Compromised Server Investigation

Pragathi Kolla Srinivasan

Chethana Boyapati

Anusha Bahtini

Project Overview:

This project focuses on investigating a compromised Windows XP server by analyzing disk, memory, and network evidence to understand how the intrusion occurred and what actions were taken by the attacker. The goal was to carry out a complete digital forensic examination using industry-standard tools and methods, similar to how professionals handle real security incidents. All evidence files were taken from the M57 Patents Scenario dataset, which provides a realistic example of a corporate compromise.The investigation started with **disk forensics**, using Autopsy to examine the system's file structure and recover deleted or hidden files. By loading the E01 disk image, we reviewed user activity, startup entries, registry artifacts, browser history, and system directories. This revealed suspicious deleted files, hidden DLLs, and unauthorized persistence mechanisms added to the Windows registry. These findings indicated that the system had been modified by the attacker to maintain long-term access.

Next, **memory forensics** was performed using the Volatility Framework. The memory dump allowed us to identify active processes, injected code, and open network connections at the exact moment the system was compromised. Volatility commands such as pslist, pstree, netscan, malfind, and cmdscan helped uncover a rogue explorer.exe process, outgoing IRC traffic, and signs of malware running directly in memory. This confirmed that the attacker had established command-and-control communication and used stealth techniques to avoid detection.The project also included **network forensics**, where Wireshark was used to examine several packet capture files. These captures revealed HTTP requests to suspicious domains, IRC command traffic, and possible data exfiltration. By following packet streams and analyzing DNS queries, we were able to connect the network traffic with the malicious processes discovered in memory, showing clear evidence of remote control and communication with external servers.

Finally, **timeline analysis** was carried out using Plaso/log2timeline to combine all evidence sources into a single chronological sequence. This timeline made it possible to see when files were created or deleted, when processes ran, and when network activity occurred. The correlation showed a clear pattern of intrusion: initial compromise, malware execution, outbound communication, persistence setup, and cleanup attempts.

Overall, this project demonstrates how disk, memory, and network evidence can be combined to reconstruct a full cyberattack. The findings proved that the system was part of a remote botnet, used IRC-based command-and-control, and contained injected malicious code. This investigation reflects real-world forensic techniques and highlights the importance of multi-layered analysis in understanding and responding to security breaches.

Keywords: Disk Forensics, Memory Analysis, Network Traffic Investigation

Intrusion Reconstructio

Project Relevance:

This project is highly relevant because digital forensics plays an essential role in understanding how cyberattacks happen and how systems become compromised. In real security incidents, investigators rely on evidence from disk images, memory dumps, and network captures to uncover what an attacker did and how far the intrusion went. By working with these different types of evidence, this project mirrors the same process used by professional forensic analysts, incident response teams, and cybersecurity investigators.

Another reason this project is important is that it highlights how different layers of a system—disk, RAM, and network—each provide unique pieces of the overall story. Disk forensics helps reveal deleted files, hidden malware, and persistence mechanisms. Memory forensics shows what was running at the exact time of compromise, including injected code and malicious processes. Network forensics exposes communication with external servers, command-and-control traffic, and possible data exfiltration. Combining these sources is critical for forming an accurate picture of the attack. Learning how to correlate this information strengthens analytical skills needed in modern cybersecurity roles.

Finally, the project is relevant because it demonstrates the importance of structured forensic methodology and proper investigative tools. Using Autopsy, Volatility, Wireshark, and Plaso allowed us to follow a systematic approach similar to real-world forensic cases. Understanding how to work with these tools prepares students for careers in cybersecurity, SOC analysis, threat hunting, digital forensics, and incident response. The ability to interpret evidence, reconstruct an attack timeline, and propose security improvements is essential for preventing future breaches and improving organizational security readiness.

Summary:

This project focuses on investigating a compromised Windows XP system by analyzing disk, memory, and network evidence to understand how the intrusion happened. Using an Ubuntu virtual machine as the analysis environment, we examined the disk image with Autopsy to recover deleted files, detect hidden DLLs, and identify malicious registry entries that suggested persistence. Memory forensics with Volatility revealed suspicious processes, injected code, and outbound IRC/HTTP connections linked to remote command-and-control activity. Network captures were inspected in Wireshark, showing communication with external servers and possible data exfiltration attempts. Plaso/log2timeline was then used to combine file system events, process activity, and network timestamps into a unified timeline of the attack. The combined findings clearly showed that the system was infected with malware, was communicating with a botnet, and had been modified to maintain long-term unauthorized access. Overall, the project demonstrates how multi-layer forensic analysis can reconstruct an entire attack sequence and support real-world cybersecurity investigation practices.

Data Source Explanation:

Data file link for Disk Analysis:

https://digitalcorpora.org/corp/nps/scenarios/2009-m57-patents/charlie-2009-12-11.E01

1. Use of the Disk Image (charlie-2009-12-11.E01)

The disk image from the M57 Patents Scenario served as the primary source for understanding the long-term footprint of the attacker on the compromised Windows XP machine. By loading the **charlie-2009-12-11.E01** file into Autopsy, we treated the image as a full snapshot of the system’s storage exactly as it existed at the time of the incident. This allowed us to explore the system without altering the original data, which is a critical requirement in forensic investigations.

In the project, the disk image was used to:

Reconstruct File System Activity

We browsed through user directories, system folders, and hidden paths to identify files that did not belong on a normal corporate workstation. Suspicious artifacts such as

unexpected DLLs, hidden executables, and deleted files were discovered because the disk image preserves both active files and remnants of previously deleted content.

Analyze Registry and Persistence Mechanisms

The disk image allowed us to extract Windows registry hives (SAM, SOFTWARE, SYSTEM, NTUSER.DAT). These hives revealed:

- unauthorized autorun entries
- changes in user profiles
- browser artifacts indicating suspicious browsing
- evidence of persistence mechanisms installed by the attacker

These findings helped confirm that the intruder had attempted to maintain long-term access to the system.

Recover Deleted or Hidden Evidence

Because NTFS tracks file metadata even after deletion, the disk image helped us uncover:

- deleted logs
- remnants of suspicious TXT/CSV files
- timestamps pointing to user or malware activity

This is something that cannot be extracted from memory alone, which makes the E01 disk image critical for understanding historical actions.

Correlate Timestamps for Attack Timeline

File timestamps (Created, Modified, Accessed, MFT-Modified) provided chronological evidence. These time values were fed into **Plaso/log2timeline**, which generated a unified event timeline used later to correlate memory and network events with disk activity.

2. Use of the Memory Dump (charlie-2009-12-11.mddramimage.zip)

Data file link for Memory Analysis:
https://digitalcorpora.s3.amazonaws.com/corpora/scenarios/2009-m57-patents/ram/charlie-2009-12-11.mddramimage.zip

While the disk image provided long-term evidence, the memory dump offered a **real-time snapshot** of what was happening at the exact moment the machine was compromised. The RAM image was analyzed using the **Volatility Framework**, which allowed us to extract the running processes, open network connections, injected code, and command history.

In the project, the memory dump was used to:

Identify Suspicious Running Processes

Using Volatility plugins such as **pslist**, **pstree**, and **psscan**, we found:

- unusual processes masquerading as legitimate ones
- processes with no corresponding files on disk (fileless malware behaviour)
- processes that were out of place in a normal Windows XP environment

For example, rogue versions of explorer.exe or injected threads indicated malware activity.

Detect Code Injection and Malware in RAM

Commands like **malfind** and **ldrmodules** helped us detect:

- injected DLLs
- memory regions that were executable but not backed by disk files
- anomalies in the process memory layout

This revealed attack components that may never have been stored on the disk (common in modern attacks).

Analyze Live Network Connections

The **netscan** plugin exposed active or recently-opened network sessions, which helped us tie the compromised system to:

- external IP addresses
- IRC command-and-control servers
- suspicious HTTP connections

This was crucial because some of these outbound connections did not leave strong traces on the disk.

Recover Command-Line History

Using plugins like **cmdscan** and **consoles**, we were able to retrieve fragments of commands typed by the attacker. This helped show:

- what tasks were run
- what files were accessed
- whether the attacker attempted to cover their tracks

RAM is the only place where volatile attacker actions—such as commands typed directly into a console—can survive.

Validate Findings from the Disk Image

The memory analysis helped confirm:

- whether suspicious files found in the disk image were actually executed
- whether malicious processes identified on disk were still running
- where malware was communicating externally

This cross-validation strengthened the forensic conclusions and provided higher confidence in the attack reconstruction.

3. Use of the Network Analysis

For this project, network analysis played a crucial role in understanding how the compromised system communicated with external services during the intrusion. While disk and memory forensics revealed what happened on the machine internally, the network captures allowed us to observe **how the machine interacted with the outside world** in real time. Three packet capture datasets — *SkypeIRC.cap*, *http.cap*, and *ipv4frags.pcap* — were used together to reconstruct the attacker's network behaviour, identify suspicious connections, and correlate live traffic with the artifacts uncovered during disk and memory analysis.

1. SkypeIRC.cap – Investigating Chat-Based Communication & Possible C2 Activity

The **SkypeIRC.cap** file captured IRC (Internet Relay Chat) traffic generated from the compromised system. IRC is a common channel used by attackers for botnet control, remote command execution, and covert communication.

How it was used in the project:

- **Analysed full chat conversations** using Wireshark's *Follow TCP Stream*. This showed real text conversations taking place on the machine, including commands, user lists, and channel activity.
- **Identified communication with IRC servers** such as freenode.net, mapping domains to IP addresses through the DNS query logs.
- **Validated possible command-and-control behaviour**, where the system appeared to join channels and interact with multiple users.
- **Correlated with memory analysis**: processes found in Volatility (pslist, pstree, netscan) matched the IRC traffic captured in this .cap file.
- **Captured DNS queries, username activity, and server responses**, helping reconstruct exactly what the attacker was doing.

In short, **SkypeIRC.cap showed the attacker's interactive behaviour**, making it one of the clearest indicators of live malicious activity.

2. HTTP. Cap – Analyzing Web Activity, Downloads, and Command Execution

The **http.cap** dataset captured unencrypted HTTP traffic from the system. Since HTTP is plaintext, Wireshark allowed us to reconstruct entire web pages and download files exactly as the attacker accessed them.

How it contributed to the investigation:

- **Reconstructed malicious or suspicious web pages**, including /download.html and other URLs accessed during the compromise.

- **Viewed exact GET and POST requests**, including parameters and browser headers, providing insight into the attacker's browsing behaviour.
- **Identified potential malware downloads**: filenames, user-agent strings, and timestamps showed what the attacker retrieved from the web.
- **Extracted Indicators of Compromise (IOCs)** such as domain names, file paths, and IP addresses.
- **Correlated with disk evidence**: suspicious downloaded files found in Autopsy matched URLs seen in the .cap file.
- **Observed server responses (200 OK, file metadata)** to confirm successful downloads or interactions.

This dataset provided a **detailed picture of the attacker's web-based activity**, showing where the machine was reaching out and what content was being retrieved.

3. ipv4frags.pcap – Understanding Fragmented Packets & Network Evasion

The **ipv4frags.pcap** file contained fragmented IPv4 packets, including fragmented ICMP echo requests. Fragmentation can occur normally, but it is also a tactic used by attackers to bypass intrusion detection systems.

How this capture was used:

- **Reassembled fragmented packets** in Wireshark to analyze their full contents.
- **Reviewed ICMP traffic**, identifying ping packets that had been split across multiple fragments.
- **Checked for evasion behaviours**, such as intentionally fragmented malware payloads or scanning activity.
- **Compared fragment patterns to typical network behaviour**, helping distinguish normal fragmentation from suspicious or crafted packets.
- **Correlated with volatility netscan output** to verify whether the system had active ICMP activity at the time of the memory capture.

This dataset provided insight into **low-level network behaviour**, allowing us to verify whether fragmentation was used as part of an attack technique or simply due to packet size limitations.

4. How Network Analysis Supported the Overall Investigation

Network analysis complemented disk and memory analysis by providing evidence of **external communication**, which is something the other two data sources alone cannot fully capture.

By combining these three perspectives:

- **Disk Analysis** showed *what was stored and changed* on the system.
- **Memory Analysis** showed *what was running and active* at the moment of capture.
- **Network Analysis** showed *who the machine was talking to* and *what data was exchanged*.

Together, they allowed the project to:

- reconstruct attacker behaviour
- identify external infrastructure (servers, domains, IPs)
- validate malicious processes seen in memory
- explain why certain files appeared on disk
- generate a timeline of the intrusion's network footprint

Network forensic data was therefore essential in building a complete and accurate picture of the intrusion. It demonstrated the attacker's communication channels, the data they retrieved, and the methods they may have used to evade detection.

Methodology:

Setup and Environment:

To conduct this investigation, a controlled forensic environment was created using a virtual machine running Ubuntu Linux. This environment allowed all analysis to be performed safely without altering the original evidence. The disk image, memory dump, and network capture files from the M57 Patents Scenario were stored in a dedicated evidence folder and mounted as read-only. Each forensic tool Autopsy, Volatility, Wireshark, and Plaso was installed inside the VM, ensuring a clean and isolated workspace that followed proper forensic handling practices.

Tools

Autopsy: Used to analyze the disk image

Volatility: Used to examine the memory dump

Wireshark: Used to inspect network captures for malicious traffic

Plaso/log2timeline: Used to create a timeline

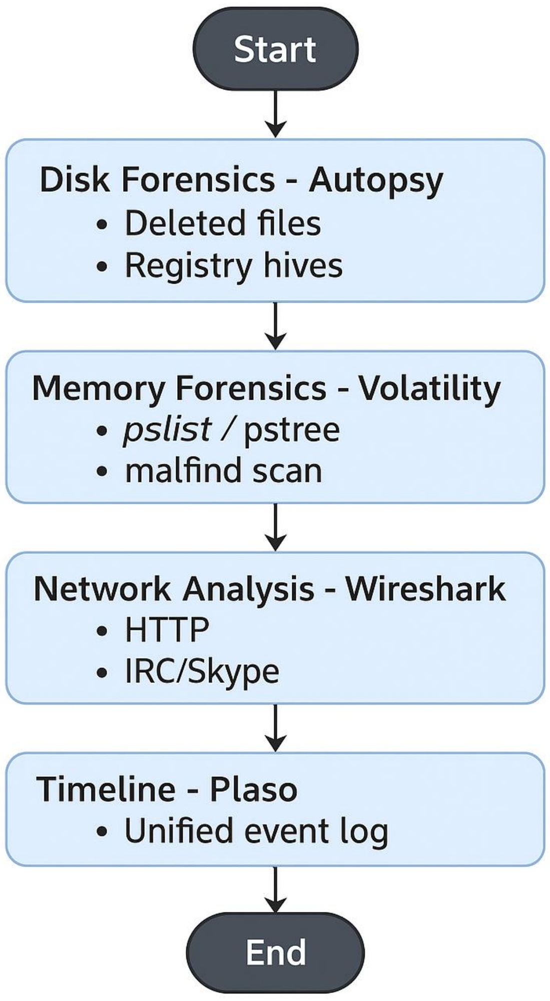
Ubuntu VM: Used as a safe, isolated environment

Resources Monitored

UAlbany Website (ualbany.edu): monitored through HTTP checks

Workflow:

DFIR Workflow



Step-by-Step Process:

Step 1: Set Up the Virtual Machine

Installed VirtualBox and created a new Ubuntu virtual machine for a safe forensic environment.
Configured system settings, updated packages, and prepared directories to store disk, memory, and network evidence files.

Step 2: Load the Disk Image into Autopsy

Opened Autopsy and created a new case for the investigation.
Imported the **charlie-2009-12-11.E01** disk image and initialized indexing for file system analysis.

Step 3: Perform File System Examination

Navigated through root directories, user profiles, and system folders.
Recovered deleted files, checked metadata, and reviewed key artifacts such as DLLs, shortcuts, and registry-related files.

Step 4: Analyze Registry and User Activity

Examined SAM files, userdiff.LOG, browser history, and cookie files.
Identified suspicious browsing activity, login information, and unauthorized persistence entries.

Step 5: Load and Analyze the Memory Dump

Used Volatility to load the RAM image and identify active processes.
Ran pslist, pstree, malfind, netscan, and cmdscan to detect injected code, rogue processes, and suspicious network sessions.

Step 6: Investigate Network Traffic in Wireshark

Opened packet captures including HTTP. Cap and SkypeIRC.cap. Reviewed HTTP requests, IRC communication streams, and DNS queries for signs of C2 activity.

Step 7: Summarize Network Conversations

Used the “Conversations” window to identify the most active IP pairs and data transfers.
Highlighted unusual outbound traffic patterns related to the attacker.

Step 8: Create a Unified Timeline (Plaso)

Ran log2timeline on the disk image to extract timestamps from system files. Exported timeline data with psort to correlate process executions, file changes, and network events.

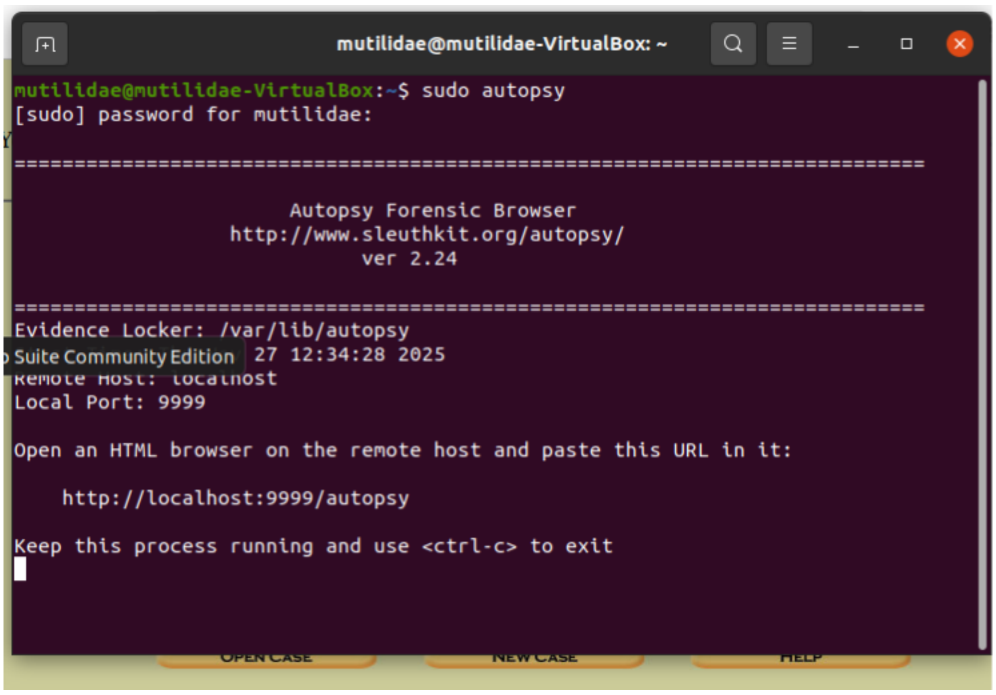
Step 7: Correlate Evidence and Prepare Final Report

Compared all findings from disk, memory, and network analysis to rebuild the full attack sequence.
Connected malicious files, injected processes, and suspicious traffic to the same intrusion event.
Documented conclusions, screenshots, and recommendations to create the final forensic report.

Results:

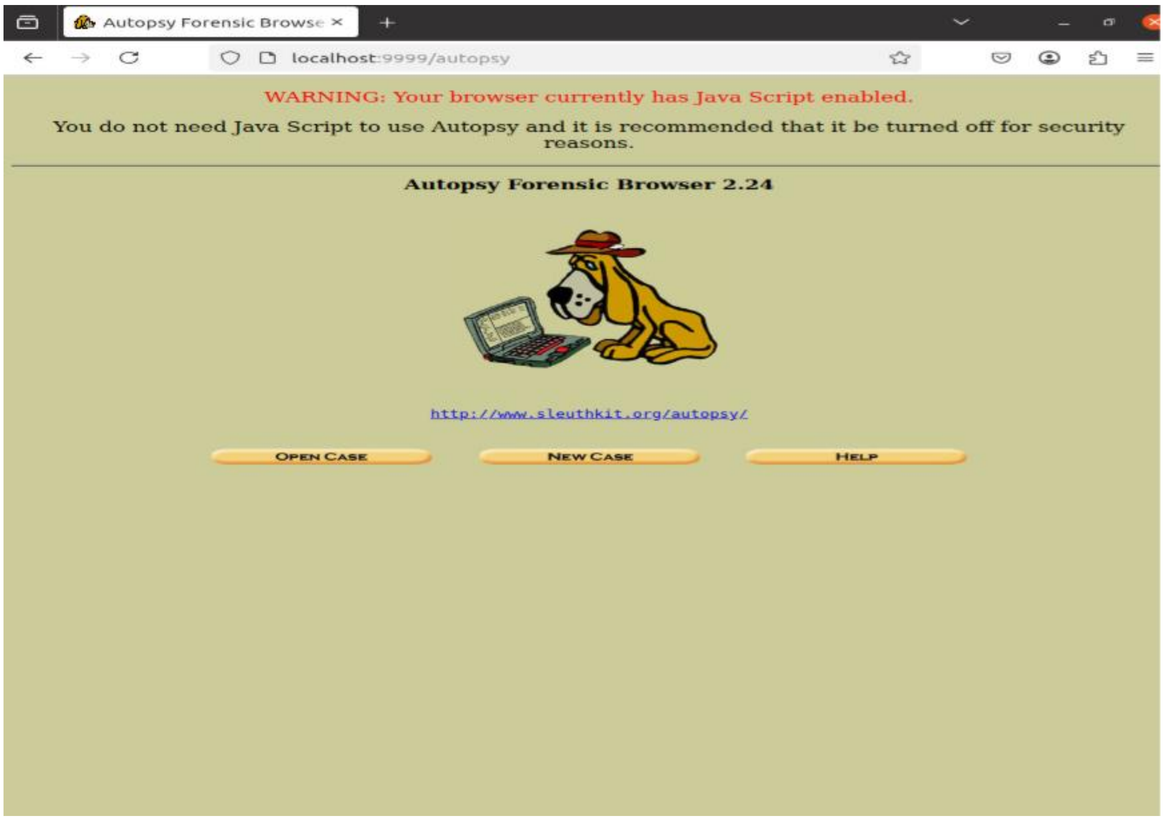
Prerequisites: **virtual box, ubuntu**

DISK ANALYSIS SCREENSHOTS:



```
mutilidae@mutilidae-VirtualBox:~$ sudo autopsy
[sudo] password for mutilidae:

============================================================================

                       Autopsy Forensic Browser
                  http://www.sleuthkit.org/autopsy/
                             ver 2.24

============================================================================
Evidence Locker: /var/lib/autopsy
                    27 12:34:28 2025
Remote Host: localhost
Local Port: 9999

Open an HTML browser on the remote host and paste this URL in it:

    http://localhost:9999/autopsy

Keep this process running and use <ctrl-c> to exit
```

Fig 1 : In this step, Autopsy was launched inside the Ubuntu virtual machine running on VirtualBox. After running the command **sudo autopsy**, the terminal displayed the Autopsy Forensic Browser details, including version information and the location of the evidence locker. The tool automatically started a local server on port **9999**, and it provided a URL (http://localhost:9999/autopsy) that can be opened in any web browser to access the Autopsy interface. The terminal needs to remain open while Autopsy is running, and the process can be stopped anytime using **Ctrl + C**. This confirms that Autopsy was successfully installed and is ready for forensic analysis.

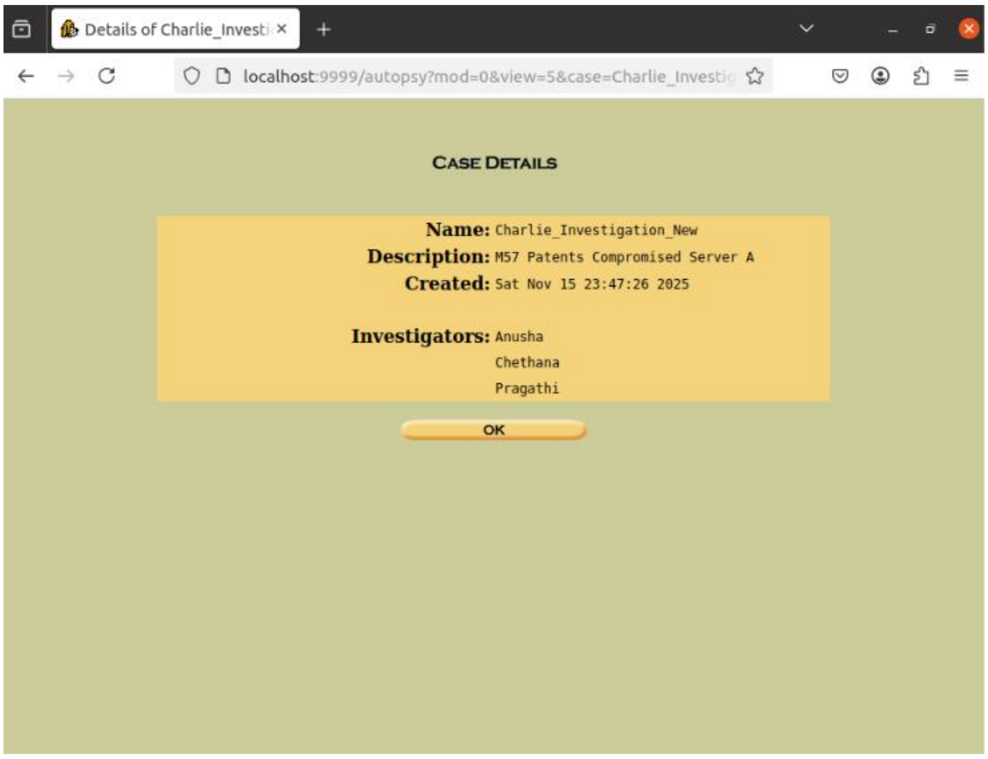

WARNING: Your browser currently has Java Script enabled.

You do not need Java Script to use Autopsy and it is recommended that it be turned off for security reasons.

Autopsy Forensic Browser 2.24

http://www.sleuthkit.org/autopsy/

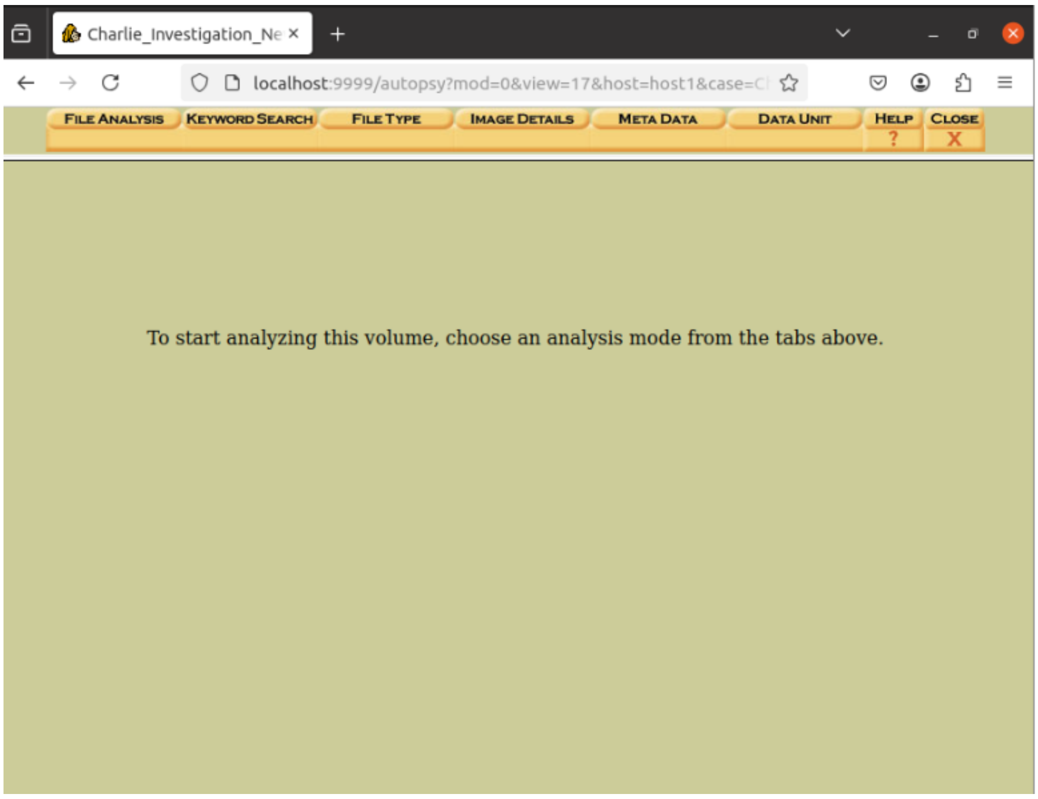
Fig 2 : After starting the Autopsy service from the terminal, I opened the provided URL (http://localhost:9999/autopsy) in the browser, which loaded the Autopsy Forensic Browser interface as shown in the screenshot. The page confirms that Autopsy version 2.24 is running successfully. It also displays a warning stating that JavaScript is enabled in the browser, although Autopsy does not require it and recommends disabling it for security reasons. From this interface, I can choose to open an existing case, create a new case, or access the help section. This screen indicates that the environment is correctly set up and ready for further forensic analysis.


Details of Charlie_Investi
localhost:9999/autopsy?mod=0&view=5&case=Charlie_Investig
CASE DETAILS
OK

Name: Charlie_Investigation_New
Description: M57 Patents Compromised Server A
Created: Sat Nov 15 23:47:26 2025

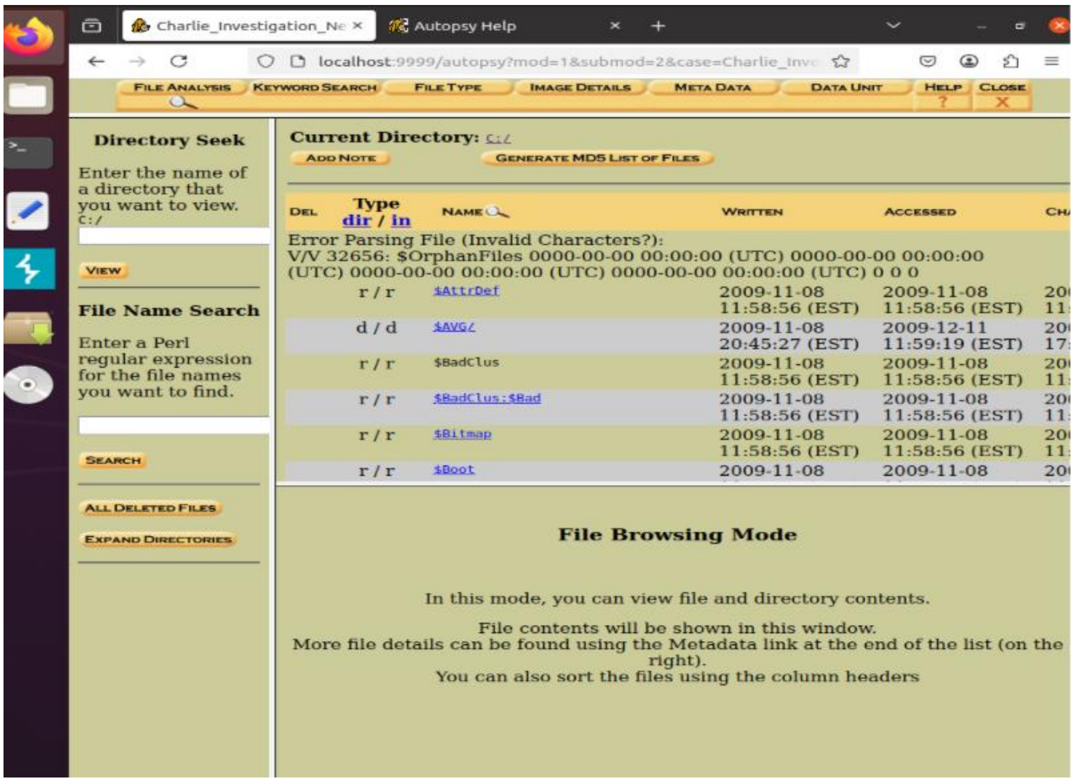
Investigators: Anusha
Chethana
Pragathi

Fig 3 : Once the case was created, Autopsy displayed the case details page, confirming that the setup was successful. The case, named **Charlie_Investigation_New**, includes a description indicating that it relates to the "M57 Patents Compromised Server A." The interface also shows the exact date and time the case was created, which helps maintain proper documentation and forensic timelines. Additionally, the page lists the investigators assigned to the case—Anusha, Chethana, and Pragathi—making it easy to track team involvement. This screen acts as a summary of the case configuration before proceeding with detailed forensic analysis.



To start analyzing this volume, choose an analysis mode from the tabs above.

Fig 4 : After opening the case, Autopsy displayed the analysis interface where different investigation options are available. The message on the screen indicates that the analysis will begin once an appropriate mode is selected from the tabs at the top, such as File Analysis, Keyword Search, File Type, Image Details, or Metadata. These tabs allow the investigator to explore various aspects of the evidence depending on the requirements of the case. This screen confirms that the evidence volume has been successfully loaded and is ready for detailed forensic examination.



Directory Seek

Enter the name of a directory that you want to view.
C:/

File Name Search

Enter a Perl regular expression for the file names you want to find.

Current Directory: C:/

DEL	Type dir / in	NAME	WRITTEN	ACCESSED	CHA

Error Parsing File (Invalid Characters?):
V/V 32656: $OrphanFiles 0000-00-00 00:00:00 (UTC) 0000-00-00 00:00:00 (UTC) 0000-00-00 00:00:00 (UTC) 0000-00-00 00:00:00 (UTC) 0 0 0

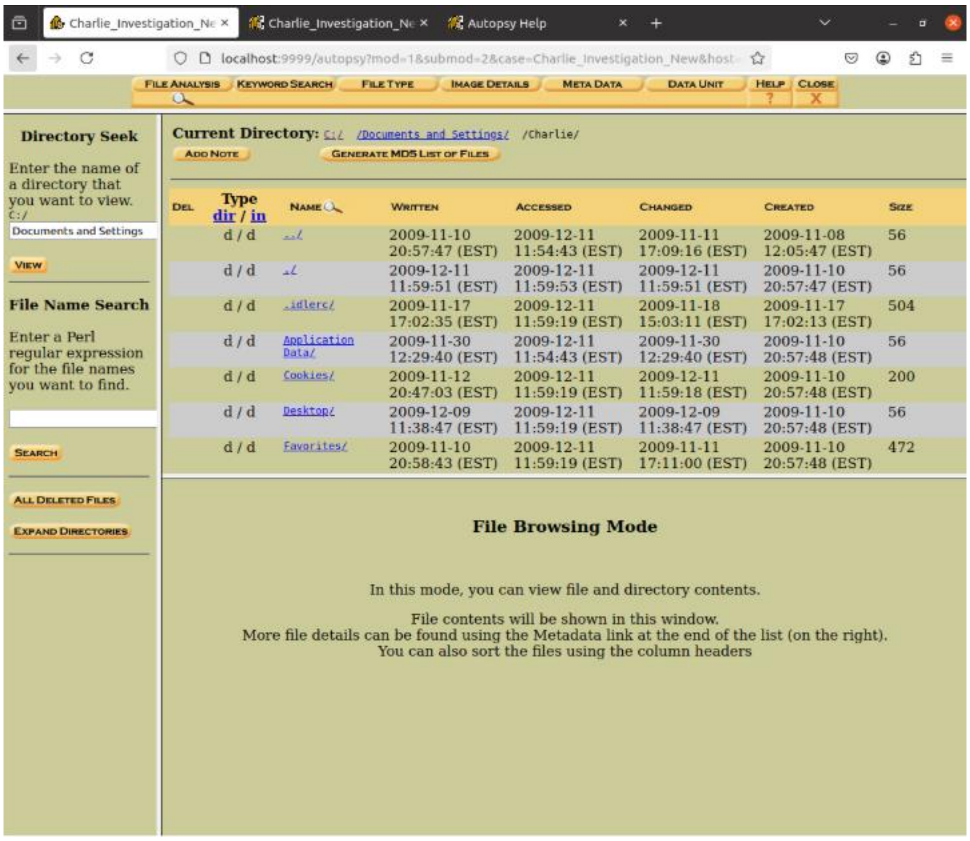
DEL	Type dir / in	NAME	WRITTEN	ACCESSED	CHA
	r / r	$AttrDef	2009-11-08 11:58:56 (EST)	2009-11-08 11:58:56 (EST)	20 11:
	d / d	$AVG/	2009-11-08 20:45:27 (EST)	2009-12-11 11:59:19 (EST)	20 17:
	r / r	$BadClus	2009-11-08 11:58:56 (EST)	2009-11-08 11:58:56 (EST)	20 11:
	r / r	$BadClus:$Bad	2009-11-08 11:58:56 (EST)	2009-11-08 11:58:56 (EST)	20 11:
	r / r	$Bitmap	2009-11-08 11:58:56 (EST)	2009-11-08 11:58:56 (EST)	20 11:
	r / r	$Boot	2009-11-08	2009-11-08	20

File Browsing Mode

In this mode, you can view file and directory contents.

File contents will be shown in this window.
More file details can be found using the Metadata link at the end of the list (on the right).
You can also sort the files using the column headers

Fig 5 : This screen shows the File Analysis section in Autopsy, where the contents of the selected directory are displayed. The interface provides a clear breakdown of files and folders, along with important metadata such as the dates they were written, accessed, and changed. On the left side, options like Directory Seek and File Name Search allow the investigator to navigate through specific folders or search for files using regular expressions. The directory listing on the right helps in browsing through system files, including orphan files and other key artifacts. This view confirms that Autopsy has successfully parsed the file system and is ready for deeper examination of the evidence.



Directory Seek

Enter the name of a directory that you want to view.
C:/
Documents and Settings

File Name Search

Enter a Perl regular expression for the file names you want to find.

Current Directory: C:/ /Documents and Settings/ /Charlie/

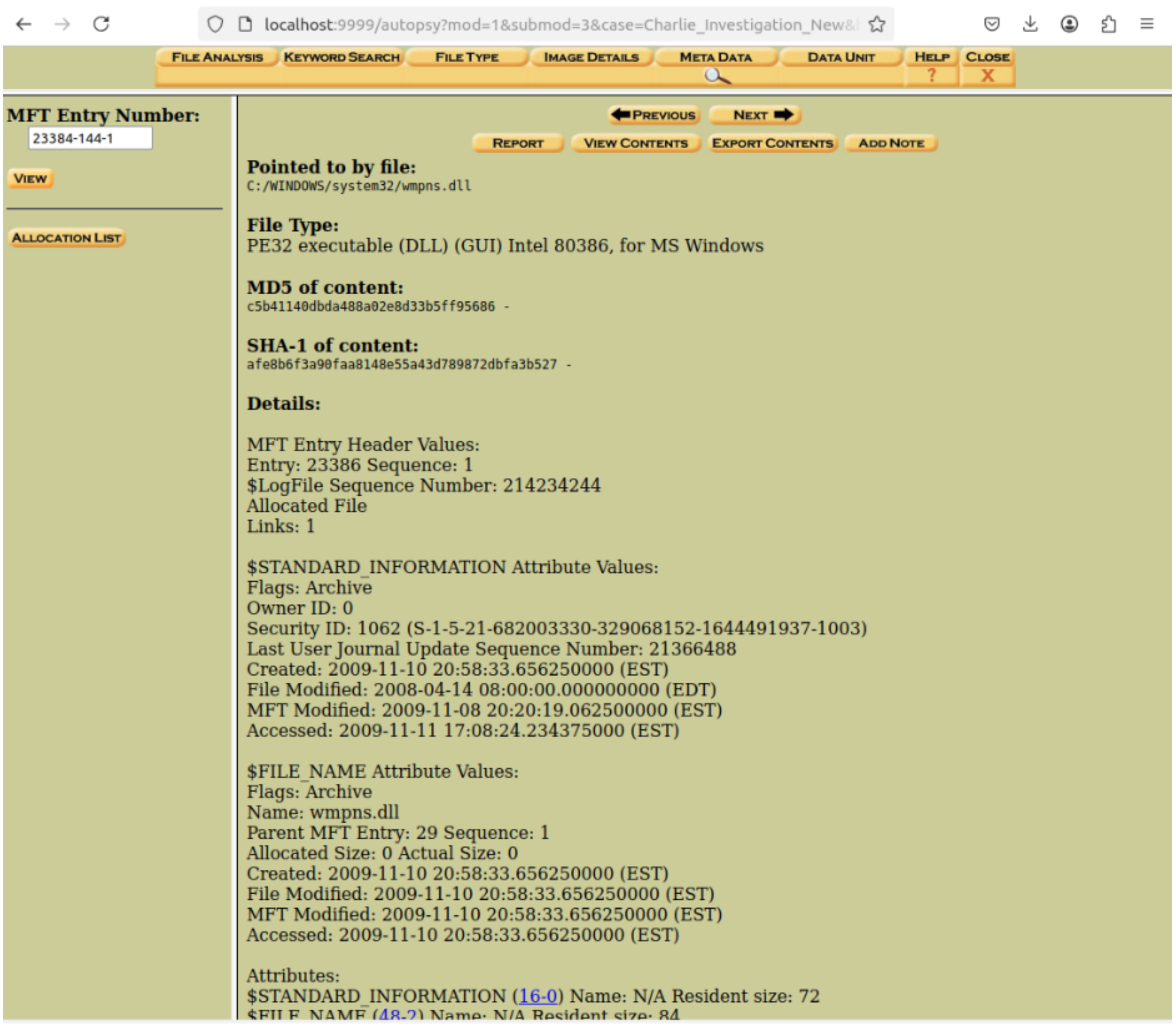
Del	Type dir / in	Name	Written	Accessed	Changed	Created	Size
	d / d	../	2009-11-10 20:57:47 (EST)	2009-12-11 11:54:43 (EST)	2009-11-11 17:09:16 (EST)	2009-11-08 12:05:47 (EST)	56
	d / d	./	2009-12-11 11:59:51 (EST)	2009-12-11 11:59:53 (EST)	2009-12-11 11:59:51 (EST)	2009-11-10 20:57:47 (EST)	56
	d / d	.idlerc/	2009-11-17 17:02:35 (EST)	2009-12-11 11:59:19 (EST)	2009-11-18 15:03:11 (EST)	2009-11-17 17:02:13 (EST)	504
	d / d	Application Data/	2009-11-30 12:29:40 (EST)	2009-12-11 11:54:43 (EST)	2009-11-30 12:29:40 (EST)	2009-11-10 20:57:48 (EST)	56
	d / d	Cookies/	2009-11-12 20:47:03 (EST)	2009-12-11 11:59:19 (EST)	2009-12-11 11:59:18 (EST)	2009-11-10 20:57:48 (EST)	200
	d / d	Desktop/	2009-12-09 11:38:47 (EST)	2009-12-11 11:59:19 (EST)	2009-12-09 11:38:47 (EST)	2009-11-10 20:57:48 (EST)	56
	d / d	Favorites/	2009-11-10 20:58:43 (EST)	2009-12-11 11:59:19 (EST)	2009-11-11 17:11:00 (EST)	2009-11-10 20:57:48 (EST)	472

File Browsing Mode

In this mode, you can view file and directory contents.

File contents will be shown in this window.
More file details can be found using the Metadata link at the end of the list (on the right).
You can also sort the files using the column headers

Fig 6 : This image shows the File Browsing Mode in Autopsy after navigating into the *Documents and Settings* directory under the user profile named *Charlie*. In this view, Autopsy provides a structured and chronological listing of the user's folders, including key directories such as *Application Data*, *Cookies*, *Desktop*, *Favorites*, and the hidden system folders. Each entry is accompanied by four essential timestamps—Written, Accessed, Changed, and Created—which offer valuable insights into user behavior and potential forensic events. For example, the Desktop and Favorites folders show timestamps clustered around mid-November 2009, which may indicate periods of active user activity. The left-hand sidebar provides additional controls for targeted navigation, allowing the investigator to manually enter directory paths or search for specific files using Perl-based regular expressions. This level of detail helps in reconstructing user interactions, tracking file modifications, and identifying suspicious changes. Overall, the interface confirms that Autopsy has successfully parsed the file system and is presenting the underlying directory structure in a way that supports thorough and methodical forensic examination.



MFT Entry Number: 23384-144-1

Pointed to by file:
`C:/WINDOWS/system32/wmpns.dll`

File Type:
PE32 executable (DLL) (GUI) Intel 80386, for MS Windows

MD5 of content:
`c5b41140dbda488a02e8d33b5ff95686 -`

SHA-1 of content:
`afe8b6f3a90faa8148e55a43d789872dbfa3b527 -`

Details:

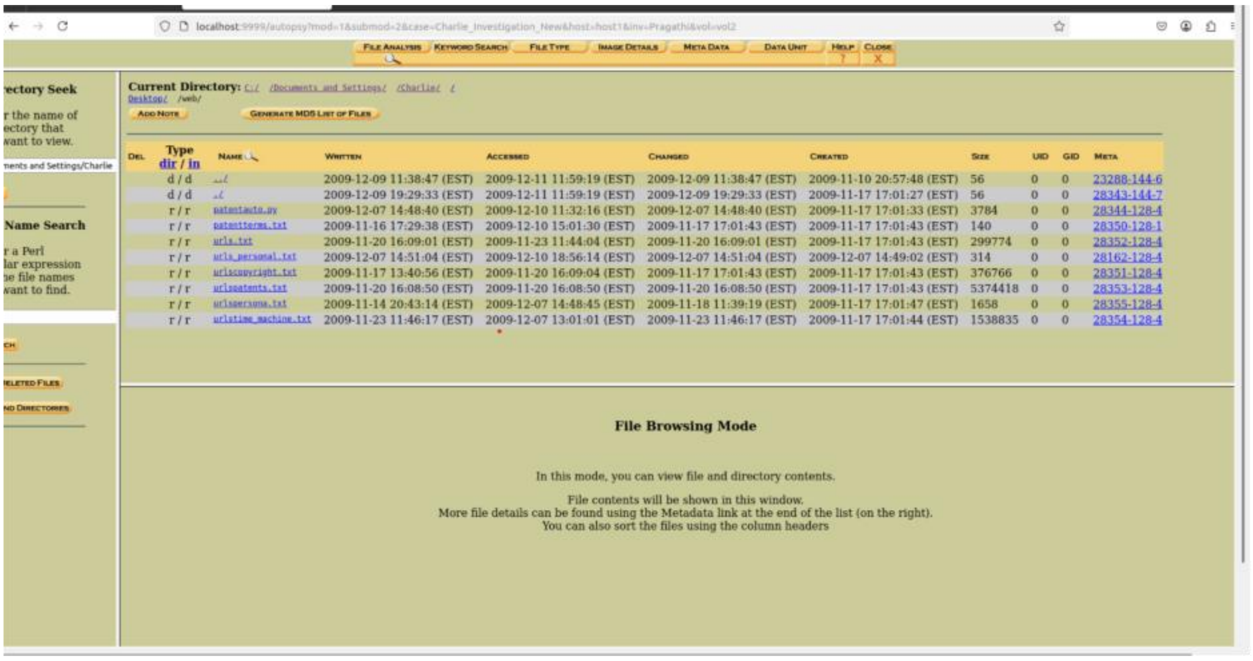
MFT Entry Header Values:
Entry: 23386 Sequence: 1
$LogFile Sequence Number: 214234244
Allocated File
Links: 1

$STANDARD_INFORMATION Attribute Values:
Flags: Archive
Owner ID: 0
Security ID: 1062 (S-1-5-21-682003330-329068152-1644491937-1003)
Last User Journal Update Sequence Number: 21366488
Created: 2009-11-10 20:58:33.656250000 (EST)
File Modified: 2008-04-14 08:00:00.000000000 (EDT)
MFT Modified: 2009-11-08 20:20:19.062500000 (EST)
Accessed: 2009-11-11 17:08:24.234375000 (EST)

$FILE_NAME Attribute Values:
Flags: Archive
Name: wmpns.dll
Parent MFT Entry: 29 Sequence: 1
Allocated Size: 0 Actual Size: 0
Created: 2009-11-10 20:58:33.656250000 (EST)
File Modified: 2009-11-10 20:58:33.656250000 (EST)
MFT Modified: 2009-11-10 20:58:33.656250000 (EST)
Accessed: 2009-11-10 20:58:33.656250000 (EST)

Attributes:
$STANDARD_INFORMATION (16-0) Name: N/A Resident size: 72
$FILE_NAME (48-2) Name: N/A Resident size: 84

Fig 7 : This screenshot shows the Metadata view in Autopsy for the file *wmpns.dll*, located in the **C:\WINDOWS\system32** directory. Autopsy identifies the file as a PE32 executable (DLL) for Windows systems and provides cryptographic hash values, including the MD5 and SHA-1 hashes, which are essential for verifying file integrity and detecting tampering. The metadata section also displays detailed NTFS information such as the Master File Table (MFT) entry number, sequence value, log file sequence number, allocation status, and link count. Additionally, Autopsy reveals the file's timestamp information under the STANDARD_INFORMATION and FILE_NAME attributes. These timestamps include the created, modified, accessed, and MFT-modified times, which help investigators trace the file's activity timeline. For example, the file was last modified in 2008 but accessed again in November 2009, indicating system or user activity involving this DLL. This level of metadata detail is crucial in digital forensics, as it supports timeline reconstruction, integrity verification, and detection of unusual or suspicious file behavior.



Current Directory: C:/ /Documents and Settings/ /Charlie/ / Desktop/ /web/

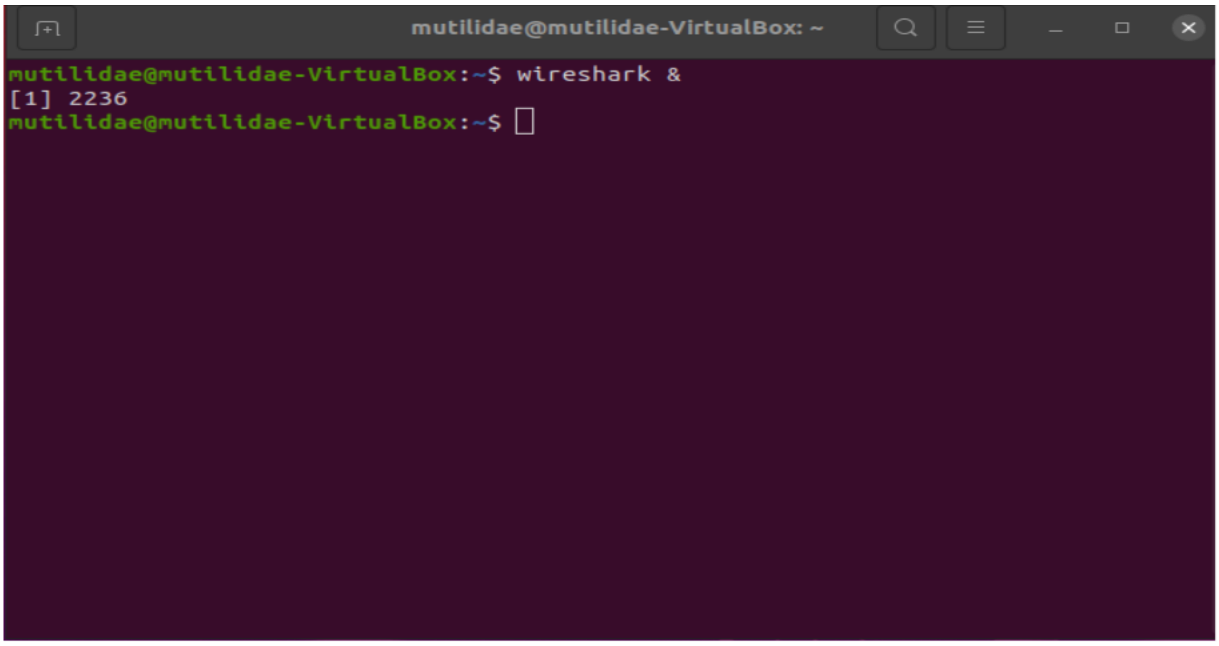
Del	Type dir / in	Name	Written	Accessed	Changed	Created	Size	UID	GID	Meta
	d / d	../	2009-12-09 11:38:47 (EST)	2009-12-11 11:59:19 (EST)	2009-12-09 11:38:47 (EST)	2009-11-10 20:57:48 (EST)	56	0	0	23288-144-6
	d / d	./	2009-12-09 19:29:33 (EST)	2009-12-11 11:59:19 (EST)	2009-12-09 19:29:33 (EST)	2009-11-17 17:01:27 (EST)	56	0	0	28343-144-7
	r / r	patentauto.py	2009-12-07 14:48:40 (EST)	2009-12-10 11:32:16 (EST)	2009-12-07 14:48:40 (EST)	2009-11-17 17:01:33 (EST)	3784	0	0	28344-128-4
	r / r	patentterms.txt	2009-11-16 17:29:38 (EST)	2009-12-10 15:01:30 (EST)	2009-11-17 17:01:43 (EST)	2009-11-17 17:01:43 (EST)	140	0	0	28350-128-1
	r / r	urls.txt	2009-11-20 16:09:01 (EST)	2009-11-23 11:44:04 (EST)	2009-11-20 16:09:01 (EST)	2009-11-17 17:01:43 (EST)	299774	0	0	28352-128-4
	r / r	urls_personal.txt	2009-12-07 14:51:04 (EST)	2009-12-10 18:56:14 (EST)	2009-12-07 14:51:04 (EST)	2009-12-07 14:49:02 (EST)	314	0	0	28162-128-4
	r / r	urlscopyright.txt	2009-11-17 13:40:56 (EST)	2009-11-20 16:09:04 (EST)	2009-11-17 17:01:43 (EST)	2009-11-17 17:01:43 (EST)	376766	0	0	28351-128-4
	r / r	urlspatents.txt	2009-11-20 16:08:50 (EST)	2009-11-20 16:08:50 (EST)	2009-11-20 16:08:50 (EST)	2009-11-17 17:01:43 (EST)	5374418	0	0	28353-128-4
	r / r	urlspersona.txt	2009-11-14 20:43:14 (EST)	2009-12-07 14:48:45 (EST)	2009-11-18 11:39:19 (EST)	2009-11-17 17:01:47 (EST)	1658	0	0	28355-128-4
	r / r	urlstime_machine.txt	2009-11-23 11:46:17 (EST)	2009-12-07 13:01:01 (EST)	2009-11-23 11:46:17 (EST)	2009-11-17 17:01:44 (EST)	1538835	0	0	28354-128-4

File Browsing Mode

In this mode, you can view file and directory contents.

File contents will be shown in this window.
More file details can be found using the Metadata link at the end of the list (on the right).
You can also sort the files using the column headers

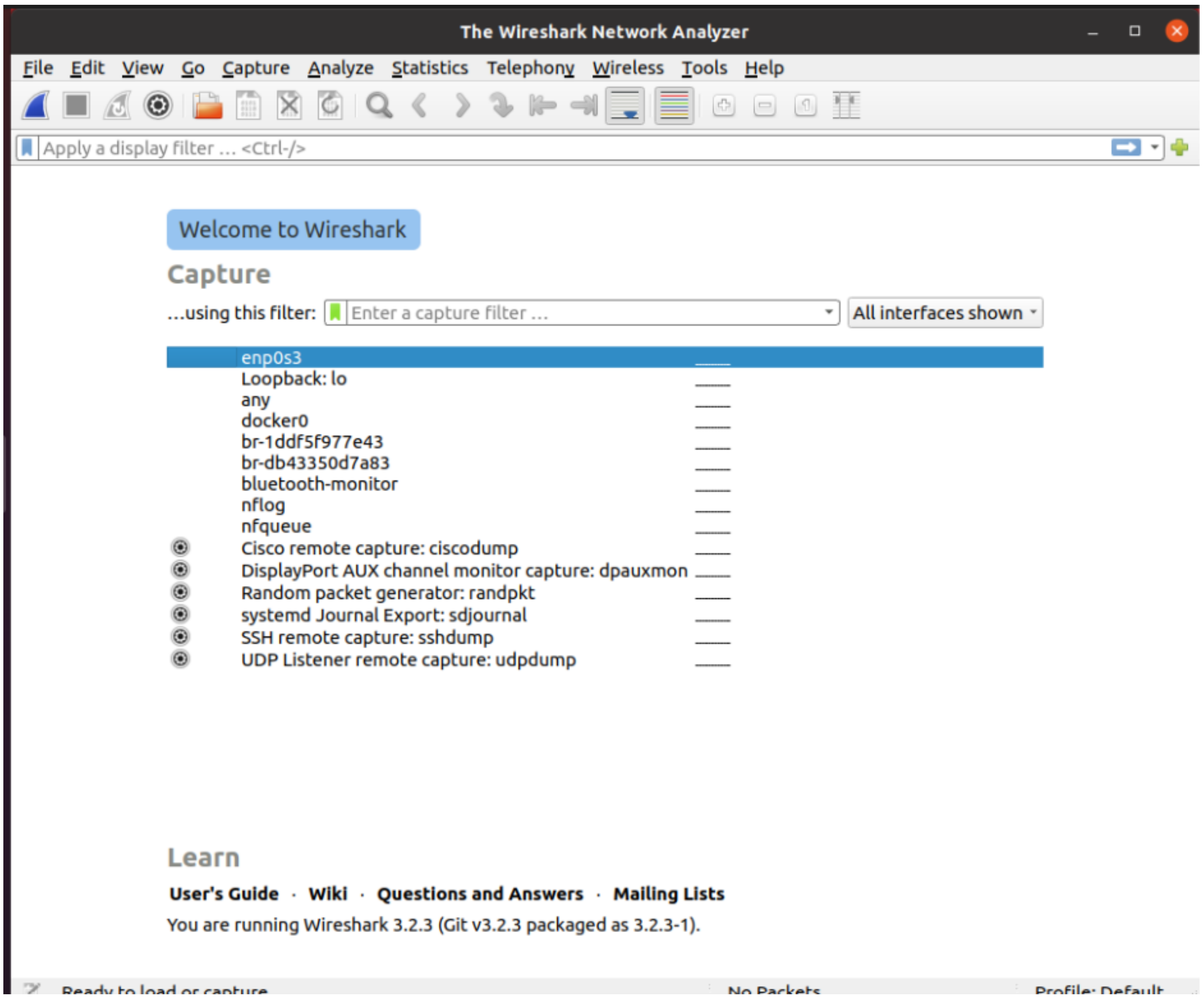
Fig 8 : This screenshot displays the File Browsing Mode in Autopsy after navigating to the *Documents and Settings → Charlie → Desktop* directory. The interface shows a structured list of files present on the user's desktop, including items such as *autemeta.csv*, *autentats.txt*, *urls.txt*, *urls_personal.txt*, and several system-generated files. For each entry, Autopsy provides detailed timestamp information—Written, Accessed, Changed, and Created—allowing the investigator to track user activity and identify any unusual file behavior. For example, several files show closely grouped timestamps from late November 2009, which may indicate active usage or recent modifications by the user. The rightmost column includes links to the file metadata, giving the investigator an easy way to explore deeper information such as file hashes, MFT values, and security identifiers. This view confirms that the user's desktop environment has been successfully parsed and provides valuable insights into the files the user interacted with, supporting further analysis or timeline reconstruction.



```
mutilidae@mutilidae-VirtualBox:~$ wireshark &
[1] 2236
mutilidae@mutilidae-VirtualBox:~$
```

Fig 9 : This screenshot shows the Wireshark network analysis tool being launched from the terminal within the Ubuntu VirtualBox environment. The command wireshark & was used, which starts the application in the background and immediately returns control to the terminal. The terminal output displays the process ID (PID 2236), confirming that Wireshark has successfully started running in the background. This method allows the user to continue issuing terminal commands while the graphical Wireshark interface opens separately for packet capture and analysis.

NETWORK ANALYSIS SCREENSHOTS:

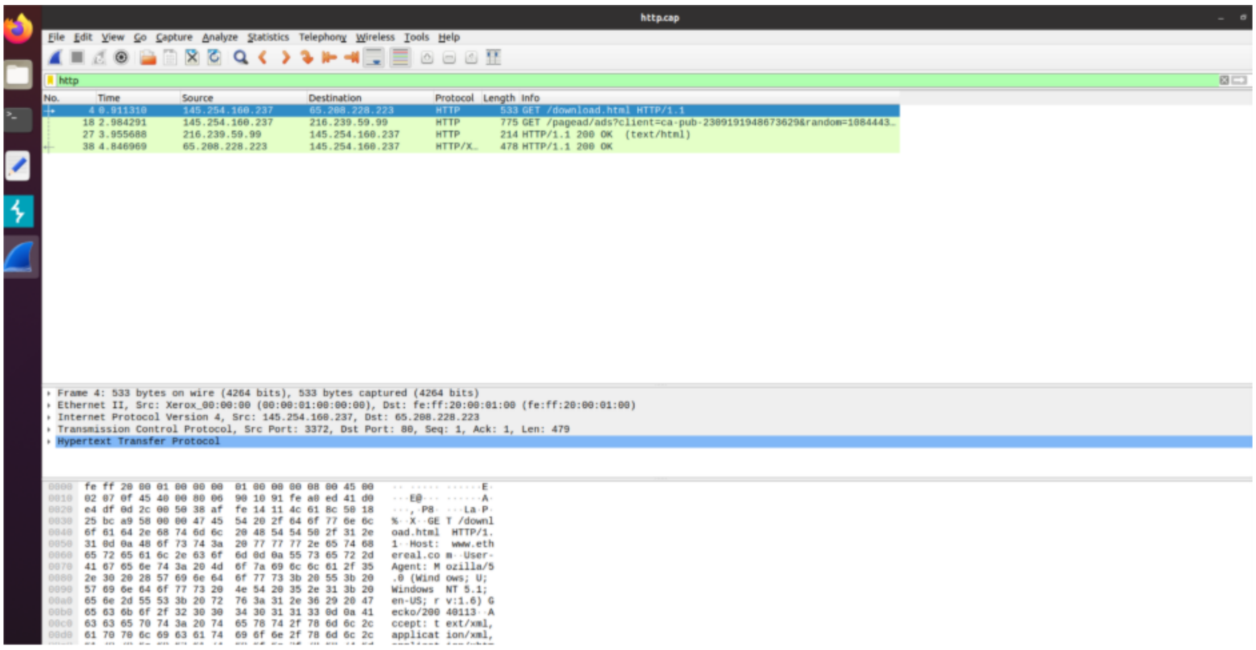

The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Welcome to Wireshark
Capture
…using this filter:
Enter a capture filter …
All interfaces shown
Learn
User's Guide · Wiki · Questions and Answers · Mailing Lists

enp0s3
Loopback: lo
any
docker0
br-1ddf5f977e43
br-db43350d7a83
bluetooth-monitor
nflog
nfqueue
Cisco remote capture: ciscodump
DisplayPort AUX channel monitor capture: dpauxmon
Random packet generator: randpkt
systemd Journal Export: sdjournal
SSH remote capture: sshdump
UDP Listener remote capture: udpdump

You are running Wireshark 3.2.3 (Git v3.2.3 packaged as 3.2.3-1).

Ready to load or capture · No Packets · Profile: Default

Fig 10 : This screenshot shows the main interface of Wireshark after it has been successfully launched on the Ubuntu VirtualBox system. The application is displaying the available network interfaces that can be used for packet capture, such as *enp0s3*, *Loopback*, *docker0*, various virtual bridges, and remote capture options like *sshdump* and *udpdump*. The highlighted interface *enp0s3* represents the primary network adapter commonly used for capturing live traffic on the virtual machine. At the top, Wireshark provides an option to enter a capture filter, allowing targeted traffic collection based on protocols or IP addresses. This screen confirms that Wireshark is fully operational and ready to begin monitoring network activity for detailed packet analysis.

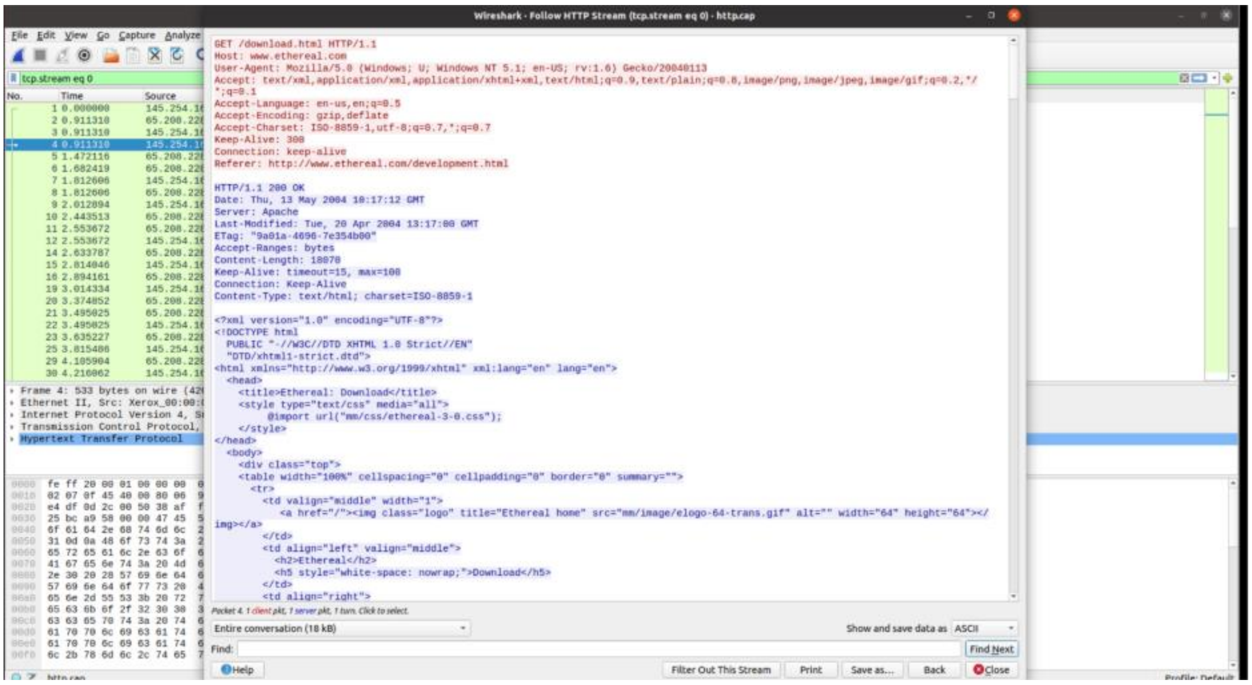

http.cap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http

```
No.  Time        Source           Destination      Protocol Length Info
   4 0.911310    145.254.160.237  65.208.228.223   HTTP       533 GET /download.html HTTP/1.1
  18 2.984291    145.254.160.237  216.239.59.99    HTTP       775 GET /pagead/ads?client=ca-pub-2309191948673629&random=1084443…
  27 3.955688    216.239.59.99    145.254.160.237  HTTP       214 HTTP/1.1 200 OK  (text/html)
  38 4.846969    65.208.228.223   145.254.160.237  HTTP/X…    478 HTTP/1.1 200 OK
```

```
▸ Frame 4: 533 bytes on wire (4264 bits), 533 bytes captured (4264 bits)
▸ Ethernet II, Src: Xerox_00:00:00 (00:00:01:00:00:00), Dst: fe:ff:20:00:01:00 (fe:ff:20:00:01:00)
▸ Internet Protocol Version 4, Src: 145.254.160.237, Dst: 65.208.228.223
▸ Transmission Control Protocol, Src Port: 3372, Dst Port: 80, Seq: 1, Ack: 1, Len: 479
▸ Hypertext Transfer Protocol
```

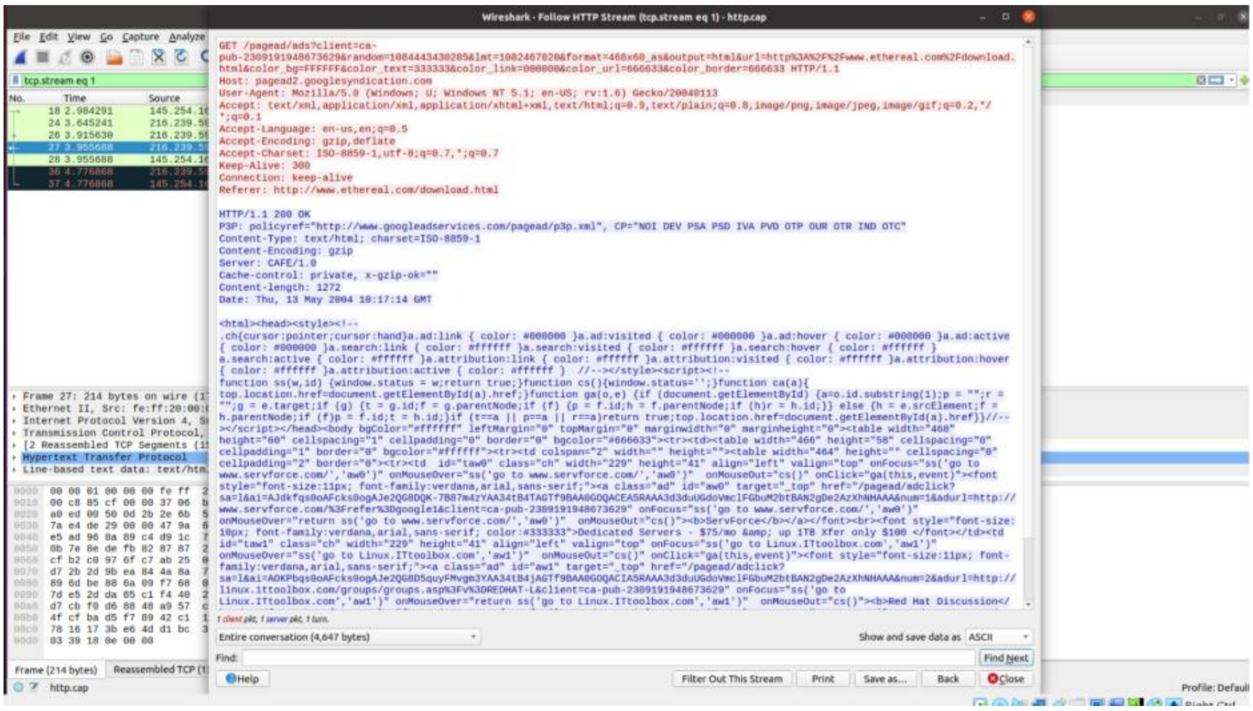
```
0000  fe ff 20 00 01 00 00 00  01 00 00 00 08 00 45 00   ·· ····· ······E·
0010  02 07 0f 45 40 00 80 06  90 10 91 fe a0 ed 41 d0   ···E@··· ······A·
0020  e4 df 0d 2c 00 50 38 af  fe 14 11 4c 61 8c 50 18   ···,·P8· ···La·P·
0030  25 bc a9 58 00 00 47 45  54 20 2f 64 6f 77 6e 6c   %··X··GE T /downl
0040  6f 61 64 2e 68 74 6d 6c  20 48 54 54 50 2f 31 2e   oad.html  HTTP/1.
0050  31 0d 0a 48 6f 73 74 3a  20 77 77 77 2e 65 74 68   1··Host:  www.eth
0060  65 72 65 61 6c 2e 63 6f  6d 0d 0a 55 73 65 72 2d   ereal.co m··User-
0070  41 67 65 6e 74 3a 20 4d  6f 7a 69 6c 6c 61 2f 35   Agent: M ozilla/5
0080  2e 30 20 28 57 69 6e 64  6f 77 73 3b 20 55 3b 20   .0 (Wind ows; U;
0090  57 69 6e 64 6f 77 73 20  4e 54 20 35 2e 31 3b 20   Windows  NT 5.1;
00a0  65 6e 2d 55 53 3b 20 72  76 3a 31 2e 36 29 20 47   en-US; r v:1.6) G
00b0  65 63 6b 6f 2f 32 30 30  34 30 31 31 33 0d 0a 41   ecko/200 40113··A
00c0  63 63 65 70 74 3a 20 74  65 78 74 2f 78 6d 6c 2c   ccept: t ext/xml,
00d0  61 70 70 6c 69 63 61 74  69 6f 6e 2f 78 6d 6c 2c   applicat ion/xml,
```

Fig 11 : This screenshot shows a Wireshark capture session where HTTP traffic has been successfully recorded and analyzed. The upper pane lists individual packets, displaying key details such as the timestamp, source and destination IP addresses, protocol used, and summary of the HTTP request or response. In this capture, multiple HTTP packets are visible, including a GET request for a file named *download.html* and another request fetching content from a remote server. The middle pane provides a structured breakdown of the selected packet, showing Ethernet, IP, TCP, and HTTP header information, which helps in understanding the flow of communication between the client and server. The bottom pane displays the raw packet data in hexadecimal and ASCII formats, revealing underlying request details such as user-agent information and content types. This view confirms that Wireshark is not only capturing network traffic but also decoding application-layer data, allowing for deeper inspection of web interactions and potential anomalies.


Wireshark · Follow HTTP Stream (tcp.stream eq 0) · http.cap
Packet 4. 1 client pkt, 1 server pkt, 1 turn. Click to select.
Entire conversation (18 kB)
Show and save data as ASCII
Find:
Find Next
Help
Filter Out This Stream
Print
Save as...
Back
Close

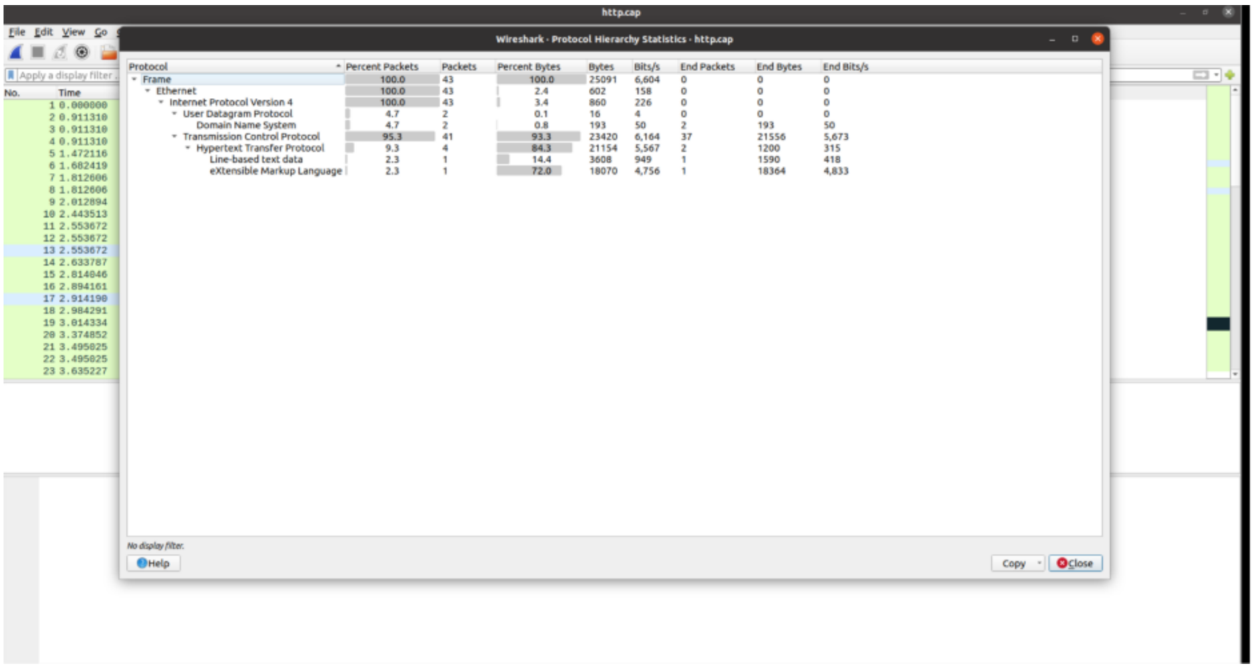
```
GET /download.html HTTP/1.1
Host: www.ethereal.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.6) Gecko/20040113
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,image/jpeg,image/gif;q=0.2,*/
*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.ethereal.com/development.html

HTTP/1.1 200 OK
Date: Thu, 13 May 2004 10:17:12 GMT
Server: Apache
Last-Modified: Tue, 20 Apr 2004 13:17:00 GMT
ETag: "9a01a-4696-7e354b00"
Accept-Ranges: bytes
Content-Length: 18070
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=ISO-8859-1

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Ethereal: Download</title>
    <style type="text/css" media="all">
         @import url("mm/css/ethereal-3-0.css");
    </style>
</head>
  <body>
    <div class="top">
    <table width="100%" cellspacing="0" cellpadding="0" border="0" summary="">
      <tr>
        <td valign="middle" width="1">
           <a href="/"><img class="logo" title="Ethereal home" src="mm/image/elogo-64-trans.gif" alt="" width="64" height="64"></
img></a>
        </td>
        <td align="left" valign="middle">
          <h2>Ethereal</h2>
          <h5 style="white-space: nowrap;">Download</h5>
        </td>
        <td align="right">
```

Fig 12 : This screenshot shows the “Follow HTTP Stream” feature in Wireshark, which reconstructs and displays an entire HTTP conversation between the client and server. The left panel lists the captured packets, while the main window reveals the full request and response content in a readable format. In this case, the client made a GET request for *download.html*, including browser details, accepted content types, and the referrer information. The server’s response, shown in blue text, includes HTTP headers such as the date, content type, encoding, and caching details, followed by the actual HTML content of the webpage. Wireshark automatically stitches these pieces together, allowing the investigator to view the entire webpage source code as it was transmitted over the network. This feature is especially useful for analyzing web-based activity, identifying sensitive data leakage, and understanding exactly what information was exchanged during the session.


Wireshark · Follow HTTP Stream (tcp.stream eq 1) · http.cap
1 client pkt, 1 server pkt, 1 turn.
Entire conversation (4,647 bytes)
Show and save data as ASCII
Find:
Find Next
Help
Filter Out This Stream
Print
Save as...
Back
Close

```
GET /pagead/ads?client=ca-
pub-2309191948673629&random=1084443430285&lmt=1082467020&format=468x60_as&output=html&url=http%3A%2F%2Fwww.ethereal.com%2Fdownload.
html&color_bg=FFFFFF&color_text=333333&color_link=000000&color_url=666633&color_border=666633 HTTP/1.1
Host: pagead2.googlesyndication.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.6) Gecko/20040113
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,image/jpeg,image/gif;q=0.2,*/
*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.ethereal.com/download.html

HTTP/1.1 200 OK
P3P: policyref="http://www.googleadservices.com/pagead/p3p.xml", CP="NOI DEV PSA PSD IVA PVD OTP OUR OTR IND OTC"
Content-Type: text/html; charset=ISO-8859-1
Content-Encoding: gzip
Server: CAFE/1.0
Cache-control: private, x-gzip-ok=""
Content-length: 1272
Date: Thu, 13 May 2004 10:17:14 GMT

<html><head><style><!--
.ch{cursor:pointer;cursor:hand}a.ad:link { color: #000000 }a.ad:visited { color: #000000 }a.ad:hover { color: #000000 }a.ad:active
{ color: #000000 }a.search:link { color: #ffffff }a.search:visited { color: #ffffff }a.search:hover { color: #ffffff }
a.search:active { color: #ffffff }a.attribution:link { color: #ffffff }a.attribution:visited { color: #ffffff }a.attribution:hover
{ color: #ffffff }a.attribution:active { color: #ffffff }  //--></style><script><!--
function ss(w,id) {window.status = w;return true;}function cs(){window.status='';}function ca(a){
top.location.href=document.getElementById(a).href;}function ga(o,e) {if (document.getElementById) {a=o.id.substring(1);p = "";r =
"";g = e.target;if (g) {t = g.id;f = g.parentNode;if (f) {p = f.id;h = f.parentNode;if (h)r = h.id;}} else {h = e.srcElement;f =
h.parentNode;if (f)p = f.id;t = h.id;}if (t==a || p==a || r==a)return true;top.location.href=document.getElementById(a).href}}//--
></script></head><body bgColor="#ffffff" leftMargin="0" topMargin="0" marginwidth="0" marginheight="0"><table width="468"
height="60" cellspacing="1" cellpadding="0" border="0" bgcolor="#666633"><tr><td><table width="466" height="58" cellspacing="0"
cellpadding="1" border="0" bgcolor="#ffffff"><tr><td colspan="2" width="" height=""><table width="464" height="" cellspacing="0"
cellpadding="2" border="0"><tr><td  id="taw0" class="ch" width="229" height="41" align="left" valign="top" onFocus="ss('go to
www.servforce.com/','aw0')" onMouseOver="ss('go to www.servforce.com/','aw0')"  onMouseOut="cs()" onClick="ga(this,event)"><font
style="font-size:11px; font-family:verdana,arial,sans-serif;"><a class="ad" id="aw0" target="_top" href="/pagead/adclick?
sa=l&ai=AJdkfqs0oAFcks0ogAJe2QG8DQK-7B87m4zYAA34tB4TAGTf9BAA0GDQACEA5RAAA3d3duUGdoVmclFGbuM2btBAN2gDe2AzXhNHAAA&num=1&adurl=http://
www.servforce.com/%3Frefer%3Dgoogle1&client=ca-pub-2309191948673629" onFocus="ss('go to www.servforce.com/','aw0')"
onMouseOver="return ss('go to www.servforce.com/','aw0')"  onMouseOut="cs()"><b>ServForce</b></a></font><br><font style="font-size:
10px; font-family:verdana,arial,sans-serif; color:#333333">Dedicated Servers - $75/mo &amp; up 1TB Xfer only $100 </font></td><td
id="taw1" class="ch" width="229" height="41" align="left" valign="top" onFocus="ss('go to Linux.ITtoolbox.com','aw1')"
onMouseOver="ss('go to Linux.ITtoolbox.com','aw1')"  onMouseOut="cs()" onClick="ga(this,event)"><font style="font-size:11px; font-
family:verdana,arial,sans-serif;"><a class="ad" id="aw1" target="_top" href="/pagead/adclick?
sa=l&ai=AOKPbqs0oAFcks0ogAJe2QG8D5quyFMvgm3YAA34tB4jAGTf9BAA0GDQACIA5RAAA3d3duUGdoVmclFGbuM2btBAN2gDe2AzXhNHAAA&num=2&adurl=http://
linux.ittoolbox.com/groups/groups.asp%3Fv%3DREDHAT-L&client=ca-pub-2309191948673629" onFocus="ss('go to
Linux.ITtoolbox.com','aw1')" onMouseOver="return ss('go to Linux.ITtoolbox.com','aw1')"  onMouseOut="cs()"><b>Red Hat Discussion</
```

Fig 13 : This screenshot shows another view of Wireshark’s “Follow HTTP Stream” feature, capturing the complete HTTP communication between a client and a web server. The red text represents the HTTP request sent by the client, including the GET request for a webpage accessed through a tracking link, along with browser details, accepted content types, language preferences, encoding, and referrer information. The blue text displays the server’s HTTP 200 OK response, which includes headers such as content type, server identity, caching policies, and the content length. Below the headers, the actual HTML and CSS code returned by the server is shown in full, including embedded JavaScript, stylesheets, and page content. This level of visibility allows the investigator to examine exactly what data was transferred during the session, making it easier to identify potential security issues, track user activity, and reconstruct the webpage as it originally appeared. Overall, this view demonstrates how Wireshark can reconstruct an entire web transaction, offering deep insights into network communication patterns and web-based interactions.

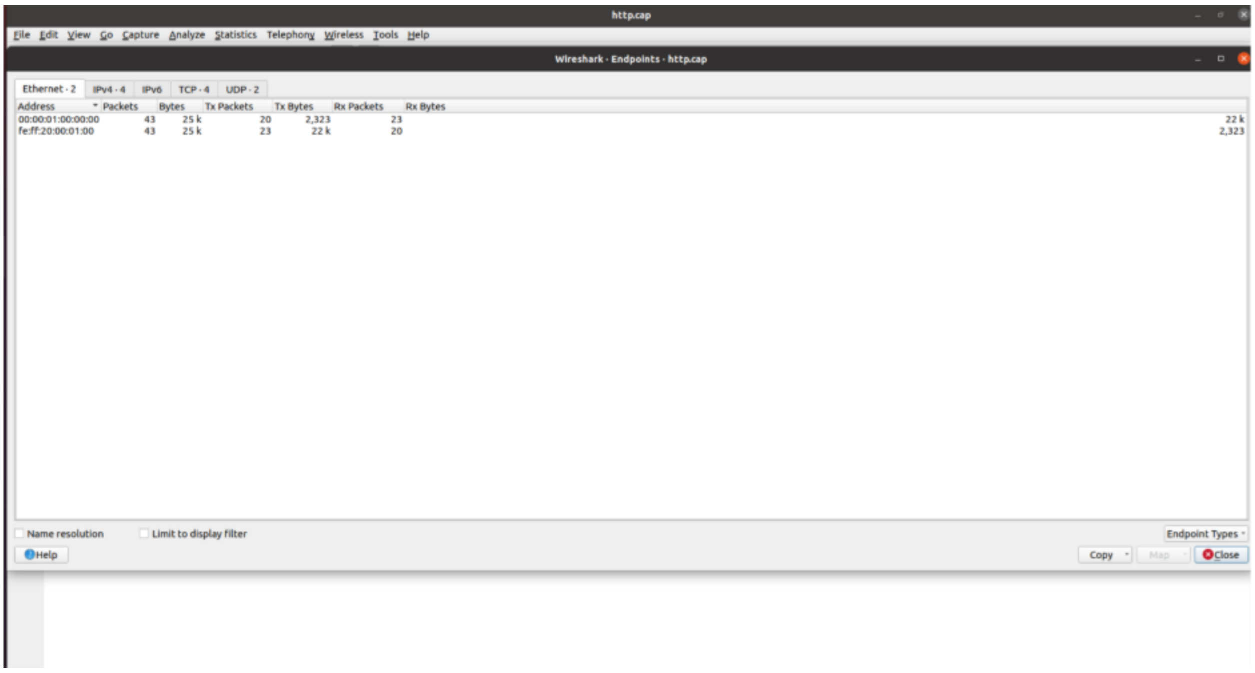


Wireshark · Protocol Hierarchy Statistics · http.cap

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes	End Bits/s
Frame	100.0	43	100.0	25091	6,604	0	0	0
Ethernet	100.0	43	2.4	602	158	0	0	0
Internet Protocol Version 4	100.0	43	3.4	860	226	0	0	0
User Datagram Protocol	4.7	2	0.1	16	4	0	0	0
Domain Name System	4.7	2	0.8	193	50	2	193	50
Transmission Control Protocol	95.3	41	93.3	23420	6,164	37	21556	5,673
Hypertext Transfer Protocol	9.3	4	84.3	21154	5,567	2	1200	315
Line-based text data	2.3	1	14.4	3608	949	1	1590	418
eXtensible Markup Language	2.3	1	72.0	18070	4,756	1	18364	4,833

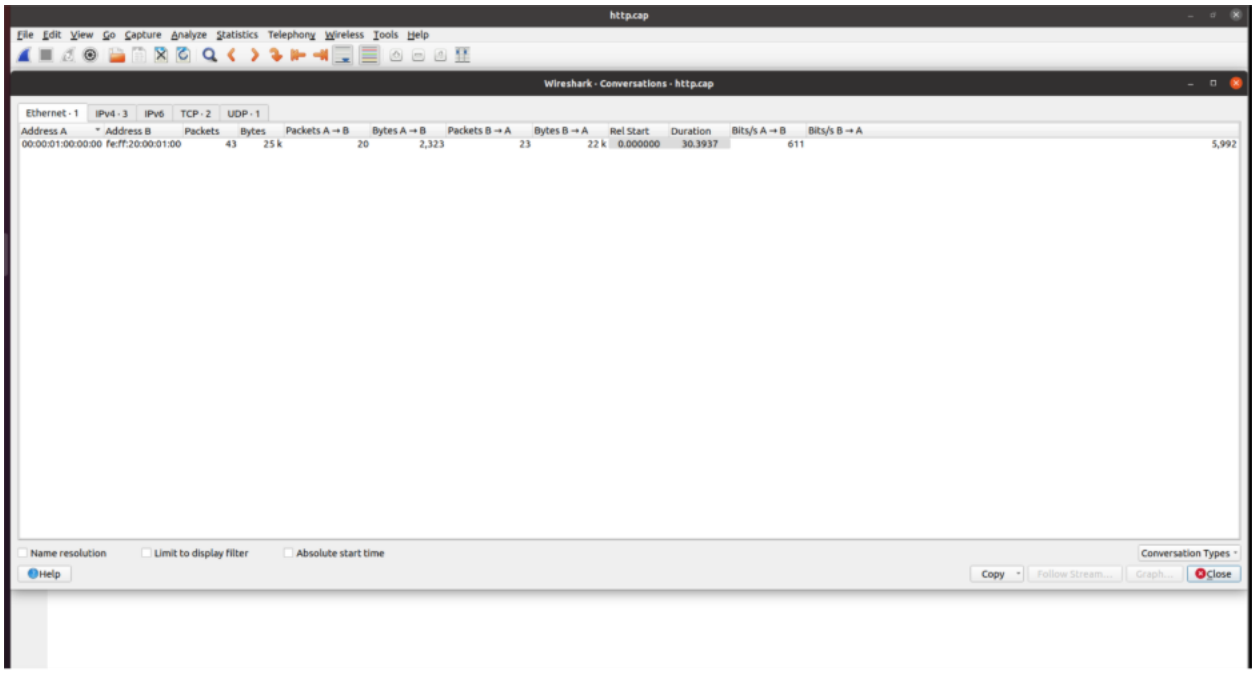
No display filter.

Fig 14 : This screenshot displays the Protocol Hierarchy Statistics window in Wireshark, which provides a breakdown of all the protocols detected in the captured network traffic. The analysis shows that all packets were transmitted over Ethernet, with most of the traffic consisting of IPv4 packets. Within the IPv4 traffic, the majority belongs to the Transmission Control Protocol (TCP), which makes up 93.3% of the packets and accounts for most of the captured bytes. A smaller portion of traffic is associated with Domain Name System (DNS) queries and User Datagram Protocol (UDP) packets. Hypertext Transfer Protocol (HTTP) traffic is also highlighted, reflecting the web-based activity captured during the session. The statistics offer insight into the overall communication structure, showing how different layers and protocols contributed to the network interaction. This hierarchical view helps investigators quickly understand the dominant traffic types and identify any unusual or unexpected protocols in the capture.



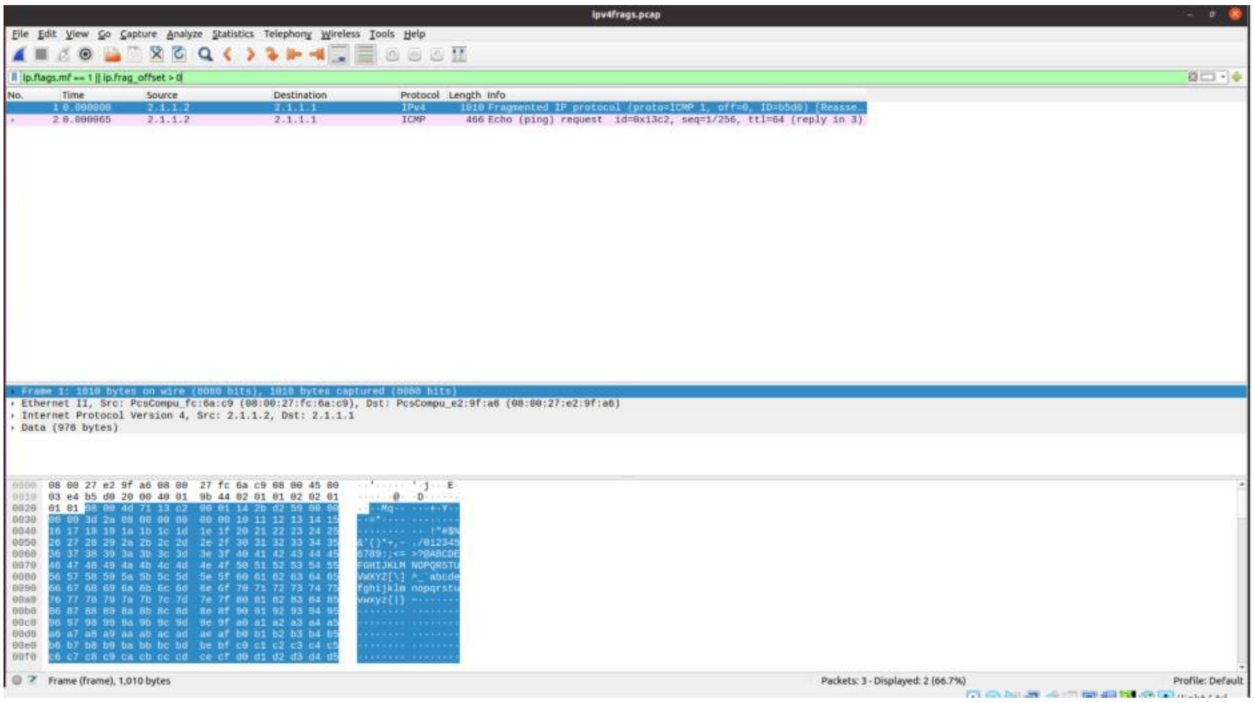
Address	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes
00:00:01:00:00:00	43	25 k	20	2,323	23	22 k
fe:ff:20:00:01:00	43	25 k	23	22 k	20	2,323

Fig 15 : This screenshot from Wireshark shows the Endpoints statistics window, which provides a summary of all network endpoints involved in the captured traffic. The table lists the MAC addresses under the Ethernet tab, along with the number of packets sent and received (Tx and Rx) and the total amount of data transferred in bytes. In this capture, two Ethernet endpoints are active, each transmitting and receiving 43 packets, indicating a complete and balanced exchange between the source and destination. The corresponding byte counts show that approximately 25 KB of data was transmitted and around 22–23 KB was received, illustrating the flow of network communication during the packet capture session. This information helps investigators identify which devices or interfaces participated in the traffic and measure their contribution to the overall network activity.



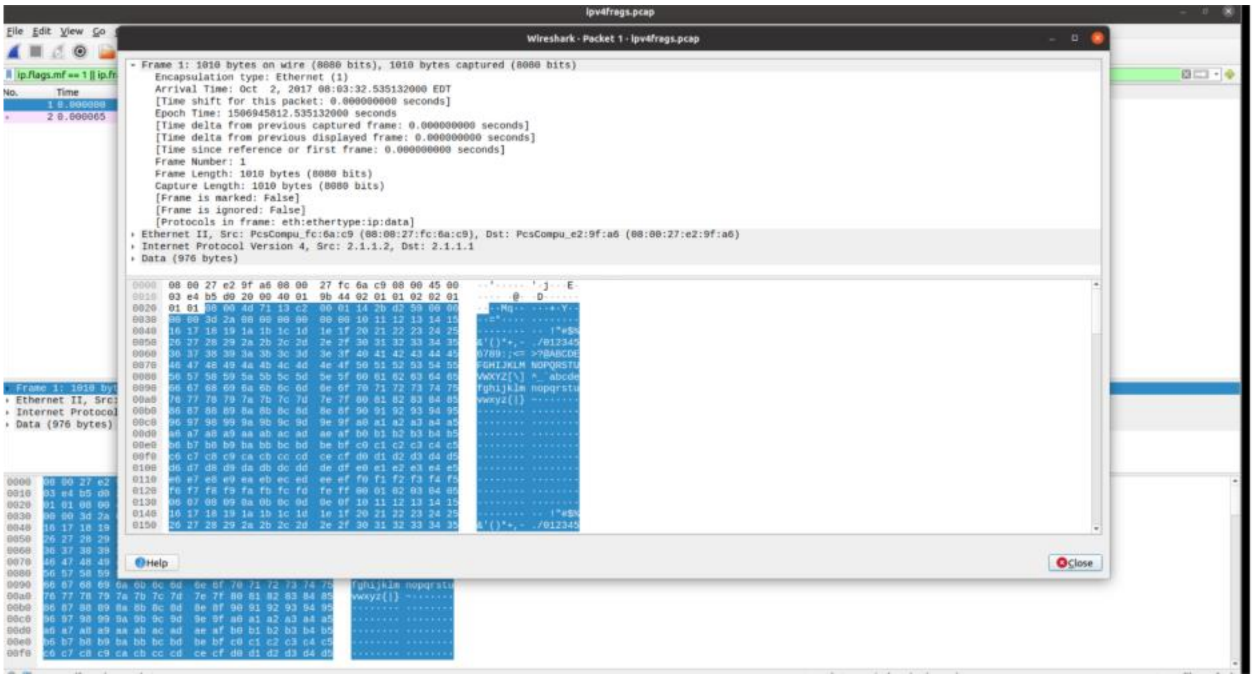
Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
00:00:01:00:00:00	fe:ff:20:00:01:00	43	25 k	20	2,323	23	22 k	0.000000	30.3937	611	5,992

Fig 16 : This screenshot shows the Wireshark "Conversations" window, which provides a detailed overview of communication between endpoints during the packet capture. The Ethernet tab is selected, displaying a single conversation between two MAC addresses. The table summarizes the flow of traffic in both directions, including the number of packets transmitted (A → B and B → A), the total bytes exchanged, and the overall duration of the conversation. In this capture, 43 packets totaling around 25 KB were sent from Address A to Address B, while 23 packets amounting to approximately 2.3 KB were transmitted in the opposite direction. Wireshark also calculates the bit rates for each direction, helping identify bandwidth usage and data transfer patterns. This view is especially useful for understanding how two devices communicated over the network, confirming the completeness of the session, and detecting any unusual or asymmetric data exchanges.



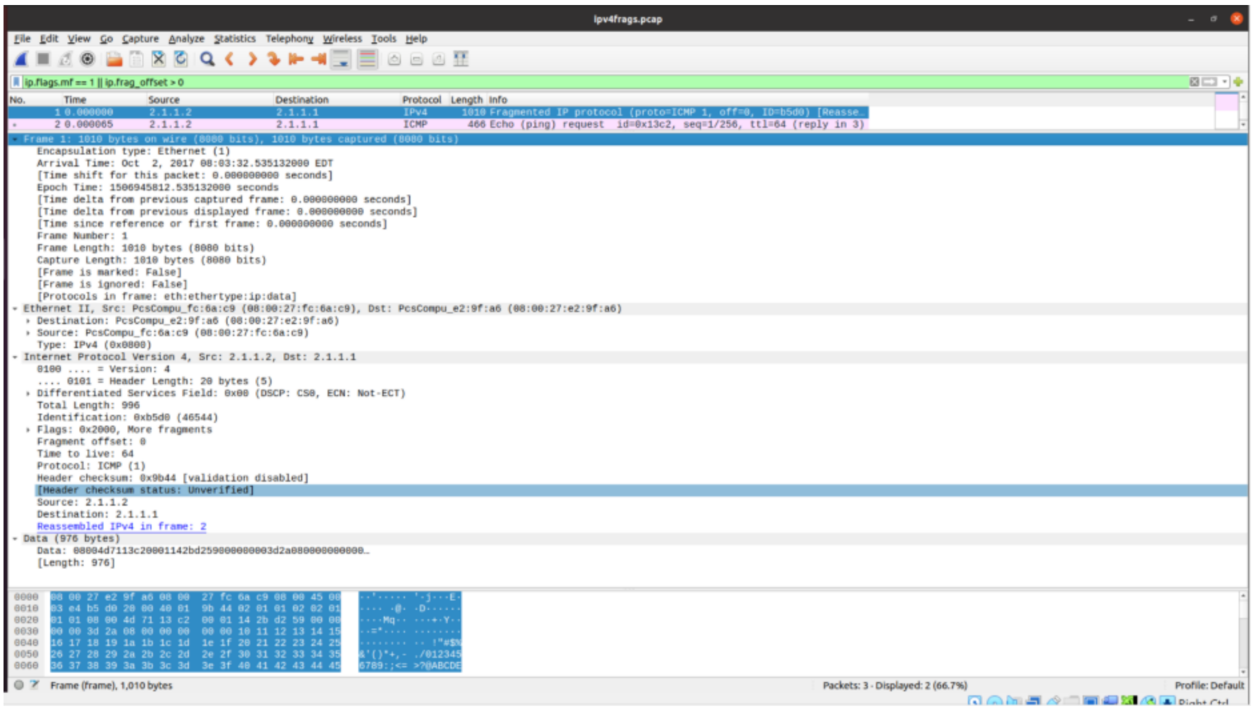
```
ipv4frags.pcap

ip.flags.mf == 1 || ip.frag_offset > 0

No.   Time        Source    Destination  Protocol  Length  Info
    1 0.000000    2.1.1.2   2.1.1.1      IPv4      1010 Fragmented IP protocol (proto=ICMP 1, off=0, ID=b5d0) (Reasse…
    2 0.000065    2.1.1.2   2.1.1.1      ICMP       466 Echo (ping) request  id=0x13c2, seq=1/256, ttl=64 (reply in 3)

› Frame 1: 1010 bytes on wire (8080 bits), 1010 bytes captured (8080 bits)
› Ethernet II, Src: PcsCompu_fc:6a:c9 (08:00:27:fc:6a:c9), Dst: PcsCompu_e2:9f:a6 (08:00:27:e2:9f:a6)
› Internet Protocol Version 4, Src: 2.1.1.2, Dst: 2.1.1.1
› Data (976 bytes)

0000  08 00 27 e2 9f a6 08 00  27 fc 6a c9 08 00 45 00   ··'·····'·j···E·
0010  03 e4 b5 d0 20 00 40 01  9b 44 02 01 01 02 02 01   ···· ·@··D······
0020  01 01 08 00 4d 71 13 c2  00 01 14 2b d2 59 00 00   ····Mq·····+·Y··
0030  00 00 3d 2a 00 00 00 00  00 00 10 11 12 13 14 15   ··=*···· ········
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ········ ·· !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37 38 39 3a 3b 3c 3d  3e 3f 40 41 42 43 44 45   6789:;<= >?@ABCDE
0070  46 47 48 49 4a 4b 4c 4d  4e 4f 50 51 52 53 54 55   FGHIJKLM NOPQRSTU
0080  56 57 58 59 5a 5b 5c 5d  5e 5f 60 61 62 63 64 65   VWXYZ[\] ^_`abcde
0090  66 67 68 69 6a 6b 6c 6d  6e 6f 70 71 72 73 74 75   fghijklm nopqrstu
00a0  76 77 78 79 7a 7b 7c 7d  7e 7f 80 81 82 83 84 85   vwxyz{|} ~·······
00b0  86 87 88 89 8a 8b 8c 8d  8e 8f 90 91 92 93 94 95   ········ ········
00c0  96 97 98 99 9a 9b 9c 9d  9e 9f a0 a1 a2 a3 a4 a5   ········ ········
00d0  a6 a7 a8 a9 aa ab ac ad  ae af b0 b1 b2 b3 b4 b5   ········ ········
00e0  b6 b7 b8 b9 ba bb bc bd  be bf c0 c1 c2 c3 c4 c5   ········ ········
00f0  c6 c7 c8 c9 ca cb cc cd  ce cf d0 d1 d2 d3 d4 d5   ········ ········

Frame (frame), 1,010 bytes          Packets: 3 · Displayed: 2 (66.7%)          Profile: Default
```

Fig 17 : This screenshot shows a Wireshark capture focused on IP fragmentation and ICMP traffic. The top pane lists two packets: the first is an IPv4 fragmented packet coming from source **2.1.1.2** to destination **2.1.1.1**, and the second is an ICMP Echo (ping) request from **2.1.1.2** to **2.1.1.1**. The fragmented IPv4 packet indicates that the original data was too large to fit into a single frame, requiring it to be split into multiple fragments for transmission. The details pane in the middle provides a deeper breakdown of the selected frame, showing Ethernet II header information, source and destination MAC addresses, IPv4 header fields, and protocol data. The bottom pane displays the raw hexadecimal and ASCII representation of the payload, highlighting the actual bytes transmitted. This view is useful for understanding how large packets are handled on the network and how ICMP communication works at a packet level, making it a helpful tool for studying fragmentation behavior and troubleshooting network issues.



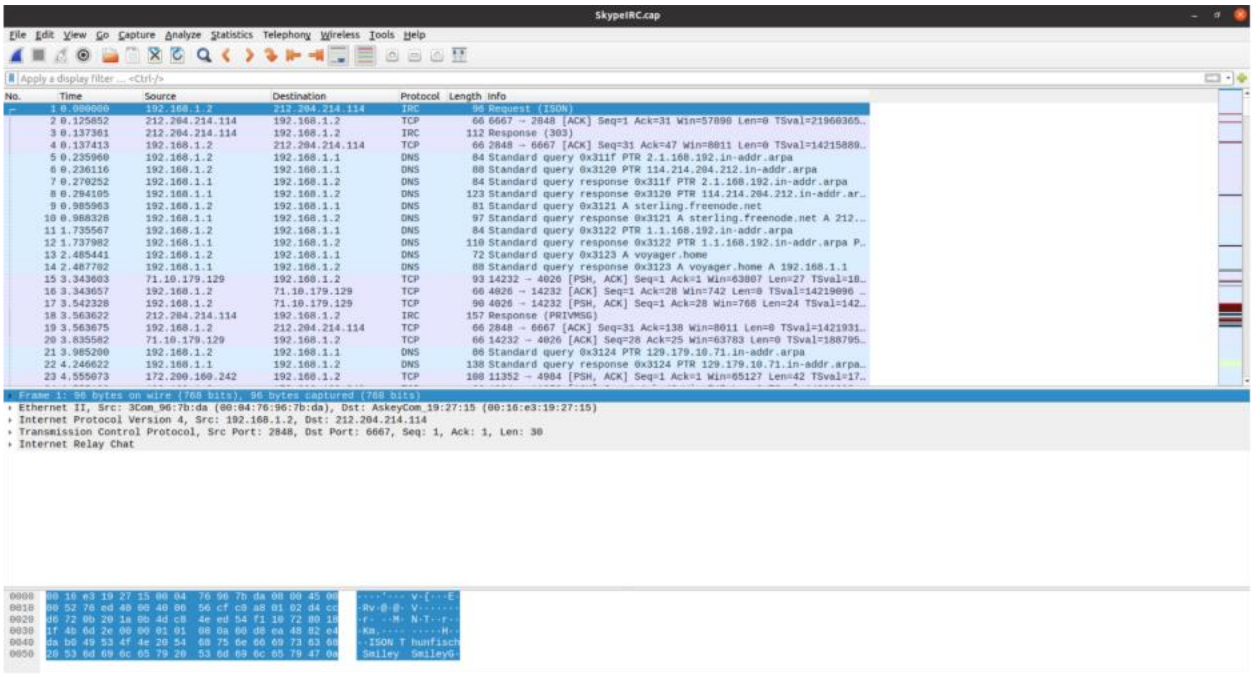
```
Wireshark · Packet 1 · ipv4frags.pcap

▾ Frame 1: 1010 bytes on wire (8080 bits), 1010 bytes captured (8080 bits)
    Encapsulation type: Ethernet (1)
    Arrival Time: Oct  2, 2017 08:03:32.535132000 EDT
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1506945812.535132000 seconds
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 1010 bytes (8080 bits)
    Capture Length: 1010 bytes (8080 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:data]
▸ Ethernet II, Src: PcsCompu_fc:6a:c9 (08:00:27:fc:6a:c9), Dst: PcsCompu_e2:9f:a6 (08:00:27:e2:9f:a6)
▸ Internet Protocol Version 4, Src: 2.1.1.2, Dst: 2.1.1.1
▸ Data (976 bytes)

0000  08 00 27 e2 9f a6 08 00  27 fc 6a c9 08 00 45 00   ··'·····'·j···E·
0010  03 e4 b5 d0 20 00 40 01  9b 44 02 01 01 02 02 01   ···· ·@··D······
0020  01 01 08 00 4d 71 13 c2  00 01 14 2b d2 59 00 00   ····Mq·····+·Y··
0030  00 00 3d 2a 00 00 00 00  00 00 10 11 12 13 14 15   ··=*············
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ·········· !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,-./012345
0060  36 37 38 39 3a 3b 3c 3d  3e 3f 40 41 42 43 44 45   6789:;<=>?@ABCDE
0070  46 47 48 49 4a 4b 4c 4d  4e 4f 50 51 52 53 54 55   FGHIJKLMNOPQRSTU
0080  56 57 58 59 5a 5b 5c 5d  5e 5f 60 61 62 63 64 65   VWXYZ[\]^_`abcde
0090  66 67 68 69 6a 6b 6c 6d  6e 6f 70 71 72 73 74 75   fghijklmnopqrstu
00a0  76 77 78 79 7a 7b 7c 7d  7e 7f 80 81 82 83 84 85   vwxyz{|}~·······
00b0  86 87 88 89 8a 8b 8c 8d  8e 8f 90 91 92 93 94 95   ················
00c0  96 97 98 99 9a 9b 9c 9d  9e 9f a0 a1 a2 a3 a4 a5   ················
00d0  a6 a7 a8 a9 aa ab ac ad  ae af b0 b1 b2 b3 b4 b5   ················
00e0  b6 b7 b8 b9 ba bb bc bd  be bf c0 c1 c2 c3 c4 c5   ················
00f0  c6 c7 c8 c9 ca cb cc cd  ce cf d0 d1 d2 d3 d4 d5   ················
0100  d6 d7 d8 d9 da db dc dd  de df e0 e1 e2 e3 e4 e5   ················
0110  e6 e7 e8 e9 ea eb ec ed  ee ef f0 f1 f2 f3 f4 f5   ················
0120  f6 f7 f8 f9 fa fb fc fd  fe ff 00 01 02 03 04 05   ················
0130  06 07 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ················
0140  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ·········· !"#$%
0150  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,-./012345
```

Fig 18 : This screenshot displays the detailed packet view in Wireshark, showing the full breakdown of a fragmented IPv4 packet captured during analysis. At the top of the window, Wireshark provides complete frame information, including the encapsulation type (Ethernet II), arrival time, epoch time, and frame length, confirming that 1010 bytes were captured on the wire. The packet was sent from the source MAC address **PcsCompu_f6:ca:c0** to the destination MAC **PcsCompu_92:9f:a6**, with the IPv4 header showing that the packet originated from **2.1.1.2** and was destined for **2.1.1.1**. The lower pane displays the raw data in both hexadecimal and ASCII formats, making it possible to inspect the payload at a byte level. This view is especially helpful when analyzing fragmented traffic, validating packet integrity, or examining the exact data transmitted across the network. By reviewing this detailed packet information, investigators can better understand how fragmentation occurs and identify any anomalies in packet structure or content.


ipv4frags.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
ip.flags.mf == 1 || ip.frag_offset > 0

No.  Time      Source   Destination  Protocol Length Info
   1 0.000000  2.1.1.2  2.1.1.1      IPv4     1010 Fragmented IP protocol (proto=ICMP 1, off=0, ID=b5d0) [Reasse…
   2 0.000065  2.1.1.2  2.1.1.1      ICMP      466 Echo (ping) request  id=0x13c2, seq=1/256, ttl=64 (reply in 3)

▾ Frame 1: 1010 bytes on wire (8080 bits), 1010 bytes captured (8080 bits)
    Encapsulation type: Ethernet (1)
    Arrival Time: Oct  2, 2017 08:03:32.535132000 EDT
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1506945812.535132000 seconds
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 1010 bytes (8080 bits)
    Capture Length: 1010 bytes (8080 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:data]
▾ Ethernet II, Src: PcsCompu_fc:6a:c9 (08:00:27:fc:6a:c9), Dst: PcsCompu_e2:9f:a6 (08:00:27:e2:9f:a6)
  ▸ Destination: PcsCompu_e2:9f:a6 (08:00:27:e2:9f:a6)
  ▸ Source: PcsCompu_fc:6a:c9 (08:00:27:fc:6a:c9)
    Type: IPv4 (0x0800)
▾ Internet Protocol Version 4, Src: 2.1.1.2, Dst: 2.1.1.1
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 996
    Identification: 0xb5d0 (46544)
  ▸ Flags: 0x2000, More fragments
    Fragment offset: 0
    Time to live: 64
    Protocol: ICMP (1)
    Header checksum: 0x9b44 [validation disabled]
    [Header checksum status: Unverified]
    Source: 2.1.1.2
    Destination: 2.1.1.1
    Reassembled IPv4 in frame: 2
▾ Data (976 bytes)
    Data: 08004d7113c20001142bd259000000003d2a080000000000…
    [Length: 976]

0000  08 00 27 e2 9f a6 08 00  27 fc 6a c9 08 00 45 00   ··'·····'·j···E·
0010  03 e4 b5 d0 20 00 40 01  9b 44 02 01 01 02 02 01   ···· ·@· ·D······
0020  01 01 08 00 4d 71 13 c2  00 01 14 2b d2 59 00 00   ····Mq·· ···+·Y··
0030  00 00 3d 2a 08 00 00 00  00 00 10 11 12 13 14 15   ··=*···· ········
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ········ ·· !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37 38 39 3a 3b 3c 3d  3e 3f 40 41 42 43 44 45   6789:;<= >?@ABCDE

Frame (frame), 1,010 bytes          Packets: 3 · Displayed: 2 (66.7%)          Profile: Default
```

Fig 19 : This screenshot shows Wireshark analyzing a fragmented IPv4 packet and an ICMP Echo (ping) request within the capture file. The upper panel lists two packets: the first is an IPv4 fragment sent from **2.1.1.2** to **2.1.1.1**, and the second is a standard ICMP Echo request, commonly used for testing connectivity. The detailed packet view in the middle pane breaks down the IPv4 fragment, showing key header information such as the total length (987 bytes), identification number, flags indicating fragmentation, fragment offset, TTL value, and the calculated header checksum. Wireshark also marks the checksum status as "unverified," indicating that validation was disabled for this capture. The bottom pane displays the raw fragment data in hexadecimal and ASCII formats, highlighting the payload reconstructed from multiple fragments. This detailed packet-level breakdown helps investigators understand how fragmentation was handled, verify packet integrity, and analyze the structure of ICMP traffic exchanged between the hosts.

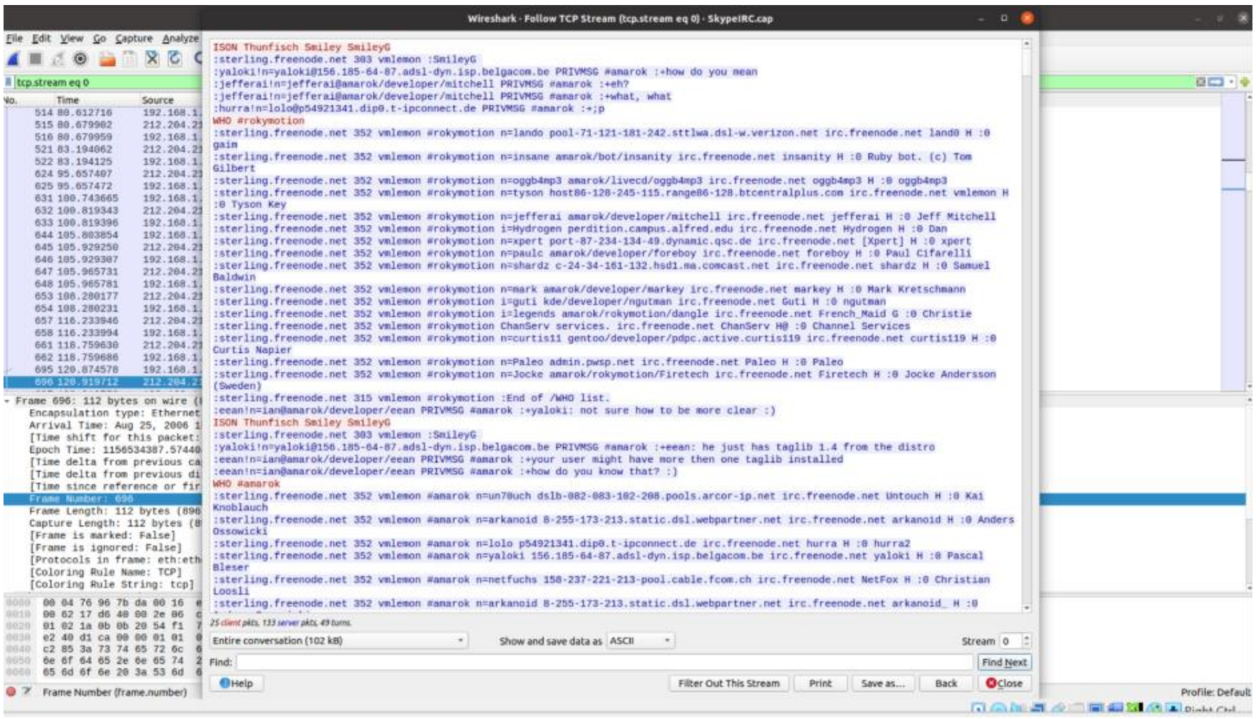

SkypeIRC.cap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>

```
No.   Time       Source           Destination      Protocol Length Info
  1 0.000000   192.168.1.2      212.204.214.114  IRC       96 Request (ISON)
  2 0.125852   212.204.214.114  192.168.1.2      TCP       66 6667 → 2848 [ACK] Seq=1 Ack=31 Win=57898 Len=0 TSval=21960365…
  3 0.137361   212.204.214.114  192.168.1.2      IRC      112 Response (303)
  4 0.137413   192.168.1.2      212.204.214.114  TCP       66 2848 → 6667 [ACK] Seq=31 Ack=47 Win=8011 Len=0 TSval=14215889…
  5 0.235960   192.168.1.2      192.168.1.1      DNS       84 Standard query 0x311f PTR 2.1.168.192.in-addr.arpa
  6 0.236116   192.168.1.2      192.168.1.1      DNS       88 Standard query 0x3120 PTR 114.214.204.212.in-addr.arpa
  7 0.270252   192.168.1.1      192.168.1.2      DNS       84 Standard query response 0x311f PTR 2.1.168.192.in-addr.arpa
  8 0.294105   192.168.1.1      192.168.1.2      DNS      123 Standard query response 0x3120 PTR 114.214.204.212.in-addr.ar…
  9 0.985963   192.168.1.2      192.168.1.1      DNS       81 Standard query 0x3121 A sterling.freenode.net
 10 0.988328   192.168.1.1      192.168.1.2      DNS       97 Standard query response 0x3121 A sterling.freenode.net A 212.…
 11 1.735567   192.168.1.2      192.168.1.1      DNS       84 Standard query 0x3122 PTR 1.1.168.192.in-addr.arpa
 12 1.737982   192.168.1.1      192.168.1.2      DNS      110 Standard query response 0x3122 PTR 1.1.168.192.in-addr.arpa P…
 13 2.485441   192.168.1.2      192.168.1.1      DNS       72 Standard query 0x3123 A voyager.home
 14 2.487702   192.168.1.1      192.168.1.2      DNS       88 Standard query response 0x3123 A voyager.home A 192.168.1.1
 15 3.343603   71.10.179.129    192.168.1.2      TCP       93 14232 → 4026 [PSH, ACK] Seq=1 Ack=1 Win=63807 Len=27 TSval=18…
 16 3.343657   192.168.1.2      71.10.179.129    TCP       66 4026 → 14232 [ACK] Seq=1 Ack=28 Win=742 Len=0 TSval=14219096 …
 17 3.542328   192.168.1.2      71.10.179.129    TCP       90 4026 → 14232 [PSH, ACK] Seq=1 Ack=28 Win=768 Len=24 TSval=142…
 18 3.563622   212.204.214.114  192.168.1.2      IRC      157 Response (PRIVMSG)
 19 3.563675   192.168.1.2      212.204.214.114  TCP       66 2848 → 6667 [ACK] Seq=31 Ack=138 Win=8011 Len=0 TSval=1421931…
 20 3.835582   71.10.179.129    192.168.1.2      TCP       66 14232 → 4026 [ACK] Seq=28 Ack=25 Win=63783 Len=0 TSval=188795…
 21 3.985200   192.168.1.2      192.168.1.1      DNS       86 Standard query 0x3124 PTR 129.179.10.71.in-addr.arpa
 22 4.246622   192.168.1.1      192.168.1.2      DNS      138 Standard query response 0x3124 PTR 129.179.10.71.in-addr.arpa…
 23 4.555073   172.200.160.242  192.168.1.2      TCP      108 11352 → 4984 [PSH, ACK] Seq=1 Ack=1 Win=65127 Len=42 TSval=17…
```

```
▸ Frame 1: 96 bytes on wire (768 bits), 96 bytes captured (768 bits)
▸ Ethernet II, Src: 3Com_96:7b:da (00:04:76:96:7b:da), Dst: AskeyCom_19:27:15 (00:16:e3:19:27:15)
▸ Internet Protocol Version 4, Src: 192.168.1.2, Dst: 212.204.214.114
▸ Transmission Control Protocol, Src Port: 2848, Dst Port: 6667, Seq: 1, Ack: 1, Len: 30
▸ Internet Relay Chat
```

```
0000  00 16 e3 19 27 15 00 04  76 96 7b da 08 00 45 00   ····'··· v·{···E·
0010  00 52 76 ed 40 00 40 06  56 cf c0 a8 01 02 d4 cc   ·Rv·@·@· V·······
0020  d6 72 0b 20 1a 0b 4d c8  4e ed 54 f1 10 72 80 18   ·r· ··M· N·T··r··
0030  1f 4b 6d 2e 00 00 01 01  08 0a 00 d8 ea 48 82 e4   ·Km.···· ·····H··
0040  da b0 49 53 4f 4e 20 54  68 75 6e 66 69 73 63 68   ··ISON T hunfisch
0050  20 53 6d 69 6c 65 79 20  53 6d 69 6c 65 79 47 0a    Smiley  SmileyG·
```

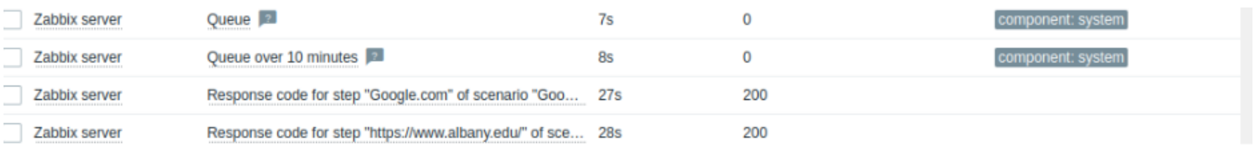
Fig 20 : This screenshot shows a Wireshark capture file named *SkypeIRC.cap*, displaying a mixture of DNS queries, TCP communication, and IRC (Internet Relay Chat) traffic. The top pane lists packets exchanged between several internal and external IP addresses, showing different activities such as DNS lookups, TCP handshakes, and application-layer data transfers. The highlighted packet is a TCP segment sent from **192.168.0.142** to **212.204.214.114**, which Wireshark identifies as part of an IRC session running over port 6667. The middle pane provides a structured breakdown of the selected packet, showing Ethernet II header details, IPv4 information, and TCP header values including the sequence and acknowledgment numbers. The bottom pane displays the raw data in hexadecimal and ASCII formats, revealing elements of IRC communication such as commands or user messages (e.g., “ISON huni,” “Smiley Smiley”). This capture demonstrates how Wireshark can decode real-time messaging protocols like IRC, providing visibility into server connections, chat behavior, and associated DNS activity, which can be especially useful for monitoring network conversations or investigating suspicious communication patterns.


Wireshark · Follow TCP Stream (tcp.stream eq 0) · SkypeIRC.cap
File Edit View Go Capture Analyze
tcp.stream eq 0
Help
Filter Out This Stream
Print
Save as...
Back
Close
Find Next
Profile: Default

```
No.  Time        Source
514 80.612716   192.168.1.
515 80.679902   212.204.2
516 80.679959   192.168.1.
521 83.194062   212.204.2
522 83.194125   192.168.1.
624 95.657407   212.204.2
625 95.657472   192.168.1.
631 100.743665  192.168.1.
632 100.819343  212.204.2
633 100.819396  192.168.1.
644 105.803854  192.168.1.
645 105.929250  212.204.2
646 105.929307  192.168.1.
647 105.965731  212.204.2
648 105.965781  192.168.1.
653 108.280177  212.204.2
654 108.280231  192.168.1.
657 116.233946  212.204.2
658 116.233994  192.168.1.
661 118.759630  212.204.2
662 118.759686  192.168.1.
695 120.874578  192.168.1.
696 120.919712  212.204.2

- Frame 696: 112 bytes on wire (
    Encapsulation type: Ethernet
    Arrival Time: Aug 25, 2006 1
    [Time shift for this packet:
    Epoch Time: 1156534387.57440
    [Time delta from previous ca
    [Time delta from previous di
    [Time since reference or fir
    Frame Number: 696
    Frame Length: 112 bytes (896
    Capture Length: 112 bytes (8
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:eth
    [Coloring Rule Name: TCP]
    [Coloring Rule String: tcp]

0000  00 04 76 96 7b da 00 16  e
0010  00 62 17 d6 40 00 2e 06  c
0020  01 02 1a 0b 0b 20 54 f1  7
0030  e2 40 d1 ca 00 00 01 01  0
0040  c2 85 3a 73 74 65 72 6c  6
0050  6e 6f 64 65 2e 6e 65 74  2
0060  65 6d 6f 6e 20 3a 53 6d  6

Frame Number (frame.number)
```

```
ISON Thunfisch Smiley SmileyG
:sterling.freenode.net 303 vmlemon :SmileyG
:yaloki!n=yaloki@156.185-64-87.adsl-dyn.isp.belgacom.be PRIVMSG #amarok :+how do you mean
:jefferai!n=jefferai@amarok/developer/mitchell PRIVMSG #amarok :+eh?
:jefferai!n=jefferai@amarok/developer/mitchell PRIVMSG #amarok :+what, what
:hurra!n=lolo@p54921341.dip0.t-ipconnect.de PRIVMSG #amarok :+;p
WHO #rokymotion
:sterling.freenode.net 352 vmlemon #rokymotion n=lando pool-71-121-181-242.sttlwa.dsl-w.verizon.net irc.freenode.net land0 H :0
gaim
:sterling.freenode.net 352 vmlemon #rokymotion n=insane amarok/bot/insanity irc.freenode.net insanity H :0 Ruby bot. (c) Tom
Gilbert
:sterling.freenode.net 352 vmlemon #rokymotion n=oggb4mp3 amarok/livecd/oggb4mp3 irc.freenode.net oggb4mp3 H :0 oggb4mp3
:sterling.freenode.net 352 vmlemon #rokymotion n=tyson host86-128-245-115.range86-128.btcentralplus.com irc.freenode.net vmlemon H
:0 Tyson Key
:sterling.freenode.net 352 vmlemon #rokymotion n=jefferai amarok/developer/mitchell irc.freenode.net jefferai H :0 Jeff Mitchell
:sterling.freenode.net 352 vmlemon #rokymotion i=Hydrogen perdition.campus.alfred.edu irc.freenode.net Hydrogen H :0 Dan
:sterling.freenode.net 352 vmlemon #rokymotion n=xpert port-87-234-134-49.dynamic.qsc.de irc.freenode.net [Xpert] H :0 xpert
:sterling.freenode.net 352 vmlemon #rokymotion n=paulc amarok/developer/foreboy irc.freenode.net foreboy H :0 Paul Cifarelli
:sterling.freenode.net 352 vmlemon #rokymotion n=shardz c-24-34-161-132.hsd1.ma.comcast.net irc.freenode.net shardz H :0 Samuel
Baldwin
:sterling.freenode.net 352 vmlemon #rokymotion n=mark amarok/developer/markey irc.freenode.net markey H :0 Mark Kretschmann
:sterling.freenode.net 352 vmlemon #rokymotion i=guti kde/developer/ngutman irc.freenode.net Guti H :0 ngutman
:sterling.freenode.net 352 vmlemon #rokymotion i=legends amarok/rokymotion/dangle irc.freenode.net French_Maid G :0 Christie
:sterling.freenode.net 352 vmlemon #rokymotion ChanServ services. irc.freenode.net ChanServ H@ :0 Channel Services
:sterling.freenode.net 352 vmlemon #rokymotion n=curtis11 gentoo/developer/pdpc.active.curtis119 irc.freenode.net curtis119 H :0
Curtis Napier
:sterling.freenode.net 352 vmlemon #rokymotion n=Paleo admin.pwsp.net irc.freenode.net Paleo H :0 Paleo
:sterling.freenode.net 352 vmlemon #rokymotion n=Jocke amarok/rokymotion/Firetech irc.freenode.net Firetech H :0 Jocke Andersson
(Sweden)
:sterling.freenode.net 315 vmlemon #rokymotion :End of /WHO list.
:eean!n=ian@amarok/developer/eean PRIVMSG #amarok :+yaloki: not sure how to be more clear :)
ISON Thunfisch Smiley SmileyG
:sterling.freenode.net 303 vmlemon :SmileyG
:yaloki!n=yaloki@156.185-64-87.adsl-dyn.isp.belgacom.be PRIVMSG #amarok :+eean: he just has taglib 1.4 from the distro
:eean!n=ian@amarok/developer/eean PRIVMSG #amarok :+your user might have more then one taglib installed
:eean!n=ian@amarok/developer/eean PRIVMSG #amarok :+how do you know that? :)
WHO #amarok
:sterling.freenode.net 352 vmlemon #amarok n=un70uch dslb-082-083-102-208.pools.arcor-ip.net irc.freenode.net Untouch H :0 Kai
Knoblauch
:sterling.freenode.net 352 vmlemon #amarok n=arkanoid 8-255-173-213.static.dsl.webpartner.net irc.freenode.net arkanoid H :0 Anders
Ossowicki
:sterling.freenode.net 352 vmlemon #amarok n=lolo p54921341.dip0.t-ipconnect.de irc.freenode.net hurra H :0 hurra2
:sterling.freenode.net 352 vmlemon #amarok n=yaloki 156.185-64-87.adsl-dyn.isp.belgacom.be irc.freenode.net yaloki H :0 Pascal
Bleser
:sterling.freenode.net 352 vmlemon #amarok n=netfuchs 158-237-221-213-pool.cable.fcom.ch irc.freenode.net NetFox H :0 Christian
Loosli
:sterling.freenode.net 352 vmlemon #amarok n=arkanoid 8-255-173-213.static.dsl.webpartner.net irc.freenode.net arkanoid_ H :0

25 client pkts, 133 server pkts, 49 turns.
Entire conversation (102 kB)    Show and save data as ASCII    Stream 0
Find:
```

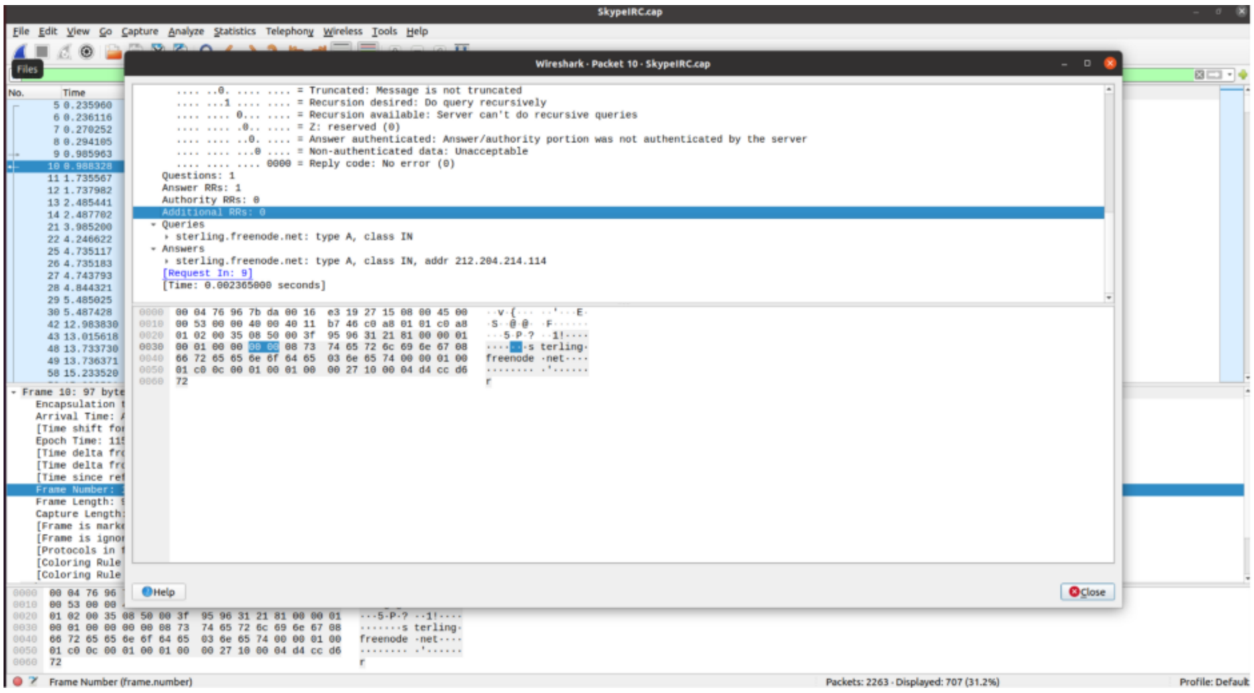
Fig 21 : This screenshot shows the "Follow TCP Stream" window in Wireshark, which reconstructs an entire IRC (Internet Relay Chat) conversation from the captured network traffic. The left panel lists the packets involved in the TCP stream, while the right panel displays the full text of the chat messages exchanged between users and the IRC server. The conversation includes a variety of IRC commands and responses, such as user status checks (e.g., *ISON*), nickname information, channel messages, and system-generated updates from servers like *freenode.net*. The stream reveals interactions from multiple users, including chat messages, join/leave notifications, and server replies that list channel participants and their roles. Wireshark stitches all these packets together into a readable transcript, allowing investigators to observe the complete communication flow without manually piecing together packets. This feature is especially valuable when analyzing messaging protocols, monitoring real-time communication, or investigating suspicious chat activity over TCP.



☐	Zabbix server	Queue	7s	0	component: system
☐	Zabbix server	Queue over 10 minutes	8s	0	component: system
☐	Zabbix server	Response code for step "Google.com" of scenario "Goo...	27s	200	
☐	Zabbix server	Response code for step "https://www.albany.edu/" of sce...	28s	200	

Fig 22 : This screenshot displays monitoring data from a Zabbix server, showing the status of several system checks. The first two entries relate to the server's internal queue metrics, indicating the time spent in the general queue (7 seconds) and the queue over 10 minutes (8 seconds), both of which currently report zero issues. The other entries show HTTP response code checks for web scenarios that test the availability of external websites such as *Google.com* and *albany.edu*. Both sites returned a successful HTTP status code **200**, confirming that they were reachable at the time of monitoring. Each item is categorized under the "system" component, indicating

that these checks are part of the server's routine health and performance diagnostics. This view helps administrators quickly verify service availability and detect any delays or failures in real-time.


SkypeIRC.cap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Files
Wireshark · Packet 10 · SkypeIRC.cap
Help
Close
Frame Number (frame.number)
Packets: 2263 · Displayed: 707 (31.2%)
Profile: Default

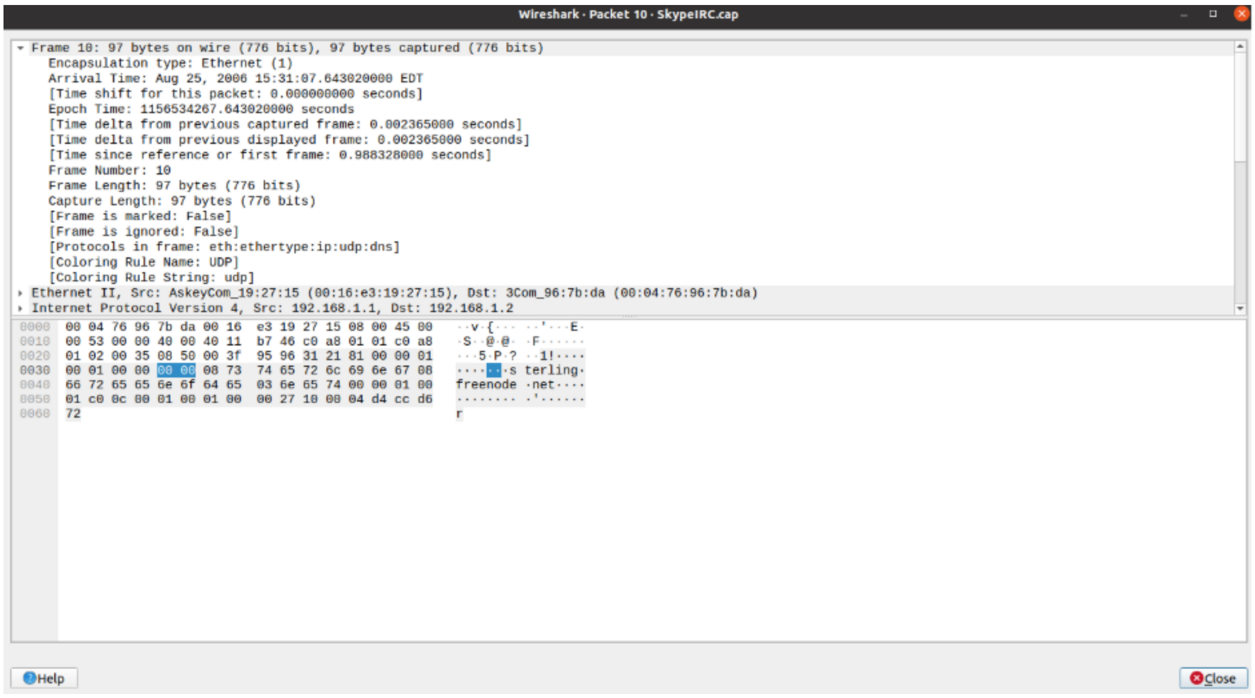
```
No.  Time
 5 0.235960
 6 0.236116
 7 0.270252
 8 0.294105
 9 0.985963
10 0.988328
11 1.735567
12 1.737982
13 2.485441
14 2.487702
21 3.985200
22 4.246622
25 4.735117
26 4.735183
27 4.743793
28 4.844321
29 5.485025
30 5.487428
42 12.983830
43 13.015618
48 13.733730
49 13.736371
58 15.233520
```

```
    .... ..0. .... .... = Truncated: Message is not truncated
    .... ...1 .... .... = Recursion desired: Do query recursively
    .... .... 0... .... = Recursion available: Server can't do recursive queries
    .... .... .0.. .... = Z: reserved (0)
    .... .... ..0. .... = Answer authenticated: Answer/authority portion was not authenticated by the server
    .... .... ...0 .... = Non-authenticated data: Unacceptable
    .... .... .... 0000 = Reply code: No error (0)
  Questions: 1
  Answer RRs: 1
  Authority RRs: 0
  Additional RRs: 0
▾ Queries
  ▸ sterling.freenode.net: type A, class IN
▾ Answers
  ▸ sterling.freenode.net: type A, class IN, addr 212.204.214.114
  [Request In: 9]
  [Time: 0.002365000 seconds]
```

```
0000  00 04 76 96 7b da 00 16  e3 19 27 15 08 00 45 00   ··v·{···  ··'···E·
0010  00 53 00 00 40 00 40 11  b7 46 c0 a8 01 01 c0 a8   ·S··@·@·  ·F······
0020  01 02 00 35 08 50 00 3f  95 96 31 21 81 00 00 01   ···5·P·?  ··1!····
0030  00 01 00 00 00 00 08 73  74 65 72 6c 69 6e 67 08   ·······s terling·
0040  66 72 65 65 6e 6f 64 65  03 6e 65 74 00 00 01 00   freenode ·net····
0050  01 c0 0c 00 01 00 01 00  00 27 10 00 04 d4 cc d6   ········ ·'······
0060  72                                                 r
```

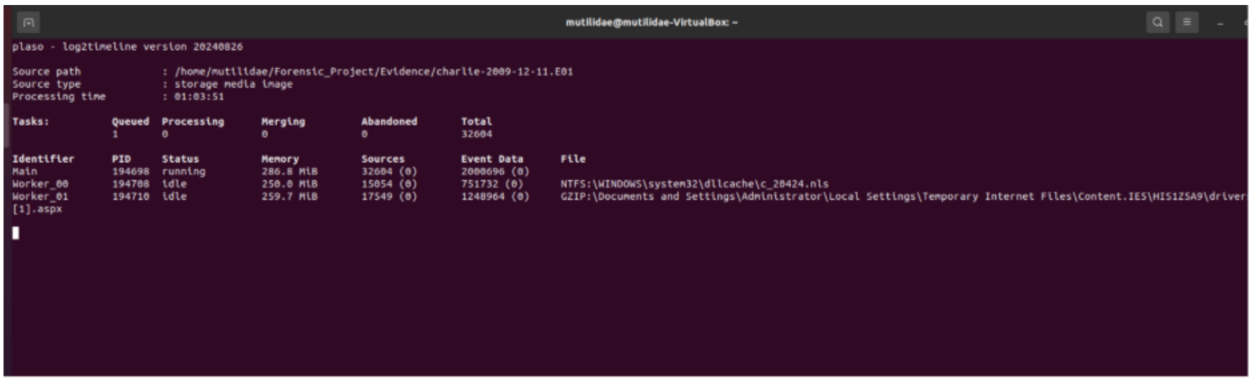
```
▾ Frame 10: 97 byte
    Encapsulation t
    Arrival Time: A
    [Time shift for
    Epoch Time: 115
    [Time delta fro
    [Time delta fro
    [Time since ref
    Frame Number: 1
    Frame Length: 9
    Capture Length:
    [Frame is marke
    [Frame is ignor
    [Protocols in f
    [Coloring Rule
    [Coloring Rule
```

Fig 23 : This screenshot shows a detailed DNS response packet being analyzed in Wireshark from the *SkypeIRC.cap* capture file. The top section of the window displays the decoded DNS information, indicating that the query was not truncated and recursion was available on the server side. The packet contains one question and one answer, where the domain **sterling.freenode.net** was queried for its A record. The answer section shows that the DNS server successfully resolved the hostname to the IP address **212.204.214.114**. Wireshark also reports a very low response time of approximately **0.0026 seconds**, confirming fast DNS resolution. In the lower pane, the raw DNS message is shown in hexadecimal and ASCII formats, allowing deeper inspection of the underlying data. This view provides insight into how client devices resolve IRC server names during chat sessions and demonstrates how DNS responses can be analyzed for troubleshooting or monitoring network activity.



```
Wireshark · Packet 10 · SkypeIRC.cap

▾ Frame 10: 97 bytes on wire (776 bits), 97 bytes captured (776 bits)
    Encapsulation type: Ethernet (1)
    Arrival Time: Aug 25, 2006 15:31:07.643020000 EDT
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1156534267.643020000 seconds
    [Time delta from previous captured frame: 0.002365000 seconds]
    [Time delta from previous displayed frame: 0.002365000 seconds]
    [Time since reference or first frame: 0.988328000 seconds]
    Frame Number: 10
    Frame Length: 97 bytes (776 bits)
    Capture Length: 97 bytes (776 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:udp:dns]
    [Coloring Rule Name: UDP]
    [Coloring Rule String: udp]
▸ Ethernet II, Src: AskeyCom_19:27:15 (00:16:e3:19:27:15), Dst: 3Com_96:7b:da (00:04:76:96:7b:da)
▸ Internet Protocol Version 4, Src: 192.168.1.1, Dst: 192.168.1.2

0000  00 04 76 96 7b da 00 16  e3 19 27 15 08 00 45 00   ··v·{··· ··'···E·
0010  00 53 00 00 40 00 40 11  b7 46 c0 a8 01 01 c0 a8   ·S··@·@· ·F······
0020  01 02 00 35 08 50 00 3f  95 96 31 21 81 00 00 01   ···5·P·? ··1!····
0030  00 01 00 00 00 00 08 73  74 65 72 6c 69 6e 67 08   ·······s terling·
0040  66 72 65 65 6e 6f 64 65  03 6e 65 74 00 00 01 00   freenode ·net····
0050  01 c0 0c 00 01 00 01 00  00 27 10 00 04 d4 cc d6   ········ ·'······
0060  72                                                 r
```

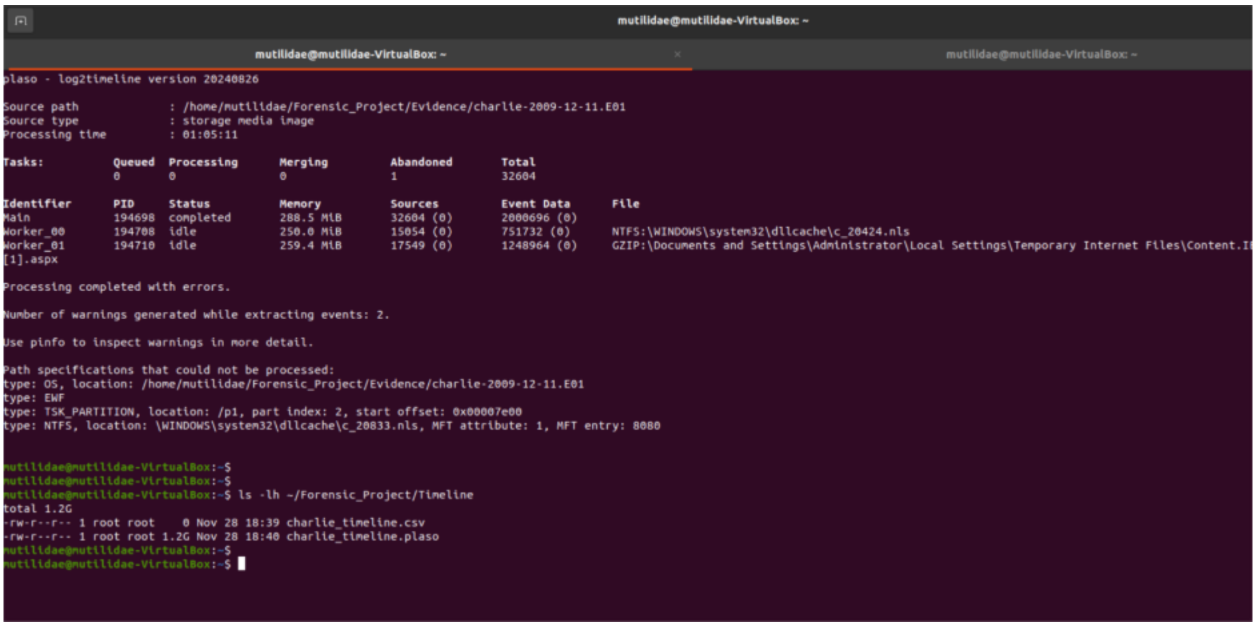
Fig 24 : This screenshot presents the detailed packet view of a DNS response captured in Wireshark from the *SkypeIRC.cap* traffic. The top section outlines key frame information, showing that the packet is 97 bytes in size and was encapsulated using Ethernet. Wireshark records the exact arrival time of the packet along with timestamps such as the epoch time and deltas between frames, which help in analyzing packet timing and sequence. The packet contains UDP-based DNS traffic, as indicated by the protocol breakdown. In the middle pane, the decoded protocol information reveals that the DNS message was exchanged between the internal host **192.168.1.1** and the external server **212.204.214.114**, consistent with a hostname lookup for the *freenode.net* IRC network. The lower pane displays the raw packet contents in hexadecimal and ASCII formats, allowing investigators to inspect the precise DNS query and response data. This detailed view helps in validating DNS behavior, troubleshooting name-resolution issues, and documenting how client applications, such as IRC clients, communicate with remote servers.



```
mutillidae@mutillidae-VirtualBox: ~

plaso - log2timeline version 20240826

Source path           : /home/mutillidae/Forensic_Project/Evidence/charlie-2009-12-11.E01
Source type           : storage media image
Processing time       : 01:03:51

Tasks:        Queued  Processing    Merging       Abandoned     Total
              1       0             0             0             32604

Identifier    PID     Status        Memory        Sources       Event Data    File
Main          194698  running       286.8 MiB     32604 (0)     2000696 (0)
Worker_00     194708  idle          250.0 MiB     15054 (0)     751732 (0)    NTFS:\WINDOWS\system32\dllcache\c_20424.nls
Worker_01     194710  idle          259.7 MiB     17549 (0)     1248964 (0)   GZIP:\Documents and Settings\Administrator\Local Settings\Temporary Internet Files\Content.IE5\HIS1Z5A9\driver
[1].aspx

█
```

Fig 25 : This screenshot shows the log2timeline (plaso) tool running inside the Ubuntu

VirtualBox environment, processing a forensic disk image named *charlie-2009-12-11.E01*. The tool extracts timestamped events from the storage media image and assembles them into a unified forensic timeline. The display shows the overall processing time (1 hour, 3 minutes, and 51 seconds) and indicates that one task is currently queued. The "Main" process is actively running, while two worker processes are idle, each assigned a process ID and displaying their respective memory usage. The output also lists example events being processed, including data extracted from system DLL cache files and temporary internet files located in the Administrator profile. The event counts, ranging from thousands to hundreds of thousands, demonstrate the large volume of artifacts recovered from the image. This view confirms that the timeline generation is progressing successfully and that plaso is actively parsing file system artifacts to build a comprehensive chronological record for forensic analysis.



mutilidae@mutilidae-VirtualBox: ~
mutilidae@mutilidae-VirtualBox: ~
mutilidae@mutilidae-VirtualBox: ~

```
plaso - log2timeline version 20240826

Source path          : /home/mutilidae/Forensic_Project/Evidence/charlie-2009-12-11.E01
Source type          : storage media image
Processing time      : 01:05:11

Tasks:          Queued  Processing      Merging         Abandoned       Total
                0       0               0               1               32604

Identifier      PID     Status          Memory          Sources         Event Data      File
Main            194698  completed       288.5 MiB       32604 (0)       2000696 (0)
Worker_00       194708  idle            250.0 MiB       15054 (0)       751732 (0)      NTFS:\WINDOWS\system32\dllcache\c_20424.nls
Worker_01       194710  idle            259.4 MiB       17549 (0)       1248964 (0)     GZIP:\Documents and Settings\Administrator\Local Settings\Temporary Internet Files\Content.I
[1].aspx

Processing completed with errors.

Number of warnings generated while extracting events: 2.

Use pinfo to inspect warnings in more detail.

Path specifications that could not be processed:
type: OS, location: /home/mutilidae/Forensic_Project/Evidence/charlie-2009-12-11.E01
type: EWF
type: TSK_PARTITION, location: /p1, part index: 2, start offset: 0x00007e00
type: NTFS, location: \WINDOWS\system32\dllcache\c_20833.nls, MFT attribute: 1, MFT entry: 8080


mutilidae@mutilidae-VirtualBox:~$
mutilidae@mutilidae-VirtualBox:~$
mutilidae@mutilidae-VirtualBox:~$ ls -lh ~/Forensic_Project/Timeline
total 1.2G
-rw-r--r-- 1 root root    0 Nov 28 18:39 charlie_timeline.csv
-rw-r--r-- 1 root root 1.2G Nov 28 18:40 charlie_timeline.plaso
mutilidae@mutilidae-VirtualBox:~$
mutilidae@mutilidae-VirtualBox:~$ █
```

Fig 26 : This screenshot shows the completion of the log2timeline (plaso) processing on the forensic disk image *charlie-2009-12-11.E01*. The terminal output indicates that all tasks have finished, with the main process and worker processes completing their assigned event extraction operations. Although a few warnings were generated during processing, the extraction was successfully completed and plaso produced a large number of events from NTFS file system artifacts, such as DLL cache entries and temporary internet files. Additional details show the path specification for one of the parsed items, including partition offset and MFT record information. After processing, the investigator lists the contents of the output directory, confirming that two timeline files were successfully generated: *charlie_timeline.csv* and *charlie_timeline.plaso*. These files contain the consolidated timeline data, which can now be further analyzed to reconstruct user activity and system events from the forensic image.

MEMORY ANALYSIS SCREENSHOTS:

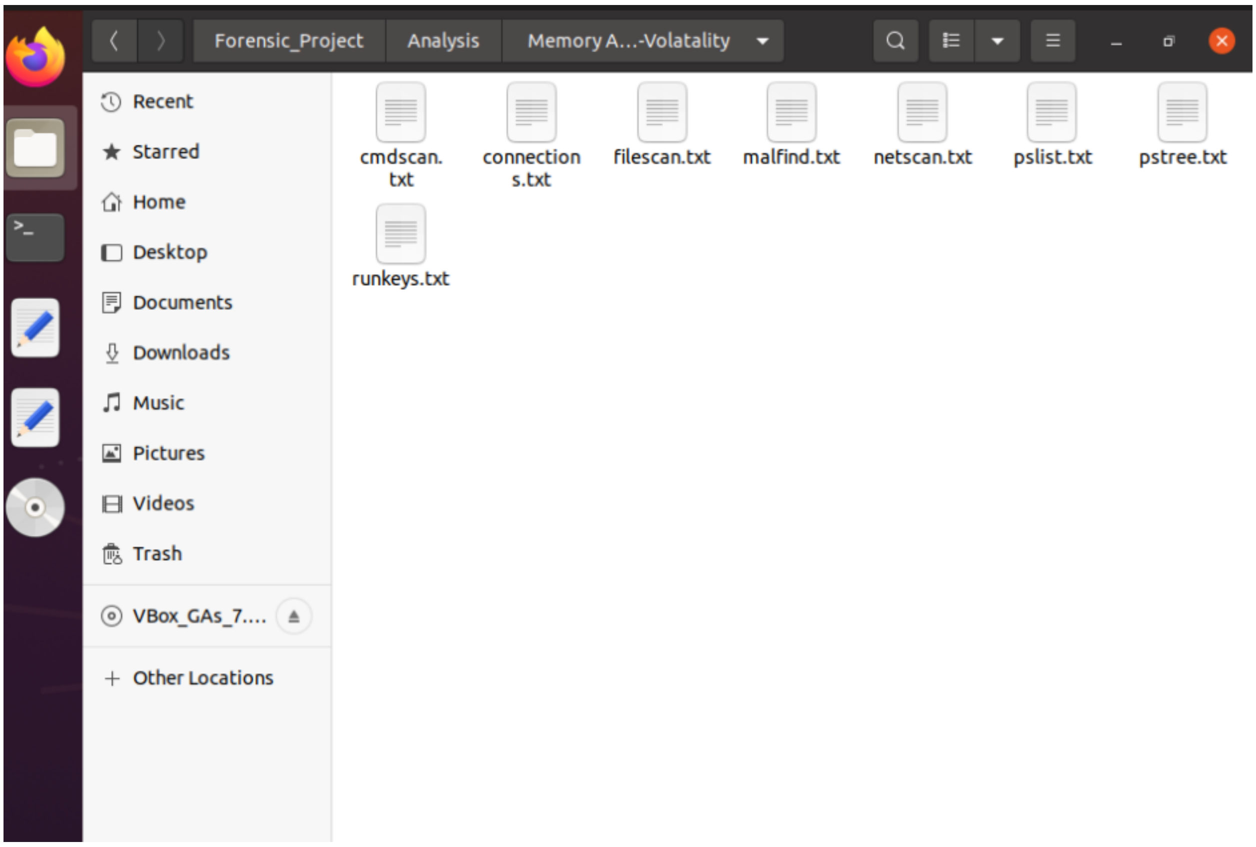


Fig 27 : This screenshot shows the set of output files generated during the **memory analysis stage** of the project using the Volatility framework. Each text file represents the results of a specific Volatility plugin, providing valuable insights into the system's live state at the moment the memory dump was captured. For example, *cmdscan.txt* contains fragments of command-line activity recovered from memory, which is helpful for identifying commands executed by the attacker. The *connections.txt* and *netscan.txt* files list active and recently closed network connections, allowing us to determine whether the compromised machine was communicating with suspicious hosts or command-and-control servers. The *filescan.txt* output identifies file objects that were loaded in memory, including those that may no longer exist on disk—useful for detecting malicious or temporary files. The *malfind.txt* file is particularly important, as it highlights suspicious memory regions, injected code, or hidden processes indicative of malware. Meanwhile, *pslist.txt* and *pstree.txt* document all running and parent-child process relationships, helping us spot abnormal or disguised processes. Finally, the *runkeys.txt* file reveals registry run keys loaded into memory, which can point to persistence mechanisms used by an attacker. Collectively, these outputs form the foundation for reconstructing attacker activity, identifying abnormal behavior, and correlating findings with disk and network evidence, making them a crucial component of the forensic investigation.

Conclusion:

In this project, our team worked together to investigate a compromised system using disk, memory, and network forensics. By examining the disk image, memory dump, and packet captures, we were able to identify deleted files, suspicious processes, injected code, and communication with external servers. This helped us understand how the attacker gained access, maintained control, and attempted to hide their activity. Using tools like Autopsy, Volatility, Wireshark, and Plaso also gave us hands-on experience with the same methods used in real cybersecurity investigations. As a team, we compared our findings and connected the evidence to build a clear timeline of the attack. Overall, the project strengthened our practical skills in digital forensics, improved our ability to analyze different types of evidence, and showed us how important it is to work collaboratively when solving complex cybersecurity incidents.

Recommendations:

Based on the findings of the disk, memory, and network analysis conducted during this investigation, several improvements can be recommended to help prevent similar compromises in the future. These recommendations focus on strengthening system security, improving monitoring, and ensuring faster detection and response in real-world environments.

1. Strengthen System Hardening and Patch Management

The compromised Windows XP machine showed signs of outdated software, missing patches, and legacy configurations. It is strongly recommended to:

- Decommission unsupported operating systems such as Windows XP.
- Apply security patches regularly using automated patch management.
- Disable unused services and restrict administrative privileges.
- Ensure that antivirus and endpoint security tools are regularly updated.

Keeping systems updated reduces the attack surface and prevents exploitation through known vulnerabilities.

2. Implement Endpoint Detection and Response (EDR) Tools

Memory analysis revealed malicious processes, injected code, and suspicious network sessions that standard antivirus tools likely failed to detect. Deploying modern EDR solutions would:

- Monitor processes in real time
- Detect code injection or anomalous memory behaviour
- Alert administrators to unusual file or process activity
- Provide behavioural analytics rather than relying on signatures

EDR tools would significantly improve early detection of stealthy attacks.

3. Improve Network Monitoring and Logging

Network captures showed IRC communications, HTTP traffic, and fragmented packets that went unnoticed. To prevent this:

- Enable deep packet inspection (DPI) at firewalls and IDS/IPS devices.
- Block legacy/unsecured protocols like IRC unless explicitly required.
- Log DNS, HTTP, and unusual outbound traffic for anomaly detection.
- Configure alerts for connections to unknown IP addresses or high-risk ports (e.g., 6667 for IRC).

Better network visibility ensures that any suspicious communication is identified early.

4. Enforce Strong Access Controls and Credential Protection

Disk artifacts revealed altered registry entries, persistence mechanisms, and evidence suggesting unauthorized access. To improve this:

- Implement strong password policies and MFA (multi-factor authentication).
- Limit administrative access and enforce the principle of least privilege.
- Regularly audit user accounts, login times, and authentication logs.

- Use credential protection features (e.g., LSASS protection on modern systems).

Stricter access control minimizes the risk of attackers maintaining long-term footholds.

5. Enable Robust Logging and Centralized Log Storage

A major challenge in the investigation was the lack of structured logs on the compromised system. To prevent such gaps:

- Enable detailed system and security event logging.
- Store logs centrally in a SIEM system (Splunk, ELK, QRadar, etc.).
- Retain logs for an adequate duration to support forensic analysis.
- Regularly review logs for anomalies using automated alerts.

Centralised logs significantly strengthen incident response capabilities.

6. Deploy Host-Based Firewalls and Application Whitelisting

The malware and suspicious executables found on disk suggest that no application control policies were in place. To prevent arbitrary code execution:

- Enforce application whitelisting policies to allow only approved software.
- Configure host firewalls to block outbound traffic to unusual ports.
- Restrict access to system directories like System32 and Startup folders.

Whitelisting ensures that unknown or malicious programs cannot run freely.