Honeypot-based monitoring, detection, response, and information protection framework*

Maxim Prodeus [1,†], Andrii Nicheporuk [1,*,†], Antonina Kashtalian[1,†], Mariia Kapustian[1,†], Tomas Sochor[2,†]

[1] *Khmelnytskyi National University, Institutska str., 11, Khmelnytskyi, 29016, Ukraine*
[2] *European Research University, Ostrava, Czech Republic*

Abstract

Threats to computer networks continue to grow, but existing security solutions consistently fail to address these challenges. In this paper, we introduce a paradigm for protecting computational resources—honeypot technology. This technique involves populating a system with data that appears authentic but is, in fact, counterfeit. Attacks can be detected by monitoring this deceptive information for access events.

Honeypots are capable of identifying malicious activities, such as insider and masquerade attacks, which go beyond traditional security measures. They can be used to mitigate privacy breaches either in advance or after they have occurred. This work explores the challenges that must be overcome to successfully deploy honeypots as part of a comprehensive security solution. It discusses scenarios in which honeypots are particularly useful, as well as the characteristics that an effective honeypot system should possess.

Additionally, we describe the tools we have developed for efficiently creating and distributing honeypots to form a network of sensors capable of detecting adversarial activities occurring anywhere within an organization's computer system.

Keywords

Honeypot, network security, threat detection, cyber defense

1. Introduction

A cyber honeypot is a specially designed system intended to mimic real servers or user devices to attract hacker attacks. It contains fake data or services that appear enticing to malicious actors but hold no actual value for the organization. The primary function of a honeypot is to log unauthorized access attempts and analyze the actions of attackers [1].

The use of honeypots remains a topic of debate in cybersecurity. They appeal to specialists due to their ease of deployment and ability to detect network intrusions in a timely manner. However, the issue arises when organizations perceive honeypots as a primary threat detection tool rather than integrating them as part of a comprehensive security strategy.

A specific type of honeypot is a honeypot file. A honeypot file is a document specifically created to detect unauthorized access and monitor the behavior of potential attackers. By placing such a file within a system, organizations can receive alerts on access attempts, aiding in the identification of security threats [2].

Honeypot files function as digital traps that remain unnoticed by regular users but attract the attention of cybercriminals. Each access attempt is logged, allowing for the tracking of suspicious activity and the analysis of potential threats. This capability not only facilitates the early detection

Intelitsis'25: The 6th International Workshop on Intelligent Information Technologies & Systems of Information Security, April 04, 2025, Khmelnytskyi, Ukraine
* Corresponding author.
† These authors contributed equally.
mprodeus99@ukr.net (M. Prodeus); andrey.nicheporuk@gmail.com (A. Nicheporuk); tomas.sochor@eruni.org (T. Sochor); yantonina@ukr.net (A. Kashtalian); kapustianm@khmnu.edu.ua (M. Kapustian)
0009-0002-2968-4648 (M. Prodeus); 0000-0002-7230-9475 (A. Nicheporuk); 0000-0002-1704-1883 (T. Sochor); 0000-0002-4925-9713 (A. Kashtalian); 0000-0001-9200-1622 (M. Kapustian)

of attacks but also provides insights into attacker behavior, which helps improve cybersecurity mechanisms [3].

The effectiveness of honeypots depends not only on proper configuration but also on their strategic placement within the network. Minimizing false positives and ensuring the system appears realistic is crucial for maintaining its attractiveness to cybercriminals. At the same time, honeypots should only be one component of an overarching cybersecurity strategy. To effectively counteract attacks, particularly ransomware, it is essential to clearly understand the capabilities and limitations of this tool [4].

2. Related works

Today, the problem of creating honeypots for analyzing malicious activity is receiving significant attention in the literature.

In study [5], the authors present HoneyHive — a distributed threat detection system based on IoT honeypot sensors. The research describes an approach to integrating honeypot systems into the Internet of Things (IoT) environment for effective monitoring and analysis of potential attacks. The core idea of the framework is that small devices with embedded honeypot modules are deployed in various network segments, simulating vulnerable nodes that attackers might target.

One of the key advantages of HoneyHive is its distributed architecture, which enables real-time network security monitoring without requiring administrator intervention. The use of lightweight honeypot modules combined with centralized threat analysis makes this system an efficient solution for securing IoT infrastructure. Additionally, special attention was given to the issue of metamorphic virus propagation; however, actively hunting for such threats remains a significant challenge.

In [6], the authors develop a conceptual model of multi-computer systems designed to facilitate the operation of antivirus honeypots and traps for detecting malware and cyberattacks within corporate networks. The proposed approach is aimed at preventing and counteracting the infiltration of metamorphic viruses.

The developed model incorporates combined honeypots and traps, as well as a decision-making controller to detect and mitigate malware and cyber threats. Moreover, such systems may include modified operating environments that execute programs in a deceptive manner for research purposes. The proposed method enhances the efficiency of detecting and preventing the spread of metamorphic viruses in corporate networks.

Study [7] introduces a new approach to detecting metamorphic viruses. The method consists of two stages: the first involves searching for equivalent functional blocks based on a statistical assessment of instruction occurrences, while the second refines the selection of equivalent blocks and determines levels of obfuscation.

The research results indicate that the proposed method effectively detects metamorphic viruses, even when complex obfuscation techniques are used. This is achieved through analyzing the correspondences between functional blocks in different versions of the virus, allowing the identification of malware despite its modifications. As a result, the methodology contributes to enhancing the reliability of cybersecurity systems in countering modern threats.

3. Creating a honeypot file using the Canarytokens tool

To generate a honeypot file, it must be marked in some way. This can be done using a token. In this study, the Canarytokens service was used [8].

After creating the honeypot file, it is recommended to add information that may attract attackers. For example, fake accounts with enticing names such as "backupadmin" or "passwords" can be created without granting them real access rights. This may encourage an attacker to attempt using these credentials, allowing their actions to be tracked.

To effectively monitor access to the honeypot file, it was placed on a dedicated SharePoint site with special settings, including disabling OneDrive synchronization to prevent offline access.

The uploaded honeypot file on the created site will attract attention but will not reveal itself as a trap.

To monitor access to the honeypot file, Microsoft 365 Compliance Center policies or Microsoft Cloud App Security (MCAS) can be used.

The primary triggers for initiating protective actions include file opening, content modification, or file relocation. A single event is sufficient to raise an alert, ensuring a fast response time to potential threats. However, this also increases the number of false positives.

4. Information monitoring, detection, response and protection framework

The developed framework is designed to detect malware intrusions. Specifically, the subsystem is capable of identifying viruses and trojans that attempt to access files containing confidential information. A virus that copies or modifies documents will be detected through file modification time monitoring. Trojan malware scanning the system for critical files may trigger the honeypot document, leading to its identification. Spyware programs that transfer files to remote servers can be detected through repeated attempts to access the honeypot [9].

Worms that spread automatically across networks may infect the honeypot file, allowing their activity to be tracked. Frequent file modifications can indicate the presence of malicious code propagating through the network. It is also important to consider that as network speeds increase, the number of malicious behaviors and malware files increases. Therefore, pattern-matching algorithms need to be faster. It is possible to use new experimental pattern matching methods, which will allow for faster response times to threats [10]. Automatic blocking of the infected device via the Windows firewall can help contain the spread of the threat [11].

A honeypot file can also be used to detect ransomware (e.g., WannaCry, Ryuk, LockBit) [12]. If malware attempts to encrypt the honeypot file, the system will log changes to its structure. Frequent modification attempts within a short period may indicate a ransomware attack [13]. Automatic blocking of suspicious processes or network activity can help prevent the further spread of ransomware.

Attackers who gain initial access to the system often scan file servers for valuable data. Honeypot files placed in various locations (local drives, cloud services, shared folders) can help identify anomalous user behavior. If an unknown device attempts to access the file, the system can notify the administrator and block the IP address.

If a honeypot file is used in corporate shared environments (such as SharePoint or OneDrive), it can help detect social engineering attacks, where an attacker tries to gain access to internal documents [14]. Alerts on honeypot file access from an unknown IP address may indicate account compromise. The execution of macros within the file can reveal malicious documents attempting to download additional malware [15].

A particularly challenging type of malware is metamorphic viruses. A metamorphic virus may attempt to modify the honeypot file to insert its code or activate malicious logic. The system detects such changes through the file modification trigger and logs the intrusion. If the virus alters the honeypot file structure (e.g., injecting new malicious code), this will be recorded [16]. If file metadata (such as modification time or access permissions) is altered, the honeypot system will also detect this. A metamorphic virus may attempt to copy the honeypot file to temporary directories or change its permissions. In such cases, the file copy or permission change trigger will activate, allowing the detection of unusual activity within the system. Some metamorphic viruses use local or network replication to spread. If the honeypot file is placed in a network environment (e.g., cloud storage or a server), any attempts to download or transfer it between hosts can be detected via a network access trigger [17].

If malware operates solely in memory, without modifying files, it will be difficult to detect. Additionally, if the file system is bypassed and the virus functions only through infected processes,

honeypot files may fail to activate. Code obfuscation techniques can also complicate detection based on file content.

The standard honeypot creation method has certain limitations that can reduce its effectiveness. First, static honeypot files can be identified by attackers if they lack sufficiently realistic attributes, such as regular content updates and metadata changes [18]. Second, traditional honeypot documents react only to specific types of access and do not account for more complex attacks, such as memory-based threats or file system bypass techniques. Additionally, these honeypots may generate a high number of false positives, particularly when legitimate users or automated processes accidentally interact with them. This can overload security systems and complicate threat analysis [19].

Artificial intelligence can also be integrated into the system to enhance its reliability. By leveraging AI for threat assessment and improving system survivability, it becomes possible to develop a broader range of threat responses and ensure resilience against large-scale attacks[20].

To enhance the technology, the following actions have been proposed:

1. Dynamic honeypot files
2. Use of a distributed honeypot file system
3. Automated response to access attempts

Instead of static documents, dynamic files should be used, which automatically modify metadata such as creation date, last access time, or last modification date. This increases the credibility of the file and makes it more difficult to identify as a honeypot.

Placing honeypot files in various locations, such as internal servers, cloud storage, and publicly accessible resources, allows for tracking attacker activity across different network segments.

Upon detecting unauthorized access, the framework can automatically trigger additional security measures, such as blocking suspicious IP addresses, activating advanced monitoring, or notifying cybersecurity analysts for immediate response.

The subsystem is divided into key segments, each of which enhances security protocols, making it more challenging to identify the honeypot file among other data, Figure 1.

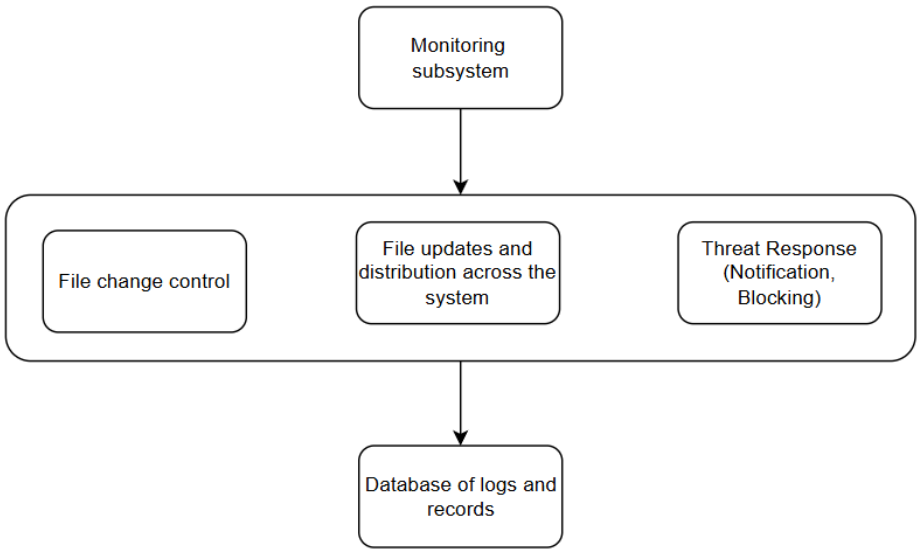


Figure 1: Automatic threat response subsystem

All these actions taken together will help increase the reliability of the system and improve protection against unauthorized intervention by attackers, Figure 2.

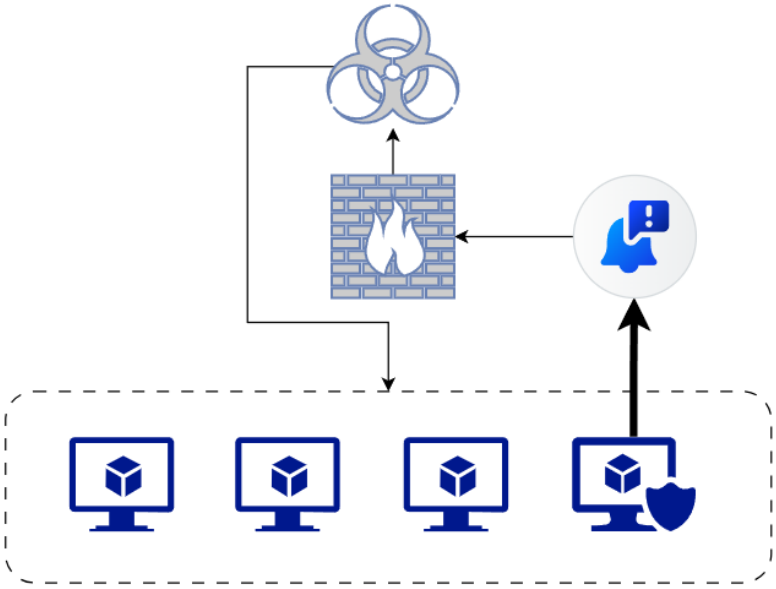


Figure 2: Scheme for detecting potential threats with a honeypot file

4.1. Dynamic honeypot files

Dynamic honeypot files are designed to eliminate the primary drawback of traditional honeypot documents—their static nature. Attackers can easily detect such files if their content or metadata remains unchanged for an extended period. To prevent this, the file must continuously change while maintaining its credibility.

This approach can be implemented using Windows built-in tools or Python scripting. The optimal solution is to create a script that automatically updates the file's metadata, makes minor changes to its content, and tracks all interactions with it.

First, the honeypot file must be prepared. It should be stored in a location where potential attackers can access it, such as a shared folder or an organization's file repository, Equation 1. If E(f) = 0, the file is created with the standard content S_0, where:

f = S_0, and S_0 = "Confidential Data - Access Restricted".

$$E(f) = \begin{cases} 1, & if\ file\ f\ exists \\ 0, & if\ file\ f\ does\ not\ exist \end{cases}, \quad (1)$$

Next, a mechanism for periodically changing the file's metadata must be implemented. In Python, this can be achieved using the os module, which allows modification of the last edited date. For example, a script can update the "Last Modified" attribute daily to make the file appear actively used [21].

The next step involves introducing random changes to the file's content. This can be done by adding small comments or variable phrases in hidden document fields. If a text file is used, several random lines can be inserted that do not alter the document's overall appearance but make it dynamically changing from the system's perspective.

At the end of the file, a random string R is added, which is generated using Equation 2, where U(a, b) represents a uniformly distributed random number in the interval [1000, 9999].

$$R = "\#LogEntry:" + U(1000,9999)\,, \quad (2)$$

where U(a, b) is a uniform distribution of a random number in the interval [1000, 9999]. After this, the new content of the file can be written as Equation 3, where S_{old} is the previous content of the file, and S_{new} is the updated content.

$$S_{new} = S_{old} + R, \quad (3)$$

After implementing these changes, the script should be configured for automatic execution. This can be represented as the file's last modified date T(f) being updated to the current time T_{now}, as shown in Equation 4, where T_{now} = timestamp(t) and t is the current time moment.

$$T(f) = T_{now}, \quad (4)$$

Thus, the honeypot file will be continuously updated, and any access to it will leave a digital trace, allowing for the analysis of potential attacker activity.

4.2. Use of a distributed honeypot file system

The distributed honeypot file system is designed to enhance the effectiveness of honeypot documents by deploying them across different parts of the IT infrastructure. If a honeypot file is placed in only one location, such as an internal server, an attacker may avoid interacting with it, reducing its effectiveness. To make the honeypot more appealing and increase the likelihood of an attacker engaging with it, multiple copies of the file should be created and distributed across various environments, such as local drives, cloud storage, internal file servers, or corporate document-sharing platforms [22].

A script is used to generate honeypot file copies and distribute them across multiple system locations. Before deployment, each file copy undergoes minor modifications to prevent them from being identical, making them appear more realistic. Once modified, the files are distributed to predefined locations, such as shared folders or directories synchronized with cloud storage.

To prevent excessive file duplication, the script checks if a file already exists in a given location before updating or overwriting it. During each execution, the script modifies the metadata of all file copies, adding a new random string to the content according to Equations 1-4. Afterward, the updated versions are automatically placed in the designated locations, as described in Equation 5.

The honeypot file is distributed across a set of locations $P = \{p_1, p_2, \ldots p_n\}$, where each file f_i in p_i receives a unique random string R_i.

$$S(f_i) = S_0 + R_i, \quad \forall_i \in [1, n], \quad (5)$$

4.3. Automated response to access attempts

Automated response to honeypot file access attempts enables the timely detection of potential threats and prevents further malicious activities within the network. If a honeypot file is opened or copied, the system must immediately record this event, send an appropriate notification to security administrators, and take additional actions, such as blocking the attacker's IP address or activating enhanced user activity monitoring [23].

The framework operates in the background mode, continuously monitoring the honeypot file for opening, modification, or copying. If an interaction occurs, the program logs the event in the security journal and sends a notification to a designated email address or system log.

The framework constantly checks the last modification time of the honeypot file f. The modification time is defined as T(f), while the current time is denoted as T_{now}, as described in Equation 6.

$$\Delta T = T_{now} - T(f), \quad (6)$$

If the difference ΔT changes, it indicates that the file has been opened or modified, as shown in Equation 7.

$$\Delta T > 0 \Rightarrow File\ changed. \quad (7)$$

If a file modification is detected, it is recorded in the event log L, which contains a list of modification timestamps $L = \{T_1, T_2, \ldots, T_n\}$. With each new access, an entry is added as $L = L \cup \{T_{now}\}$, where T_{now} represents the timestamp of the latest file modification.

Additionally, the framework can automatically add the attacker's IP address to the Windows Firewall block list, preventing further access attempts to the network [24]. The attacker's IP address is retrieved from the system log and denoted as $IP_{attacker}$. If the file is modified more than N times within a specified time interval ΔT_{block}, the attacker's IP address is added to the blocked address list B, as described in Equation 8.

$$\text{If } |L| > N \text{ за } \Delta T_{block}, \text{то } B = B \cup \{IP_{attacker}\}, \tag{8}$$

After this, the framework adds a rule to the Windows Firewall, setting $F_{block}(IP_{attacker}) = 1$, where F_{block} is the function that blocks access via Windows Firewall.

By combining all the aforementioned actions, the monitoring function can be represented as Equation 9.

$$\begin{cases} \text{If } \Delta T > 0, \text{то } L = L \cup \{T_{now}\} = 1 \\ \text{If } |L| > N \text{ за } \Delta T_{block}, \text{то } F_{block}\{IP_{attacker}\} = 1' \end{cases} \tag{9}$$

5. Experimental study of the effectiveness of the presented framework

The first step will be the creation of a honeypot file. After creating the file with pseudo-information attractive to attackers, the file is assigned a lure attribute.

Once the honeypot file is created, scripts are connected to it to ensure its dynamism and changes over time, so that the lure does not appear static.

In the case of Windows, this can be done through the Task Scheduler by creating a new task that will execute a script at specific intervals [25].

The next step will be the implementation of spreading the honeypot file across the system.

To automate the distribution process, Python can be used. Let the simultaneous existence of honeypot files be limited to fifty instances to avoid overloading the system with duplicates.

To make this process automatic, in the case of Windows, it is necessary to add a task to the Task Scheduler. The task is configured to execute the script periodically, for example, once a day or every hour. This solution ensures constant rotation of honeypot files in the system and increases the likelihood that attackers will interact with them [26].

To monitor access to the honeypot file, notification policies, such as those built into operating systems, can be used, or other options can be integrated if desired, as shown in Figure 3.

Finally, automatic response to triggers is configured, as shown in Figure 4.

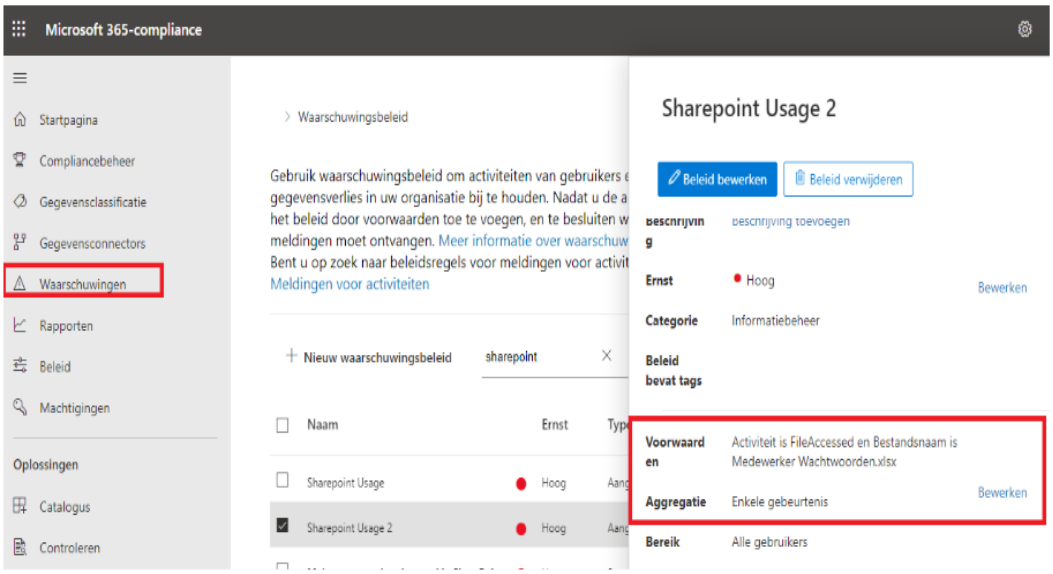


Figure 3: Create a notification policy in Microsoft 365

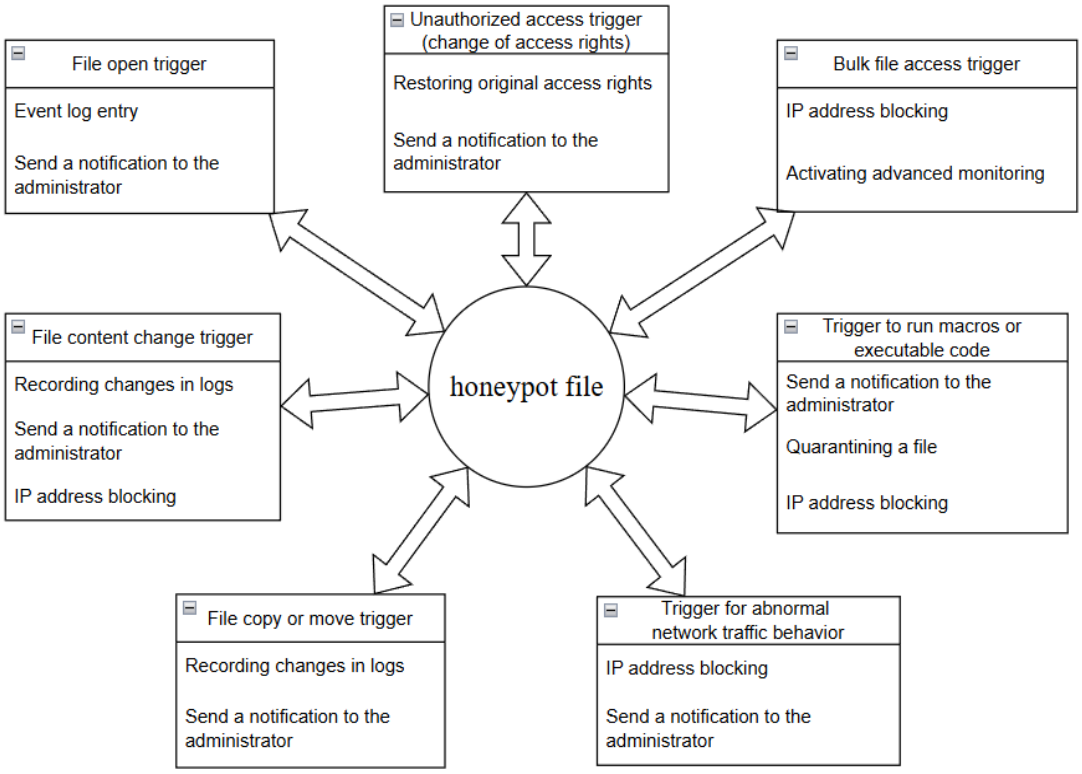
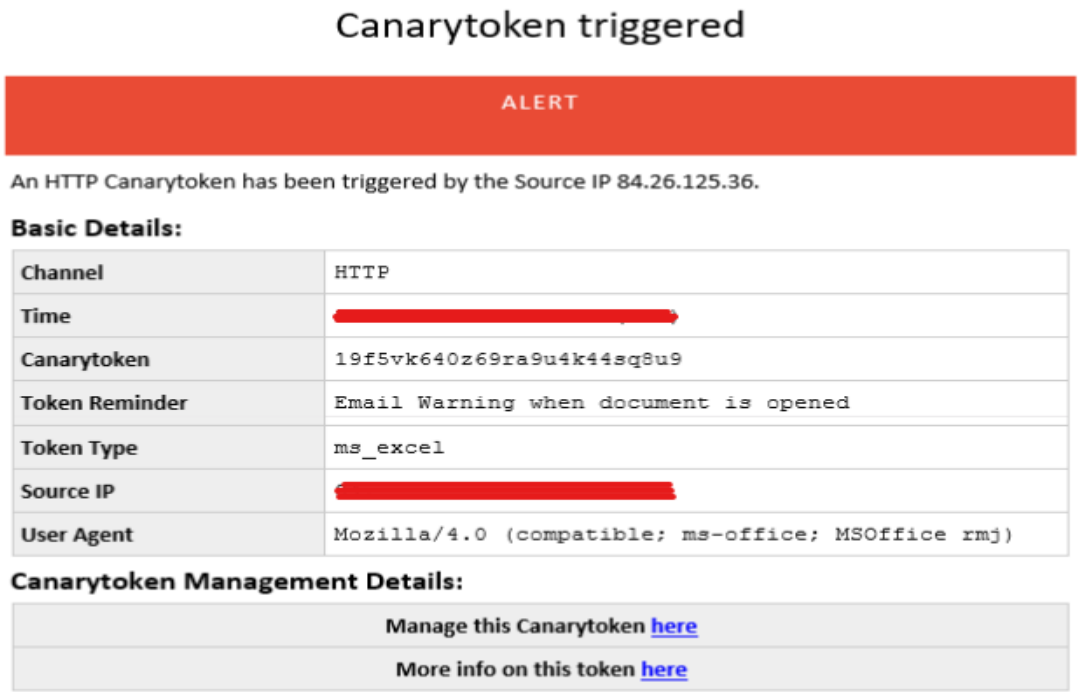


Figure 4: Honeypot file triggers and reactions to them

For the response mechanism to work effectively, it is necessary to run the script in the background during the operating system startup. For example, this can be done using the Windows Task Scheduler, which will ensure the program is launched automatically after the computer reboots. Thanks to this approach, the system will be able to respond instantly to suspicious activities without requiring constant administrator intervention [27]. Therefore, if someone attempts to download the flagged file, a notification will be sent to the linked email address, and to check the system for intrusion, it will suffice to review the notification, as shown in Figure 5.



Canarytoken triggered

ALERT

An HTTP Canarytoken has been triggered by the Source IP 84.26.125.36.

Basic Details:

Channel	HTTP
Time	[redacted]
Canarytoken	19f5vk640z69ra9u4k44sq8u9
Token Reminder	Email Warning when document is opened
Token Type	ms_excel
Source IP	[redacted]
User Agent	Mozilla/4.0 (compatible; ms-office; MSOffice rmj)

Canarytoken Management Details:

Manage this Canarytoken here
More info on this token here

Figure 5: File upload notification email.

To evaluate the effectiveness of the subsystem, 50 honeypot files were deployed in various environments, including local servers, cloud storage, and network resources. Over 30 days, 125 access attempts were recorded, of which 80 were identified as potentially malicious. From Table 1, it can be seen that the proposed solution had a false positive rate of 10%, while a standard Canarytokens file without improvements showed an average of 30%, indicating a high level of threat detection accuracy. The automatic IP blocking system for attackers was triggered in 95% of cases, ensuring a prompt response to threats. Activity monitoring showed that the system's average response time to a file access attempt is 2 seconds, allowing for timely security measures. As can be seen from Table 2, at least four out of seven triggers can be activated for each type of threat, which helps reduce the number of false positives and ensures responses only to potential threats.

Table 1
Comparison of methods

Methods	False positives	Efficiency	Response time	Reliability
Standard approach	30%	60%	1c	70%
Proposed framework	10%	95%	2c	95%

Table 2
Correspondence of triggers to different types of threats

Triggers / Threats	Viruses	Trojan Horses	Spyware programs	Worms	Ransomware	Metamorphic viruses
File open trigger	Yes	Yes	Yes	No	Yes	No
File content change trigger	Yes	Yes	No	No	Yes	Yes
File copy or move trigger	Yes	No	Yes	Yes	Yes	No
Unauthorized access trigger (change of access rights)	Yes	Yes	No	Yes	Yes	Yes
Bulk file access trigger	No	No	Yes	Yes	Yes	Yes
Trigger for abnormal network traffic behavior	No	Yes	Yes	Yes	No	Yes
Trigger to run macros or executable code	Yes	Yes	No	No	Yes	Yes

Overall, the system ensured stable operation without overloading resources, and its integration with cybersecurity mechanisms significantly reduced the risks of data compromise.

6. Conclusions

This paper presents an innovative framework for detecting cyber threats based on dynamic honeypot files, which combines automated monitoring, distributed placement of honeypot, and instant response to suspicious activity. The proposed approach effectively identifies a wide range of threats, including viruses, trojans, ransomware, and metamorphic viruses, by utilizing dynamically changing files with realistic metadata.

Key findings of the research include a reduction in false positives to 10% compared to conventional methods (30%), achieved through a combination of triggers such as file content changes, anomalous network traffic, and macro execution. Additionally, the system demonstrates high effectiveness in automated response, blocking 95% of attacker IP addresses through integration with Windows Firewall, with an average response time of 2 seconds. The framework is also scalable, supporting the distributed deployment of up to 50 honeypot files across various environments (local servers, cloud storage, corporate platforms), thereby increasing the likelihood of detecting malicious actors.

However, the system has certain limitations. For example, it is ineffective against malware that operates exclusively in memory or bypasses the file system. Additionally, detecting obfuscated code remains challenging and requires further refinement of analysis algorithms.

The proposed framework can be integrated into corporate security systems to enhance protection against attacks, social engineering, and automated cyberattacks. Future research could focus on expanding functionality to combat more sophisticated threats, such as AI-driven attacks, as well as optimizing resource usage when scaling the system.

This work makes a significant contribution to the development of proactive cybersecurity methods, demonstrating that the combination of dynamic honeypot technologies and automation can significantly enhance the security level of modern IT infrastructures.

Declaration on Generative AI

During the preparation of this work, the authors used Grammarly in order to: grammar and spelling check; DeepL Translate in order to: some phrases translation into English. After using these tools/services, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] R. Campbell, K. Padayachee, T. Masombuka, A survey of honeypot research: Trends and opportunities, in: Proceedings of the 10th International Conference for Internet Technology and Secured Transactions (ICITST), London, UK, 2015, pp. 208–212. doi:10.1109/ICITST.2015.7412090.

[2] D. Fraunholz, H. D. Schotten, An adaptive honeypot configuration, deployment and maintenance strategy, in: International Conference on Cyber Situational Awareness, Data Analytics and Assessment (Cyber SA), IEEE, 2017, pp. 1–8.

[3] A. Pauna, V. V. Patriciu, Enhancing cybersecurity with honeypot systems: A case study, in: Proceedings of the 11th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), IEEE, 2019, pp. 1–6.

[4] R. Gu, Z. Yang, Y. Ji, Machine learning for intelligent optical networks: A comprehensive survey, Journal of Network and Computer Applications, 2020.

[5] Z. D. Madison, Honeyhive – A Network Intrusion Detection System Framework Utilizing Distributed Internet of Things Honeypot Sensors, 2022.

[6] O. Savenko, S. Lysenko, A. Kryschuk, Multi-agent based approach of botnet detection in computer systems. In: Communications in Computer and Information Science 291 (2012) 171–180. https://doi.org/10.1007/978-3-642-31217-5_19.

[7] O. Savenko, S. Lysenko, A. Nicheporuk, B. Savenko, Metamorphic viruses' detection technique based on the equivalent functional block search, in: CEUR-WS, 1844 (2017) 555–569.

[8] Canarytokens, Canarytokens – Quick, Free, Detection for the Masses, https://canarytokens.org/generate.

[9] D. Fraunholz, H. D. Schotten, An adaptive honeypot configuration, deployment and maintenance strategy, in: International Conference on Cyber Situational Awareness, Data Analytics and Assessment (Cyber SA), IEEE, 2017, pp. 1–8.

[10] I. Obeidat, M. AlZubi, Developing a faster pattern matching algorithm for intrusion detection system, International Journal of Computing 18(3) (2019) 278–284. doi:10.47839/ijc.18.3.1520.
[11] Z. Peng, X. Gu, S. Natarajan, J. Zhang, Modeling social worm propagation for advanced persistent threats, 2021. https://doi.org/10.1016/j.cose.2021.102321.
[12] G. Kambourakis, C. Kolias, Honeypots for ransomware detection: A case study on WannaCry and LockBit, Computers & Security 95 (2020) 101823.
[13] S. Lysenko, O. Atamaniuk, O. Bokhonko, V. Vorobiyov, Method for detection of ransomware cyber threats based on honeypot, in: CEUR-WS, ISSN 2307-5732, 2023, pp. 300–309.
[14] A. Alsaheel, Y. Nan, L. Yu, ATLAS: A practical framework for adaptive threat detection in enterprise environments, in: IEEE Symposium on Security and Privacy (SP), IEEE, 2021, pp. 1–18.
[15] B. Eriksson, G. Pellegrino, A. Sabelfeld, Black Widow: Blackbox data-driven web scanning, in: IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2021, IEEE, pp. 1125–1142. doi:10.1109/SP40001.2021.00022.
[16] O. Savenko, S. Lysenko, A. Nicheporuk, B. Savenko, Approach for the unknown metamorphic virus detection, in: Proceedings of the 8th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, Bucharest, Romania, 2017, pp. 71–76.
[17] G. Markowsky, O. Savenko, S. Lysenko, A. Nicheporuk, The technique for metamorphic viruses' detection based on its obfuscation features analysis, in: CEUR-WS, Vol. 2104, ISSN: 1613–0073, 2018, pp. 680–687.
[18] G. Kambourakis, Z. Genç, Dynamic honeypot configuration to mitigate static detection in ransomware attacks, Computers & Security 96 (2020) 101923.
[19] R. Beuran, T. Inoue, Y. Tan, Realistic cybersecurity training via scenario progression management, in: European Symposium on Security and Privacy Workshops (EuroS&PW), IEEE, 2019, pp. 67–76. doi:10.1109/EuroSPW.2019.00014.
[20] N. Doukas, P. Stavroulakis, N. Bardis, Review of artificial intelligence cyber threat assessment techniques for increased system survivability, in: Malware Analysis Using Artificial Intelligence and Deep Learning, Springer International Publishing, 2021, pp. 207–222. https://doi.org/10.1007/978-3-030-62582-5_7.
[21] S. Sethuraman, T. Jadapalli, D. Sudhakaran, Flow-based containerized honeypot approach for network traffic analysis: An empirical study, Computer Science Review (2023) 5–10.
[22] M. Baykara, R. Das, A novel honeypot-based security approach for real-time intrusion detection and prevention systems, Journal of Information Security and Applications (2018) 103–116. doi:10.1016/j.jisa.2018.06.004.
[23] D. Fraunholz, M. Zimmermann, H. D. Schotten, SOAR-integrated honeypots for automated threat response, in: Proceedings of the 17th International Conference on Availability, Reliability and Security (ARES), ACM, 2022, pp. 1–10.
[24] T. Nguyen, M. Jones, Automated threat response in honeypot-enabled networks using dynamic firewall rules, in: IEEE International Conference on Cyber Security and Resilience (CSR), IEEE, 2021, pp. 1–9.
[25] R. Gupta, A. Patel, Automating security maintenance in Windows environments: A task scheduler approach, in: International Conference on Computational Science and Computational Intelligence (CSCI), IEEE, 2020, pp. 1–6.
[26] B. Alotaibi, K. Elleithy, Scalable honeypot deployment using Python scripting for enterprise networks, Journal of Cybersecurity and Privacy 1(2) (2021) 234–250.
[27] L. Johnson, C. Martinez, Persistent security automation in Windows: Leveraging task scheduler for background threat response, in: IEEE Symposium on Cybersecurity Applications and Technologies (SCAT), IEEE, 2022, pp. 1–7.