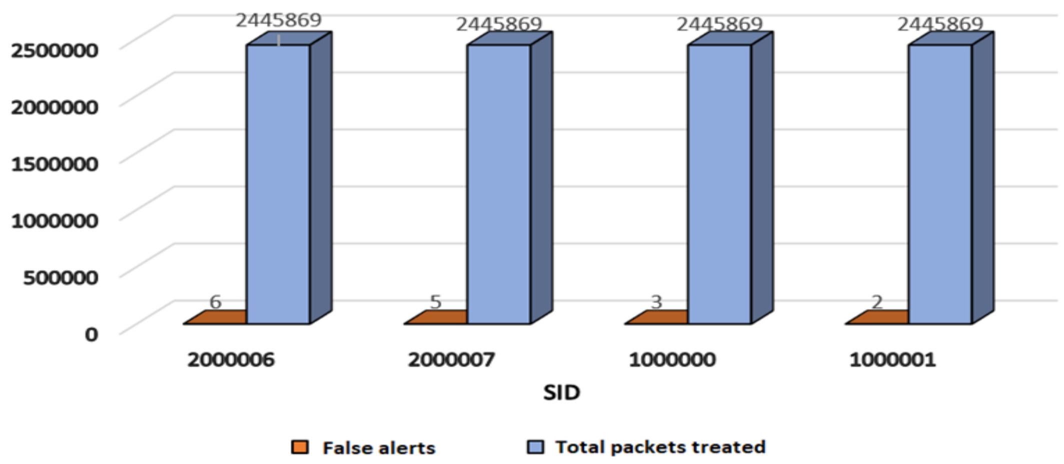


Figure 17: Graphic columns of the results of false alerts

The following figure displays the records for four days, 16 false alerts were taken from 2445869 packets treated for 4 days. The change in the wrong alert rate can be clear after each test, with an acceptable average of 0.000643 %, by this rate, its reliability can be confirmed. There is always a wrong positive rate on this type of alert, so it is necessary to check the context of alerts in order to determine whether one of them deals with a real alert because of the use of a P2P network.

CONCLUSION

In this study, our main focus was on presenting a novel approach to detect the usage of P2P applications within a network and mitigate the associated risks and drawbacks that pose a significant threat to both company and client privacy. To achieve this, we conducted a thorough analysis of the traffic generated by popular P2P file sharing applications such as µTorrent and eMule, which rely on the BitTorrent and eDonkey protocols, respectively. Through this analysis, we extracted digital signatures indicative of these protocols.

Building upon this analysis, we developed a strategy that employed new rules within the "Snort" intrusion detection system to identify and detect P2P applications. By implementing this strategy over several time periods, we achieved a remarkable 96% detection rate for P2P application usage within the network. However, we observed that P2P applications continue to evolve rapidly, necessitating the regular updating of rules to ensure effective and reliable detection within the network.

REFERENCES

[1] Saxena P, Sharma SK. Analysis of network traffic by using packet sniffing tool: Wireshark. International Journal of Advance Research, Ideas and Innovations in Technology. 2017;3(6):804-808.

[2] Sen S, Wang J. Analyzing peer-to-peer traffic across large networks. IEEE/ACM Transactions on Networking. 2004;12(2):219–232. doi:10.1109/tnet.2004.826277

[3] Hwang IS, Rianto A, Pakpahan AF. Peer-to-peer file sharing architecture for software-defined TWDM-PON. Journal of Internet Technology. 2020;21(1):23-32.

[4] Shoab M, Jubayrin S. A. Intelligent neighbor selection for efficient query routing in unstructured P2P networks using Q-learning. Applied Intelligence. 2021;52(6):6306–6315. doi:10.1007/s10489-021-02793-6

[5] T2021_21 Risks of File Sharing (15th December 2021), Guyana National CIRT, URL: https://cirt.gy/Tips?page=5, 2021.

[6] BitTorrent, Inc. (n.d.). MTorrent (uTorrent): A very tiny BitTorrent client. Retrieved from https://www.utorrent.com/, 2023.

[7] Project.net - official emule homepage. downloads, help, docu, news... (n.d.). Retrieved from https://www.emule-project.com/home/perl/general.cgi?l=1&rm=download, 2024.

[8] Jaw E, Wang, X. A novel hybrid-based approach of Snort Automatic Rule Generator and security event correlation (SARG-SEC). PeerJ Computer Science. 2022;8. doi:10.7717/peerj-cs.900.

RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR
ISSN 2675-6218

[9] Mehdi M. Interception of P2P Traffic in a Campus Network. Romanian Journal of Information Technology & Automatic Control/Revista Română de Informatică și Automatică. 2019;29(2):21-34. doi:10.33436/v29i2y201902.

[10] Locher T, Schmid S, Wattenhofer R. edonkey & emule's kad: Measurements & attacks. Fundamenta Informaticae. 2011;109(4):383-403, doi:10.3233/fi-2011-518.

[11] Andrew Loewenstern, A. N. (n.d.). BitTorrent.org. Retrieved from https://www.bittorrent.org/beps/bep_0005.html. 2008.