Journal of Network and Computer Applications 214 (2023) 103606

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Using rootkits hiding techniques to conceal honeypot functionality

Maryam Mohammadzad, Jaber Karimpour [*]

Department of Computer Sciences, University of Tabriz, Tabriz, Iran

ARTICLE INFO

Keywords:
Computer network
Cyber-attacks
Honeypot
Intrusion detection system
Rootkit
Security

ABSTRACT

Honeypot is one of the existing technologies in the area of computer network security. The goal of Honeypot is to create a tempting target for the attacker. The system that is considered as a Honeypot in the network includes the services and functions of a real system that the attacker sees as a normal system and enters for exploitation. In this way, Honeypot can monitor the behaviors, patterns, and tools used in various attacks. Certainly, if the intelligent attacker realizes the existence of this trap in the target network, he can design ways to bypass it. In this case, Honeypot practically loses its effectiveness. Therefore, the issue of hiding Honeypot is one of the primary challenges in this field. In this paper, a solution to this problem is presented. In the present paper, Honeypot is concealed using the concealment techniques used in Rootkits. We use application examples and theoretical analysis results to show that the proposed Honeypots concealment approach is strong against existing kernel-based Honeypots detection methods. Sebek, VMScope, and Qebek are three Honeypots that we choose for comparison purposes. The proposed Honeypot has been compared with them in virtualization, memory usage, and kernel modification. The experimental results show that the proposed hidden Honeypot in addition to low kernel modification has no track in the memory. Also, the proposed Honeypot does not use virtualization. It can successfully be concealed in the kernel of the target system without any effect on the target system. We also implemented the proposed algorithm on the example network and test all Sebek's detection methods on it. The experimental results show that the proposed kernel-based approach can bypass these detection methods.

1. Introduction

Computer networks with internet connections are vulnerable to exploits and threatened by hackers (Artail et al., 2006). Network administrators use several methods to protect their networks. Installing Firewalls and Intrusion Detection Systems (IDS) are examples of these methods (Krishnaveni et al., 2018). Monitoring malicious threats on the Internet and analyzing how attackers exploit vulnerabilities in the system can provide valuable information to network administrators so that they can apply effective protection mechanisms to their system. Therefore, there is a need for methods to gather information about attacker behaviors (Lackner, 2021). This information can also help to understand the methods and tools of attackers and identify unknown security vulnerabilities, and thus lead to better protection of the system and network. Honeypot is one of the modern technology in the area of IDS that provides a suitable interface for a successful attack. Honeypot does not provide real services to users, so any interaction with it is suspicious and indicates a malicious activity (Uitto et al., 2017; Rowe, 2019). In fact, the main purpose of Honeypot is to gather information about intruder's activities by luring attackers (Uitto et al., 2017; Tsikerdekis et al., 2018; Rowe, 2019). If we want to define a Honeypot, it is a computing system that is created to directly intract with intruders, it is designed to be attacked, recently many number of researches has been done to increase the level of its deception (Pittman et al., 2020). As mentioned Honeypots are designed to provide a tempting target to circumvent the attacker. Therefore a Honeypot will be very powerful when a large number of attacks successfully attacked it. Using of honeypots has two main advantages: First, attackers tend to Honeypot systems instead of real systems. Therefore the details of attacker's behavior can be achieved. Second, the defenders can be able to design better defense systems by achieved these data (Akshay et al., 2020; Liao et al., 2013).

When an attacker attacks a network, he/she identifies several systems on that network. All of these detected systems form a network and are called "botnets". Each of the systems in this network is called a "bot" (Jayasinghe et al., 2014). The attacker installs a malicious program on each of these bots, which he controls. This botnet can in a way identifies honeypot. The botmaster first commands to all bots to send malicious traffic to other systems. Based on whether the bots send this malicious traffic out or not, it can determine if the bot is a trap or a real vulnerable system (Wang et al., 2010; Su, 2010). Therefore,

* Corresponding author.
E-mail address: karimpour@tabrizu.ac.ir (J. Karimpour).

https://doi.org/10.1016/j.jnca.2023.103606
Received 12 January 2022; Received in revised form 20 January 2023; Accepted 20 February 2023
Available online 24 February 2023

there are some advanced attackers which try to identify the Honeypot and design some strategies to disable it (Li et al., 2020). Installation of a Honeypot in an enterprise network can lead to a more efficient and secure system, but if the attacker knows its existence, its values will be somewhat reduced (Sohal et al., 2018).

In general, network administrators need to create a system with hidden data capture tools to achieve the idea of deceiving attackers (Miah et al., 2020; Jayasinghe et al., 2014). Two different types of data capture tools are used in Honeypots: network-based and kernel-based (Almutairi et al., 2012). In the network-based tools data capturing can be done by recording the associated network traffic, but this approach is doomed to fail if the intruder utilizes encryption to connect to the Honeypot (Dornseif et al., 2004). To defeat the encryption and getting decrypted data, the Honeypot researchers decided to develop kernel-based data capture tools. In this approach, they can be access data after that decryption process performed by Operating System (OS) in the normal way (Enemy, 2003). When we use a kernel-based data capture tool for watching intruder activities in our Honeypot, an important question should be answered, how we can hide it?. The Honeypot becomes useless if we do not use any strong hiding method because a professional intruder can find and kill it.

“Hiding oneself from the enemy” is a common concept between Rootkit and Honeypot. The Rootkit was developed by the Blackhat community to hide malicious activity from system administrators and includes very powerful concealment techniques, some of which make changes at the kernel of the operating system and can hide system services, drivers, and programs. In this paper, to solve the honeypot detection problem, the hiding methods used in Rootkits have been used to hide the honeypot functionality.

1.1. Objective

Honeypot is widely used to study attacker behaviors. However, professional attackers may try to detect Honeypot and design attacks to disable it (Li et al., 2020). Some approaches have been proposed to solve the detection problem of Honeypot. Using a Virtual Machine Monitor (VMM) is a common idea that can be seen in the literature (Jiang and Wang, 2007; Song et al., 2015; Kouba, 2009), choosing a different hypervisor as a VMM in each of these strategies, has made them different from each other. Using a VMM as a platform for the Honeypot system can be a step towards hiding it, because it is outside the host and since the attacker is operating inside the host, it cannot detect the Honeypot. Therefore, Honeypot can record the attacker’s activities by using the VMM capabilities to dominate the host. However, using a VMM has its problems, the VMM is detectable and can be attacked and threaten Honeypot. Virtual machine identification is a critical issue for honeypots which use VMM to monitor attacker behavior (Lin et al., 2021). Security issues are challenging in virtualization and raise serious security concerns (Kapil and Mishra, 2021). Even if the VMM is not attacked, an attacker can compare the response speed of all systems on the network and detect the Honeypot. In Lin et al. (2021) an approach to detect virtual machine has been presented. This paper provides a probability-based thread scheduling analysis model to describe the time difference between physical machines and virtual machines. The results of this paper show that all type of virtual machines can be detect in this way.

In this paper, An approach to conceal the functionality of the Honeypot has been proposed. Kernel modification approaches and virtualization are two common traditional methods proposed to conceal the Honeypot. In the proposed approach, the Honeypot is inside the host itself, rather than being transferred to the virtual machine layer. As mentioned above, Rootkit has a collection of hiding techniques that can conceal itself in the target host. Therefore we can use these techniques to achieve our main goal, the concealment of monitoring functionality in the Honeypot System. DKOM is one of the strongest hiding techniques which have been used in the proposed approach. This is the first time that the monitoring tool of the Honeypot can be concealed in the kernel of Honeypot without using virtualization or memory and kernel modification.

Besides the concealment issue of Honeypot, there is another issue, suppose, attackers, use encrypted communications in their attacks, in this case, if the Honeypot is placed in the application layer of the host, all of the captured data will be useless, Therefore this is a primary issue to consider. To solve this problem, in the proposed solution, Honeypot is placed at the kernel level of the operating system. As another issues in the previous researches, Manipulation of system functions in previous Honeypots creates a suspicious kernel that can be detected by an attacker. The main advantage of the proposed solution is that, in addition to not using a VMM, the system functions of the Honeypot are not manipulated.

In summary, the contributions of this paper are as follows:

- In the proposed approach, for the first time, the kernel-level data capture tool is hidden inside the host, accessing all system events containing encrypted data.
- The proposed approach does not use VMM as the executive platform of the Honeypot. It has made the Honeypot fake system no different from the normal network systems in terms of system response speed, making it more difficult for the attacker to identify the Honeypot.
- The implementation of the Honeypot system’s data monitoring tool at the kernel level of the Operating System (OS) has given the proposed method all the benefits of VMMs, without any additional work on the binary translation to access events within the system.
- The proposed approach does not manipulate the system functions. Manipulation of system functions in the kernel of the Honeypot makes it suspicious and the attacker can easily detect the existence of the Honeypot.

1.2. Organization

The paper is organized as follows: Section 2 introduces Rootkits and present a description of their hiding techniques. Section 3 provides a review in researches published in Honeypot detection and detection prevention. Section 4 present proposed algorithm to hide data capture tool in High interaction Honeypot. Section 5 shows the result of implementation of proposed algorithm and finally, Section 6 presents a conclusion of the research.

2. Rootkits and hiding techniques

In general, attackers who attack networks or computer systems fall into two categories. ***WhiteHat*** - A whitehat attacker is a person who looks for security vulnerabilities in the system and then reports these security vulnerabilities to system administrators. ***BlackHat*** - A blackhat attacker is a person who looks for security flaws in the system and after finding flaws, performs malicious activities in it. A Rootkit is a type of malware that black hat attackers use for various purposes in their attacks. Rootkit represents stealth techniques which can gather information furtively, and prevent itself from being discovered by system administrators (Tsaur and Chen, 2010).

The main idea of rootkits techniques is access to system calls events’ data. It perform this by modifying kernel syscall table to execute new version of system function. In fact, A Rootkit is a type of malware that is designed to gain root access to a computer system while hiding itself from the user and the operating system. Such programs are usually used right after successful attack on a system. Attacker installs the Rootkit so she can easily return and gain privileged rights lately again Nadim et al. (2021). A well-designed Rootkit can hide files, data, processes, and network ports, and can typically survive a system restart. The effect of this stealthy design allows the Rootkit to perform malicious activities such as recording keystrokes of victim system (Kouba, 2009; Arnold, 2011).

During past few years, blackhats developed several Rootkits and used various techniques and system levels to implement them (Kouba, 2009). In fact, the evolution of rootkits is divided into four phases. In the first phase, a Rootkit just alter Unix's log files to hide the presence of certain users. In the second phase, it also replaces some important programs to hide users' activities. Substituting of programs can be easily found by integrity checking programs such as Tripwire. In the third phase rootkits aim to intercept user mode function requests and hide desired information in memory. In order to improve stealth level, rootkits and detectors are transferred to the kernel mode in the fourth phase. The targeted OS is also changed from the UNIX to Microsoft Windows (Arnold, 2011; Tsaur and Chen, 2010). Rootkits are programs that are permanently and secretly installed on a computer system. Rootkits can be divided into five different categories: user-level, kernel-level, hypervisor-level, bootkits, and firmware Rootkits (Nadim et al., 2021). Many concealment techniques such as Permission-based, Direct code modification, Direct data modification, and Hooks have been used in their implementation (Mysliwietz and Moonsamy, 2020).

Direct Kernel Object Manipulation (DKOM) is the modern hiding technique that is used in kernel-based Rootkits. DKOM can manipulate kernel data structures to hide processes and loaded drivers or change privileges, etc. Using DKOM technique, kernel-level rootkits can change the kernel data structures. The hide ability of the DKOM technique is greater than kernel hooking. Because, in this technique, objects change dynamically over time during normal operation. EPROCESS data structure is the OS kernel's representation of a process object (Nadim et al., 2021). FU and Fu To Rootkits are the first known Rootkits that use DKOM, which modifies the EPROCESS doubly linked list in Windows to hide the Rootkit processes and drivers (Kouba, 2009; Tsaur and Chen, 2010). For example, how to hide the driver is explained. There is a $MODULE\_ENTRY$ structure for each loaded driver in kernel, these structures are maintained in doubly linked list. $DRIVER\_OBJECT$ is the other data structure that contains a pointer to $MODULE\_ENTRY$. Each MODULE ENTRY structure contains $LIST\_ENTRY$ object with FLINK and BLINK pointers, DKOM alter this links to hide driver. Fig. 1 shows this process. In addition to processes, active ports can also be hidden by DKOM. Implementation of the DKOM technique is very difficult because if it is not implemented correctly, it will cause the system to crash (Nadim et al., 2021).

Some research has been done on the detection of kernel-based Rootkits. Researchers use learning-based methods because machine learning and deep learning have high accuracy for the automatic detection of known or unknown malicious software. Hardware performance counters in Singh et al. (2017), virtual memory access pattern of an application in Xu et al. (2017), and system call execution times in Luckett et al. (2016) are approaches to detection of kernel-based Rootkits, but none of them have the ability to detection of DKOM (Nadim et al., 2021). DKOM is the most powerful technique for hiding kernel-based Rootkits. In this paper, we proposed a hidden kernel-based honeypot to solve the two main problems: First, access to encrypted data of the attacker, Second, make kernel-based Honeypot undetectable is so the DKOM method is a good choice for achieving our goal.

In this paper we only focus on Windows Operating System (OS), the reason for choosing this OS is that this type of operating system is more popular than others, in often organizations, most employees are normal users that familiar with this OS so the administrator uses this OS for all systems of the network. If we are selecting this OS in Honeypot, the attacker thinks this system is a normal system of the network. The second reason to select Windows as the Honeypot's OS is that in the proposed method the DKOM has been used to conceal the Honeypot functionality, which this technique, is implemented in the kernel of Windows OS.

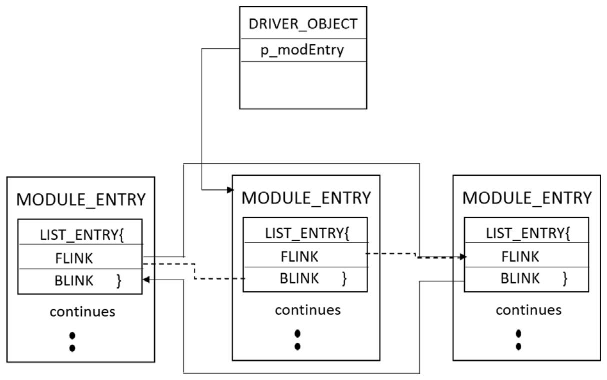


Fig. 1. Direct Kernel Object Manipulation (DKOM) Technique.

3. Related works

One of the main features of Honeypot is its high level of concealment, which helps the attacker to be completely deceived and log in. Honeypot hiding plays an important role in the attacker's decisions.

In 2003, the Honeynet project team (Enemy, 2003) introduced Sebek by publishing the paper "Know Your Enemy". Sebek is known as the first kernel-based capture data. To secretly monitor Honeypot data, The Honeynet project decided to use techniques of kernel-based rootkits to obtain data from inside the Honeypot's kernel. This idea led to the production of the Sebek data capture tool. Sebek aims to bypass encryption, due to the fact that the execution of any program in the operating system is available from within the kernel, allows the attacker to enter the system and then use the module already loaded on the kernel to record all attacker's activities. This tool provides the following features:

- Record keystrokes used in an encrypted session
- Recover files copied by Secure Copy Protocol (SCP) protocol
- Capture passwords used to log in remotely.
- Recover passwords to access burneye-protected binaries

Sebek's development cycle started with the Linux operating system and then spread to the OpenBSD, Solaris and Win32 operating systems. In fact, Sebek uses the Hooking technique to manipulate the system call table pointer of the read system function to point to its own version. As a result, when the user executes an action to create a system call to the read function, the system call table pointer points to the Sebek function, and finally, after reporting the data obtained from this call, the kernel continues to execute the main read function. This technique is shown in Fig. 2. To hide its client program, Sebek has used the techniques used in loadable kernel rootkits, such as adore Rootkit, by installing another piece of program as a cleaner after installing the main program piece, which removes the Sebek program name from the list of modules. Hiding in this way is not strong and there are ways to detect it. The data collected in the Honeypot is sent to the server, packets sent from the client to the server must be hidden from intruders. So Sebek's designers have devised measures to send their packages securely. Using a LAN to send packets to a server is unsafe. If Sebek uses UDP to send packets over the LAN, the attacker can detect the sending of these packets by listening to the network traffic, resulting in Sebek's presence in the host. So before sending packets via UDP, it changes the kernel so that users do not see these packets. In 2004, the Sebek data monitoring tool designed for high-interaction Honeypots failed with the NoSEBrEak code provided in Dornseif et al. (2004).

In 2007, in Jiang and Wang (2007) introduced their tool called VMscope. The main purpose of designing this tool is to provide a host-based monitoring tool that has all the capabilities of Sebek but is stronger and more stable than that. Due to the problems that Sebek

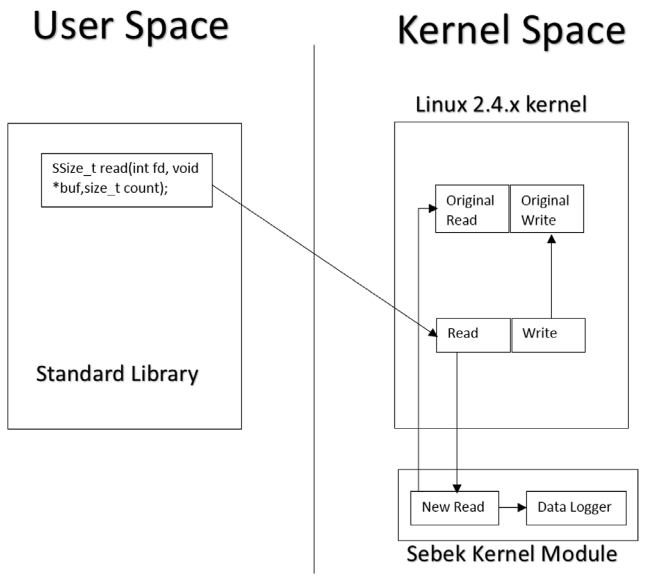


Fig. 2. Obtaining data of read function in Sebek (Miah et al., 2020).

monitoring tool has, an attempt has been made to use a method to obtain hacker information that, despite having access to all hacker activities such as system calls, remains hidden from intruders. Also in Sebek, the report file prepared with UDP packets was sent to the server, which created a way to identify the packet, which we discussed previously. In the design of this tool, the report file is stored in such a way that there is no need to send packages containing data to the server, and as a result, the issue of packets being leaked, unlike Sebek, is not discussed here at all. Obtaining data through system calls is the only way to access events within the operating system. Therefore, VMscope, like Sebek, collects system function data when it is called. A system function can be replaced by another function that pursues the same goal. For example, suppose a file is opened (sys-open system function call) and its contents are read (sys-read system call) and written to a network socket (sys-write system call), in which case three functions are called Instead, there is a sys-sendfile system function that does all three. Therefore, instead of calling the above three functions, the user may only reach his goal by calling the sys-sendfile function. This is taken into account in the design of the VMscope tool, and therefore all system functions are monitored. In fact, VMscope designs a highly interactive Honeypot like Sebek, except that, unlike Sebek, it does not make any changes to the Honeypot kernel by configuring its Honeypot on its chosen hypervisor, QEMU. Using a binary translation technique, this hypervisor allows access to events inside the guest system from within the host operating system, thus placing the monitoring tool outside the Honeypot, ie in the host operating system.

In 2009, Thomas Kuba (Kouba, 2009) Simulated the user behavior that communicates with the system via an SSH connection and examined the performance of his system using a virtual Honeypot. In this dissertation, pointing out that Sebek is installed on a honey trap and is easily identifiable, he concluded that he should use the capabilities of a virtual machine to monitor data. In this dissertation, among the two types of virtualization techniques described in the previous section, the privatization feature implemented in the Xen Hypervisor is used. As mentioned earlier, in this technique, to access system calls, the virtual machine must be installed in the guest system of the operating system in which changes have been made and adapted to this technique. Therefore, in the dissertation, a virtual Honeypot was created on it as a Honeypot and a network configuration of a network of Honeypots, using Xen hyperweavers and installing the Scientific Linux version, which is a modified version of the Linux operating system and is designed to be compatible with Xen. The VAccess library allows the guest operating system to be monitored. This library is designed for the Linux operating system and is able to capture everything inside the Linux kernel. The va-register-hook function in this library is able to hook the read and write functions in the guest operating system to access the desired data of these calls. This function is a function used in the implementation of this program when the simulated system of user behavior acts as a human user and enters the honey trap by creating an SSH connection. Although cryptography is used, the honey trap can be completely Access his information.

In 2010, the honeynet project team by publishing a paper (Song et al., 2015), while introducing the shortcomings of Sebek, introduced the Qebek Tools. In the previous section, it was mentioned that Sebek is a module that loads on the kernel due to the fact that an attacker can execute its own code like Sebek at the same kernel level, it can detect and disable it. In fact, Qebek was designed by the Honeynet project as a successor to Sebek to counteract the recognition and use of the benefits of using a virtual machine monitor. The Qebek tool consists of 5 main parts: interception module, breakpoint system, SVR routine, Introspection module and output module. Interception module invokes the qebek-bp-check function when changing the pointer to the guest machine instruction. Once the intruder is logged in, it can directly execute the system calls and create a new system call. The breakpoint system directly connects the implementation of system calls. SVR routine is called to read data from the virtual memory address extracted from the registers. Introspection monitors all events monitored in Sebek, including keystrokes, creation of processes and network activities. But network monitoring is different from Sebek. After obtaining information from inside the Honeypot, the obtained information should be output. Because this tool was created to fix Sebek bugs, the report file is created in the same format used in Sebek so that these files can be analyzed by existing analyzers.

In 2012, Shi et al. (2012) have presented a model for Honeypot which imitated and can understand network changes and adapt itself. This idea has made better the monitoring and concealment of the Honeypot.

In 2014, Daisuke et al. in Miyamoto et al. (2014) presents Intercept as a virtual Honeypot, this approach, uses Duplicate actual system to conceal functionality of Honeypot. This Honeypot was designed to web applications.

In 2015, Christoph et al. in Christoph et al. (2015) presents Apate as a physical Honeypot, in this approach kernel-level approach was used to concealment of Honeypot.

In 2019, Qi et al. (2019) Presented the CONCEAL framework. In this paper, concealment, detectability, and deterrence are the three main criteria for evaluating the framework. In this strategy, the defender tries to invalidate the attacker's information by performing address mutation, fingerprint anonymization, and configuration diversification.

In 2020 (Miah et al., 2020), is presented an approach to conceal cyber deceptions. In this paper, a model based on game theory is introduced. In this model, fake and real objects may naturally have a different distribution of specific system features and confuse the attacker. Also in this strategy, the defender can make some changes to real and fake objects to make detection more difficult.

In 2018, Dowling et al. (2018) have Used reinforcement learning to conceal Honeypot activity. In this research, they use augmented learning to design an adaptive honeypot that quickly learns ways to identify and adapts itself to these methods so that it can no longer be identified. The practical results of this research show that when a robot discovers the existence of a honeypot, a compatible honeypot learns how to overcome this initial diagnosis.

In 2021, Suratkar et al. (2021) have used Q-learning to conceal Honeypot activity. In this research, They use Q-learning in an SSH-based Honeypot called Cowrie to increase interaction with attackers for avoiding detection of Honeypot. Also in this proposed method, a dataset of Honeypot's collected data has been increased.

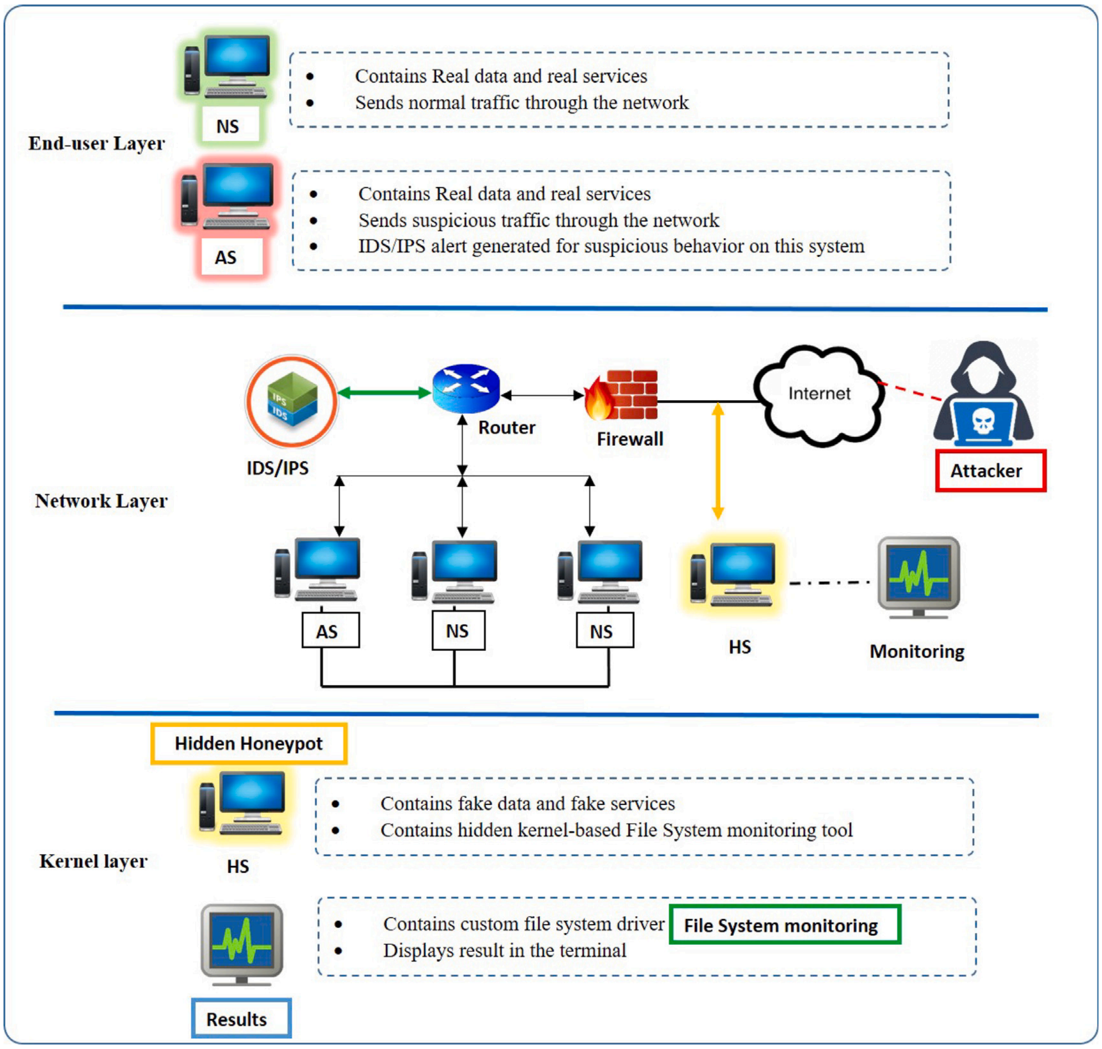

End-user Layer
NS
Contains Real data and real services
Sends normal traffic through the network
AS
Contains Real data and real services
Sends suspicious traffic through the network
IDS/IPS alert generated for suspicious behavior on this system
Network Layer
IDS/IPS
Router
Firewall
Internet
Attacker
AS
NS
NS
HS
Monitoring
Kernel layer
Hidden Honeypot
HS
Contains fake data and fake services
Contains hidden kernel-based File System monitoring tool
Contains custom file system driver
Displays result in the terminal
File System monitoring
Results

Fig. 3. Schematic view of proposed method.

Sebek (Enemy, 2003), VMScope (Jiang and Wang, 2007), and Qebek (Song et al., 2015) are three Honeypots that we choose for comparison purposes. The reasons for choosing these three basic tools are that the Sebek and Qebek are two kernel-based honeypots and the VMScope is the base of VMM-based honeypots. The other works did not focus to kernel-level concealment and using VMM in the honeypot.

4. The proposed method

In this section, the working of proposed conceal method of Honeypot and its various components is discussed in detail. In the proposed method, the exist systems will be categorized into three categories: Normal System (NS), At_risk System (AS), Honeypot System (HS). NSs are systems used by authorized users and operating on the network with authorized privileges. These systems are not attacked and do not threaten network security. ASs are systems which the network receives suspicious traffic from them and may have been attacked by a series of initial attacks. HSs are systems which have vulnerabilities that are intentionally placed on the network to lure the attacker to infiltrate them for monitoring by defender. Usually, professional hackers in the detection phase obtain the necessary information about the targets of the attack, then design the attacks according to their targets and can cause serious damage to the target system. Honeypot acts as a trap, luring these professional attackers into the fake system and monitoring the tools and methods used in the attacks. A professional attacker can detect a network trap and disable it. This detection is done through the effects that the HS has on its kernel and memory. In the proposed method, a solution for hiding functionality of HS is presented. Fig. 3 shows the shema of proposed method. In the proposed method, all NSs And ASs in the network have the required security mechanisms such as firewall, Intrusion Detection System (IDS) and Intrusion Prevention System (IPS), and the use of HSs is not used as a defense mechanism against attacks. So IDS detects any attack in each AS, and it generates the attack alarm. In this paper, the operation of the IDS is considered as a BlackBox. The proposed method focus on How to conceal functionality of HS in kernel level of OS.

The architecture shown in Fig. 3 contains following modules:

- Attacker
- Hidden Honeypot
- File System Monitoring
- Results

In the following of this section each of these modules will explained in details.

4.1. Attacker

An attacker is a malicious person trying to gain unauthorized access to a system or network. The attacker can be a person or a bot. Since Honeypot lacks useful services for authorized users, any request that

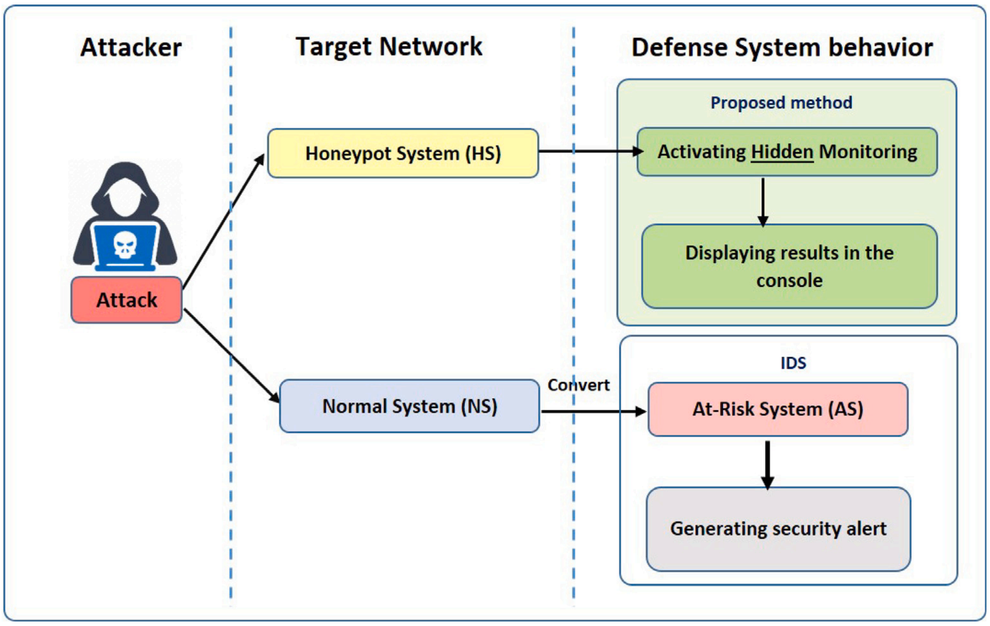


Fig. 4. The general flow of the attacker's interaction with target network in the considered architecture.

enters the system is considered an attack. The general flow of the attacker's interaction with target network in our architecture is shown in Fig. 4. The attacker connects to the target network by sending his command and get response from victim. If the system that the attacker targeted is one of the NSs in the network, This system then becomes a AS and the IDS generate a security alert. The process of the IDS is considered in the paper as a BlackBox. If the system that the attacker targeted is one of the HSs then the monitoring activated and shows results in the output console. The monitoring process is done hiddenly, the proposed method used DKOM technique to hide this part of monitoring process.

4.2. Hidden Honeypot

The hiding process is the main step in the proposed method. In this paper, DKOM technique was used to hide data capture tool in the Honeypot. This technique is one of the strongest Rootkit techniques to hide any process or any loaded driver in a system kernel. The DKOM technique to hide a loaded driver is shown in Fig. 1. This is a basic DKOM technique that was used in Fu and FuTo Rootkits. Unfortunately, these Rootkits detected by Unti-Rootkits softwares such as GMER (Tsaur and Chen, 2010; Kouba, 2009). Woei-Jiunn Tsaur and Yuh-Chen Chen in Tsaur and Chen (2010) presented a new hidden driver. In that paper, To increase the level of hiding, the basic DKOM have been done in 5 steps. This paper also uses these 5 steps to improve hiding of the data capture tool. These steps are listed below:

- Removing Object Drivers and Object Devices from Object Directory
- Removing Object Drivers from Driver Object_Type
- Removing Object Devices from Device Object_Type
- Removing Drivers from PsLoadedModuleList
- Altering Object Driver Appearance

Object Directory, Object_Type and PsLoadedModuleList are three structure that keep objects. For purpose of hiding we applied DKOM in these structures. Normally, to change kernel objects such as processes, we must enter the Object Manager object structure in the kernel. The object management structure is the central part of access to kernel objects, which provides common functions such as creating, deleting, and protecting all objects. One of the advantages of DKOM is that it bypasses object management, which makes it easier to bypass all object access checks and therefore more difficult to detect. One of the limitations of DKOM is that it can only modify objects that the kernel stores in memory. For example, the OS keeps a list of all running processes in a structure in memory so that this structure can be changed to hide a process. On the other hand, there is no data structure in memory that holds a list of all the files in the file system, so DKOM is not able to hide the files. Despite this limitation, DKOM is able to hide processes, device drivers, and ports. Our goal is to hide the driver of our monitoring tool therefore DKOM will be a good option for our purpose.

As mentioned earlier, all operating systems store information about drivers and processes in the form of structures or objects in memory. Therefore, we can change these structures directly without the need to hook. One of the first places an attacker can look for the effects of a surveillance system is a list of drivers. An attacker could easily get a list of all the drivers running on the system by running the driverquery.exe command. This is the same problem with the Sebek monitoring tool that after installing the Windows version, the name Sebek.sys is fully visible in the list of drivers, and despite all the mechanisms used to hide this tool, as mentioned before, Sebek can Easily defeated.

The kernel holds the path of the drivers using an object called MODULE-ENTRY. In fact, for every driver that is loaded in the kernel, a MODULE-ENTRY structure is created for it. These structures are stored in memory as a two-way linked list. There is another data structure called DRIVER-OBJECT, which has a pointer to the MODULE-ENTRY drivers, which is located at 0×14 inside this structure. So first we create a pointer that points to the MODULE-ENTRY drivers, then we move it to get to the MODULE-ENTRY driver that we want to hide. The pseudo code for obtaining a pointer is as follows:

$PMODULE-ENTRY\, pm-current;\ pm-current =* ((PMODULE-ENTRY\, *)((DWORD)DriverObject{+}0{\times}14));$

The Fig. 1 shows the two-way link list and how the DKOM algorithm works with continuous directional lines. As can be seen in the figure, the $MODULE - ENTRY$ for the driver has a member called LIST-ENTRY that contains two parameters that are of particular importance to us. The parameter that points to the next driver in the list is FLINK and the parameter that points to the previous driver in the list is BLINK. The pseudo-code for running this algorithm is as follows:

$*\ ((PDWORD)pm - current- > le - mod.Blink) = (DWORD)pm - current- > le - mod.Flink;$

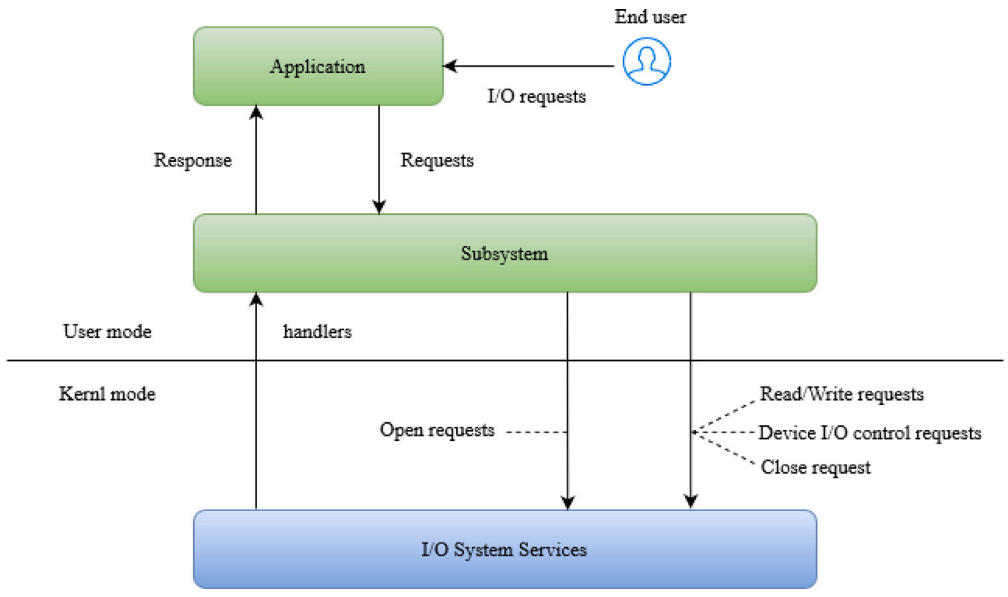


Fig. 5. I/O Request Packet (IRP).

$pm - current- > le - mod.Flink- > Blink = pm - current- > le - mod.Blink;$

In this pseudo code, the pm-current code points to the MODULE-ENTRY of the driver that we want to hide. In this way we were able to hide a custom driver from the list of loading drivers in the kernel.

4.3. File system monitoring

File system monitoring is a requirement for any modern data security strategy. In the proposed approach, due to the importance of file system monitoring, the process of monitoring the attacker's behavior is limited to monitoring the honeypot file system. The proposed approach could be expanded to monitor other parts of Honeypot. A file system filter driver is a driver that modifies file system behavior. It can change the behavior of the file system in our favor. This driver is a kernel module that runs as a part of the OS. In fact, the file system filter driver allows Windows users to be able to monitor access to directories, open, close, or modify files of the kernel. The presented honeypot uses this driver's capabilities to monitor file system events in the honeypot kernel.

4.3.1. I/O Request Packet (IRP)

When a user attempts to communicate with kernel-level he sends an IRP to the kernel. This IRP is handled by kernel's Major Functions that have a variety of type. Fig. 5 shows detail of IRP mechanism.Some of the major function that handles IRPs are: IRP_MJ_CLOSE, IRP_MJ_CREATE, IRP_MJ_READ and IRP_MJ_WRITE. In this paper, we created a handler to IRP_MJ_CREATE request to monitor the creation or opening of a file. At first, we must initialize the driver object dispatch table to set up our own create handler for this type of request, Algorithm 1. Nothing is done in the FsFilterDispatchPassThrough controller and the normal procedure is followed. The algorithm sets the object driver's of all major functions to normal procedure through a while loop. At the end of the algorithm Our goal is to implement a controller for IRP_MJ_CREATE packages. Algorithm 1 set the driver object of all major functions through a while loop. At the end of the algorithm the driver object of CREATE major function has been set to FsFilterDispatchCreate. So in FsFilterDispatchCreate we have to extract the file name from an object called PFILE_OBJECT and print it in the output and then continue with the normal procedure. Note that the PFILE_OBJECT object has the correct file name only if the CREATE operation is performed. The Algorithm of the FsFilterDispatchCreate controller is shown in Algorithm2.

```
Algorithm 1: Setting Controller to all of IRP Packets except CREATE
Result: Setting Controller to all of IRP Packets except CREATE
Let IRP_MJ_MAXIMUM_FUNCTION be the number of all types of IRPs
Let FsFilterDispatchPassThrough be the original Controller for each types
Let MajorFunction be the list of Major functions.
while a new function exist in the list (i from 0 to IRP_MJ_MAXIMUM_FUNCTION) do
    $DriverObject - -MajorFunction[i] = FsFilterDispatchPassThrough;$
end
$DriverObject - -MajorFunction[IRP - MJ - CREATE] = FsFilterDispatchCreate;$
```

```
Algorithm 2: Controller algorithm of FsFilterDispatchCreate
Result: IRP-MJ-CREATE IRP Handler
NTSTATUS FsFilterDispatchCreate( in PDEVICE-OBJECT DeviceObject, in PIRP Irp)
$PFILE - OBJECTpFileObject = IoGetCurrentIrpStackLocation(Irp)- > FileObject;$
$Print(pFileObject- > FileName);$
return FsFilterDispatchPassThrough(DeviceObject, Irp);
```

4.4. Results

The honeypot has several internal components:

- Hidden file system filter driver: To monitor attacker behavior.
- Terminal: To display reports of the attacker's behavior in real time.
- Other Utilities:To provide some common services and applications to make Honeypot look real.

Fig. 6 shows the algorithm of monitoring and displaying the results. All of gathered information about attacker, have displayed in the terminal and donot store in the memory. This causes to Honeypot donot have any effect in the memory so undetectable by detection ways which uses memory affectness. Logging and logs sending to the display termina are performed in parallel to prevent performance loss. Fig. 7 shows the

Table 1
Virtual network configuration settings.

Endpoints	Operating System (OS)	Virtual Adaptor	Vulnerability	Firewall	Automatic update	Installed services	Open ports
NS1	Windows XP SP3	NAT & Host-only	Low	Enabled	Enabled	SSH, SMB	None
NS2	Linux Ubuntu 8.04	NAT & Host-only	Low	Enabled	Enabled	SSH, SMB	None
HS	Windows XP SP3	NAT & Host-only	Very High	Disabled	Disabled	SSH, SMB	22,135,139,445

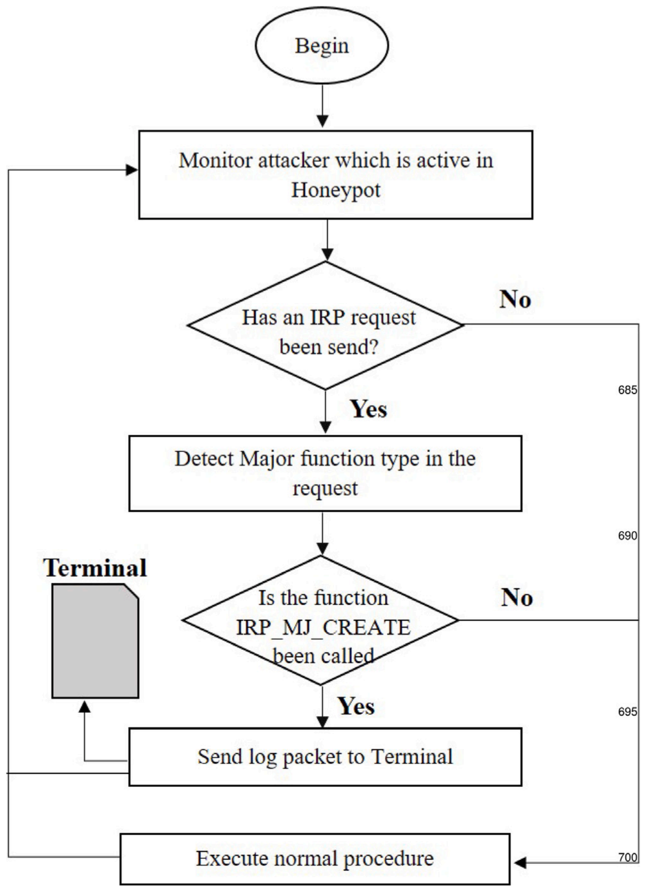
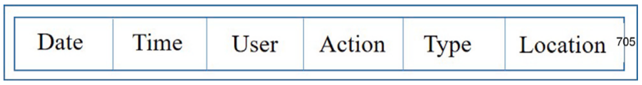

Begin
Monitor attacker which is active in Honeypot
Has an IRP request been send?
No
Yes
Detect Major function type in the request
Terminal
Is the function IRP_MJ_CREATE been called
No
Yes
Send log packet to Terminal
Execute normal procedure

Fig. 6. Algorithm of monitoring and displaying results.



Date	Time	User	Action	Type	Location

Fig. 7. The log format of the monitoring module sent to display terminal.

log format where the "User" field indicates whether the user is either attacker or authorized user. Because of any interaction with Honeypot is unauthorized so this feild already set to "Attacker" . The "Action" is the field used to indicate the operation type of attacker, it can be either CREATE or OPEN action. The "Type" feild is used to determine the attacker is trying to open/create a file or folder. The "Location" feild used to determine location of targeted file/folder in both of CREATE and OPEN actions.

5. Case study and analysis

5.1. Case study

The case study is used to validate the effectiveness of the proposed concealment method in kernel-based Honeypot, the typical network topology is utilized in this paper, which is shown by Fig. 8. Using virtual networks helps us to simulate all the features of a real system in a small environment. Virtual networks are also less expensive to maintain and will be a good option for implementing our proposed concealment method. There are three endpoints in the network, and the network configuration settings shown in Table 1. For each virtual machines, 2 types of network adaptor Host-only and Network Address Translation (NAT) adjusted. To simulate a real attack process, All of the network systems must be able to connect to each other, to achieve this, we need to use Internet Protocol (IP) address for each guest systems. A unique IP address assigned for them by the Host-Only network adaptor. Also the NAT network used to provide an internet connection for guest machines of the virtual network.

We need a vulnerable system in our virtual network as HS. There are many ways to create this vulnerability, but we are looking for the easiest way to create a vulnerable laboratory system. To achieve this, it makes sense to use an intact version of Windows XP. Because based on studies on the types of attacks on Windows XP, we came to the conclusion that an intact version of Windows that has not been updated has penetration ways that even novice and non-professional hackers can also easily penetrate. This is the reason why we use the Windows XP OS. HS has four critical open ports and vulnerable versions of OS, SSH, and SMB. Table 2 shows the list of already detected vulnerabilities in these critical ports. Assume that attacker has Root access in the Attack Host, which is the starting point of a network attack. The attack goal is to open some sensitive files in the HS. Kali Linux is a Debian-based Linux distribution designed to perform intrusion testing. In order to simulate an initial attack on the designed network to validate the proposed method, This type of OS has been used.

To test the hiding ability of the proposed DKOM-based HS, we simulate an attack on the considered network. The strategy used in the simulated attack has 4 steps that have listed below:

- IP scanning
  The network IPs obtained by Nmap scanner on the considered network is shown in Fig. 9. This process has been done by the attacker. The result of this search indicates that our network is configured correctly.
- Port scanning
  Port scanning has been done by nmap $-O\langle IPAddress\rangle$ command. In this command, the parameter -O also specifies the type of target OS. Tables 3 and 4 show the results obtained by Nmap for target HS. This process has been done by the attacker.
- Vulnerability testing
  According to the list of open ports in the HS, the attacker starts to test all open ports listed in Table 3. The Nmap command $-p\langle port\rangle - scriptsmb - check - vulns - -script - argsunsafe = 1$ indicates the presence or absence of vulnerabilities in certain port. The network resource vulnerabilities obtained by Nmap scanner on 192.168.56.102 IP address which belongs to HS are shown in Table 5. This process has been done by the attacker. Conficker is a fast-spreading worm that targets a vulnerability (MS08-067) in Windows operating systems. CVE-2009-3103 is array index error in the SMBv2 protocol implementation in srv2.sys in Microsoft Windows OSs allows remote attackers to execute arbitrary code or cause a denial of service. According to the results of the Honeypot vulnerability test, this system has a vulnerability of MS08-067 and the attacker can exploit this vulnerability.

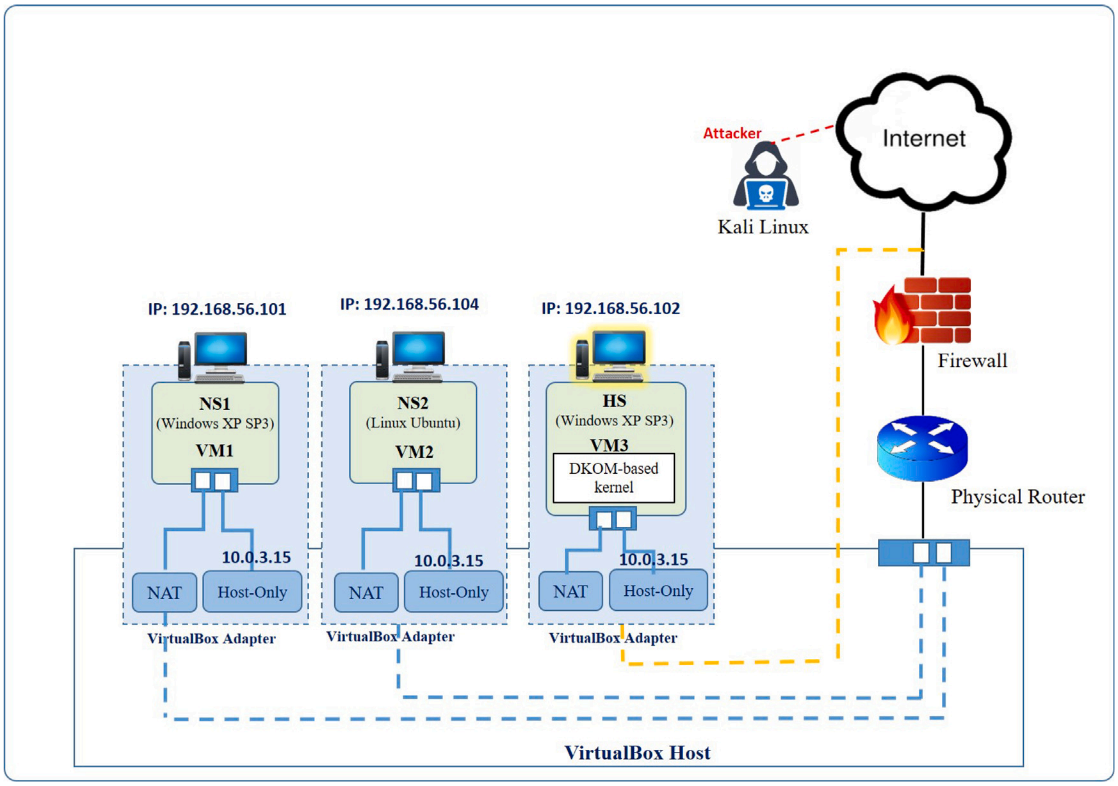


Fig. 8. Implemented virtual network topology.

Table 2
Firewall ports vulnerabilities.

Port	Vulnerability	Affected software	Description
445	MS08–067	Microsoft Windows 2000, Windows XP, and Windows Server 2003	The vulnerability could allow remote code execution if an affected system received a specially crafted RPC request.
139	CVE-2017–0174	Microsoft Windows	Windows NetBIOS allows a denial of service vulnerability when it improperly handles NetBIOS packets.
135	CVE-2020–7589	Microsoft Windows NT, Windows XP	There is a RPC vulnerability in Windows NT where a malformed request to port 135 could cause denial of service (DoS).
22	CVE-2017–3819	Secure Shell (SSH) subsystem	This vulnerability can be triggered via both IPv4 and IPv6 traffic. An established TCP connection towards port 22, the SSH default port, is needed to perform the attack.

Table 3
Result of Nmap for port scanning on the HS (open ports list with details).

Port	State	Service
22	open	ssh
135	open	msrpc
139	open	netbios-ssn
445	open	microsoft-ds

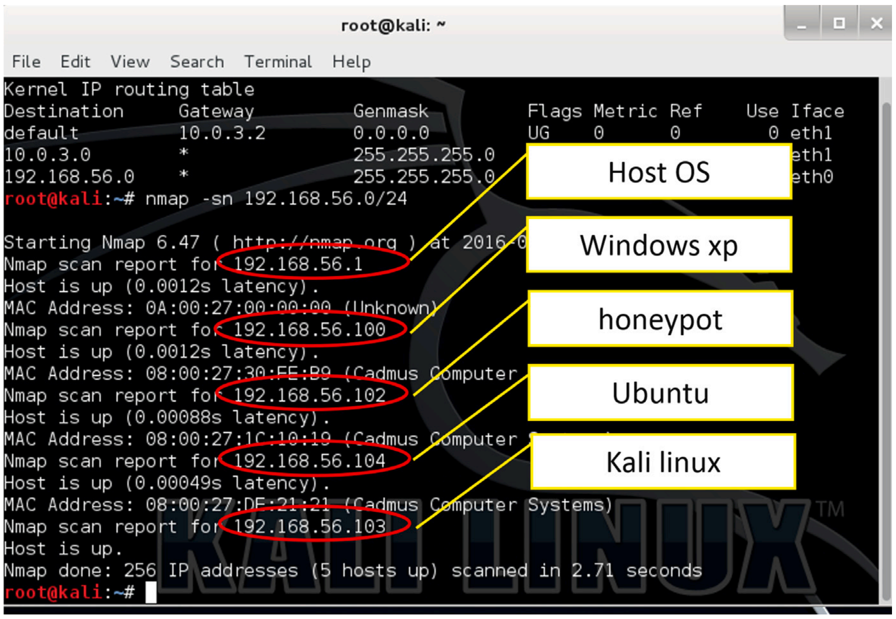
• Penetrate to the victim system (Honeypot)
Metasploit is an open-source framework that helps developers to create working exploits as new vulnerabilities are discovered. Metasploit is known as the best vulnerability assessment and exploit development tool. Penetration Testers use Metasploit to check vulnerabilities in the targeted system and run a suitable exploit on the targeted system. Metasploit can be used on different operating systems like Windows, macOS, and Linux. We use Kali Linux for Penetration Testing in the considered virtual network.
The attacker infiltrates the HS system immediately after finding the vulnerability. As soon as the attacker enters the system, the file system monitoring module starts monitoring, see Alg. Fig. 6. The monitoring System can instantly track all of his activities as he explores files and folders. In our simulation, the attacker infiltrates the HS system on 2021-09-07 for the first time and create a file in drive C, the attacker then return to the HS on 2021-09-07 for the second time and open the file Me.txt from drive C. File system monitoring module could successfully monitor this attack. The result of monitoring is shown in Table 6.

Table 4
Result of Nmap for port scanning on the HS (running system information).

Mac address	Device type	Running	OS detail	Network distance
08:00:27:1C:10:19	general purpose	Microsoft Windows XP	SP2-SP3	1 hop

Table 5
Result of Nmap for vulnerability testing on the considered virtual network.

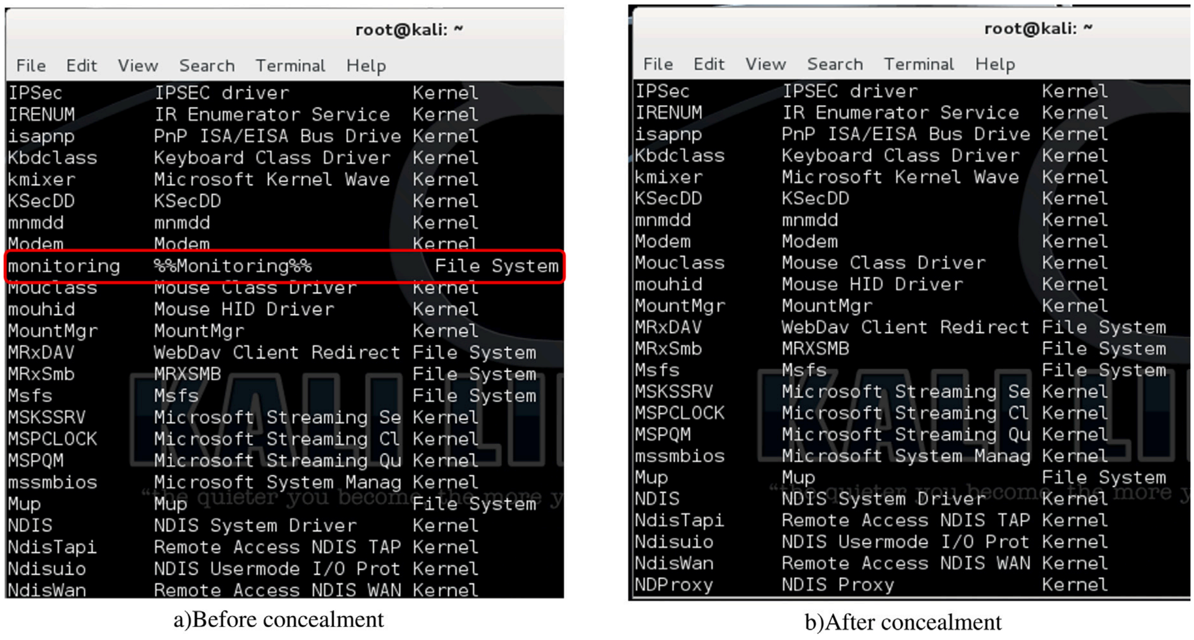
Endpoint	OS	Port	Vulnerability				
			MS08–067	Conficker	CVE-2009–3103	MS06–025	MS07-029
HS	Windows XP SP3	445	VULNERABLE	Likely CLEAN	NOT VULNERABLE	NO SERVICE	NO SERVICE



```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         10.0.3.2        0.0.0.0         UG    0      0        0 eth1
10.0.3.0        *               255.255.255.0                           eth1
192.168.56.0    *               255.255.255.0                           eth0
root@kali:~# nmap -sn 192.168.56.0/24

Starting Nmap 6.47 ( http://nmap.org ) at 2016-0
Nmap scan report for 192.168.56.1
Host is up (0.0012s latency).
MAC Address: 0A:00:27:00:00:00 (Unknown)
Nmap scan report for 192.168.56.100
Host is up (0.0012s latency).
MAC Address: 08:00:27:30:FE:B9 (Cadmus Computer
Nmap scan report for 192.168.56.102
Host is up (0.00088s latency).
MAC Address: 08:00:27:1C:10:19 (Cadmus Computer
Nmap scan report for 192.168.56.104
Host is up (0.00049s latency).
MAC Address: 08:00:27:DE:21:21 (Cadmus Computer Systems)
Nmap scan report for 192.168.56.103
Host is up.
Nmap done: 256 IP addresses (5 hosts up) scanned in 2.71 seconds
root@kali:~#
```

Fig. 9. Result of Nmap for IP scanning on the considered virtual network.



```
IPSec        IPSEC driver             Kernel
IRENUM       IR Enumerator Service    Kernel
isapnp       PnP ISA/EISA Bus Drive   Kernel
Kbdclass     Keyboard Class Driver    Kernel
kmixer       Microsoft Kernel Wave    Kernel
KSecDD       KSecDD                   Kernel
mnmdd        mnmdd                    Kernel
Modem        Modem                    Kernel
monitoring   %%Monitoring%%             File System
Mouclass     Mouse Class Driver       Kernel
mouhid       Mouse HID Driver         Kernel
MountMgr     MountMgr                 Kernel
MRxDAV       WebDav Client Redirect   File System
MRxSmb       MRXSMB                   File System
Msfs         Msfs                     File System
MSKSSRV      Microsoft Streaming Se   Kernel
MSPCLOCK     Microsoft Streaming Cl   Kernel
MSPQM        Microsoft Streaming Qu   Kernel
mssmbios     Microsoft System Manag   Kernel
Mup          Mup                      File System
NDIS         NDIS System Driver       Kernel
NdisTapi     Remote Access NDIS TAP   Kernel
Ndisuio      NDIS Usermode I/O Prot   Kernel
NdisWan      Remote Access NDIS WAN   Kernel
```

```
IPSec        IPSEC driver             Kernel
IRENUM       IR Enumerator Service    Kernel
isapnp       PnP ISA/EISA Bus Drive   Kernel
Kbdclass     Keyboard Class Driver    Kernel
kmixer       Microsoft Kernel Wave    Kernel
KSecDD       KSecDD                   Kernel
mnmdd        mnmdd                    Kernel
Modem        Modem                    Kernel
Mouclass     Mouse Class Driver       Kernel
mouhid       Mouse HID Driver         Kernel
MountMgr     MountMgr                 Kernel
MRxDAV       WebDav Client Redirect   File System
MRxSmb       MRXSMB                   File System
Msfs         Msfs                     File System
MSKSSRV      Microsoft Streaming Se   Kernel
MSPCLOCK     Microsoft Streaming Cl   Kernel
MSPQM        Microsoft Streaming Qu   Kernel
mssmbios     Microsoft System Manag   Kernel
Mup          Mup                      File System
NDIS         NDIS System Driver       Kernel
NdisTapi     Remote Access NDIS TAP   Kernel
Ndisuio      NDIS Usermode I/O Prot   Kernel
NdisWan      Remote Access NDIS WAN   Kernel
NDProxy      NDIS Proxy               Kernel
```

Fig. 10. List of drivers at perspective of attacker after and before hiding.

Table 6
Result of monitoring module.

Date	Time	User	Action	Type	Location
2021–09–07	15:30	Attacker	CREATE	File	C:
2021–09–08	08:00	Attacker	OPEN	File	C:Me.txt

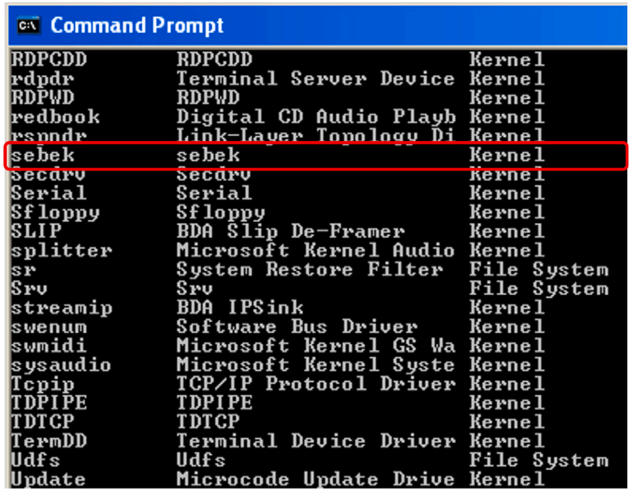
The main purpose of this paper is to provide a solution for the concealment of Honeypot functionality. In this simulation, attacker run driverquery.exe and look at all running drivers. To validate the proposed concealment method, this process has been done before and after applying the concealment method. The results show that the DKOM technique has been able to successfully hide the functionality of Honeypot from the attacker Fig. 10.

5.2. Analysis of results

The results of the proposed Honeypot concealment method can be analyzed from various aspects. In this paper, we have selected one kernel-based honeypot Sebek and three VMM-based honeypot to perform comparison. In the following, we analyze the results from aspects of detection, and response time.

1. Detection
Sebek is a kernel-based Honeypot, which can be compared to the proposed solution in terms of detection issues. To analyze the result of the concealment, we installed a Sebek tool on one of the virtual network systems and executed the driverquery.exe command on it. It was observed that Sebek's module name is in the list of drivers Fig. 11. While in the same environment we executed the same command on our Honeypot system and it was observed that there is no trace of the hidden monitoring tool in the list of drivers Fig. 10.
In Dornseif et al. (2004) some ways to detect Sebek tools has been introduced. In this section, we describe how the proposed concealment method is resistant against them.

   - **dd attack**
   The Sebek inserts the data obtained from a system call into packets and sends the packets out of the host via the UDP stream by adding the header, magic numbers, the source port, and the destination port. In other words, for one-byte data, a packet containing 100 bytes of data is sent to the server. Now if an attacker sends a READ request to read one byte of memory and runs this command 100 times per second, Sebek causes network congestion and slows down the server so much, that the response time to the ping request Increases from 0.8 ms to 4800 ms indicates the presence of a monitoring tool.
   Proposed method: In the proposed concealment method, the data obtained from the file system filter driver is printed on the output screen and can be viewed in real-time. Therefore, no report file is exported to cause congestion.
   - **Searching for modules loaded on the kernel**
   The Sebek is a kernel-based module that uses a random number smaller than 1 billion to the name of the module. The module name can be found from the list of loaded modules by creating an exploratory identification.
   Proposed method: Our monitoring tool is also kernel-based and is loaded on the kernel. Using the DKOM technique to hide the module provides Honeypot undetectable even if the attacker looks for it through the list of modules loaded on the kernel.



```
RDPCDD       RDPCDD                  Kernel
rdpdr        Terminal Server Device  Kernel
RDPWD        RDPWD                   Kernel
redbook      Digital CD Audio Playb  Kernel
rspndr       Link-Layer Topology Di  Kernel
sebek        sebek                   Kernel
Secdrv       Secdrv                  Kernel
Serial       Serial                  Kernel
Sfloppy      Sfloppy                 Kernel
SLIP         BDA Slip De-Framer      Kernel
splitter     Microsoft Kernel Audio  Kernel
sr           System Restore Filter   File System
Srv          Srv                     File System
streamip     BDA IPSink              Kernel
swenum       Software Bus Driver     Kernel
swmidi       Microsoft Kernel GS Wa  Kernel
sysaudio     Microsoft Kernel Syste  Kernel
Tcpip        TCP/IP Protocol Driver  Kernel
TDPIPE       TDPIPE                  Kernel
TDTCP        TDTCP                   Kernel
TermDD       Terminal Device Driver  Kernel
Udfs         Udfs                    File System
Update       Microcode Update Drive  Kernel
```

Fig. 11. List of drivers in the Honeypot system that uses Sebek tools.

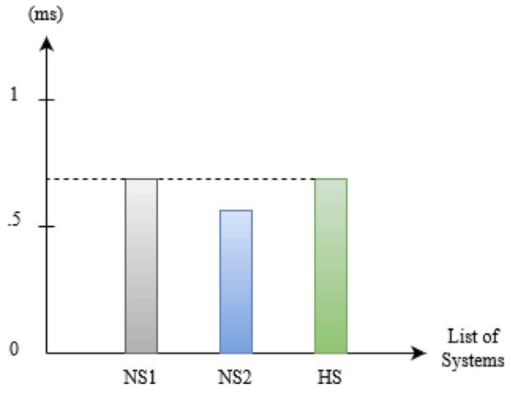


Fig. 12. Response time analysis of the considered network.

   - **Memory Usage**
   Sebek uses variables that must be stored in memory so that its effects can be found by searching in memory. It can be very helpful to know that the names of the Sebek Symbols start with a letter and are followed by a 3-digit number.
   Proposed method: The proposed concealment method does not use any variables stored in memory.
   - **Replacing system functions with a new version**
   Sebek uses the system function placement with its modified version causing a discrepancy in the stored memory, which makes it easily identifiable.
   Proposed method: We are able to view system changes using file system filter driver features without altering important system functions. These drivers work in a similar way to anti-viruses, so the intruder cannot detect the difference in the functionality of Honeypot and anti-viruses.

2. Response time

The most important parameter in detection of fake system is its response time, it is performed by comparing response time of all systems in the network. In this section, we analyze the response time of the Honeypot system with other systems in the network. Fig. 12 shows the result of measuring the response time of each of the systems in the case study. As seen in the figure, The response time of the honeypot system is exactly the same as the similar system in the network and is no different from the real system.

Table 7
Classification detail of the level of virtualization, memory usage and Honeypot's kernel modifications in Sebek, Qebek, VMScope, and proposed Honeypot.

Level	Virtualization	Memory usage	Kernel modification
Low	The monitoring tool is located in the kernel of the Honeypot system and the VMM was not used.	The monitoring operation has not left any trace of the monitoring in the memory of the Honeypot system.	The Monitoring operations are performed without manipulating system functions.
Medium	The main monitoring is executed in VMM but some operations in Honeypot's system are done.	The execution of some operations needs to Honeypot's memory usage	Some kernel functions are modified or hooking done on a kernel object
High	A virtual Honeypot is implemented and the monitoring tool is located entirely out of Honeypot's system and VMM used for monitoring	Honeypot memory is used to perform monitoring operations.	Some system functions are replaced for monitoring.

Table 8
Comparison of previous methods and proposed method.

Tool	Encryption	Data range	Honeypot's memory	Function replacement	Data transportation	Offline analyzing
Sebek	✓	high	×	×	×	✓
VMscope	✓	very high	✓	✓	✓	✓
Qebek	✓	high	✓	✓	✓	✓
Xen-based	✓	high	✓	×	✓	✓
proposed	✓	medium	✓	✓	✓	×

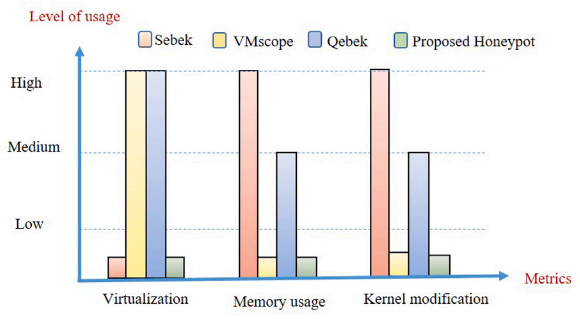


Fig. 13. Comparison of the existence of three destructive factors in Sebek,VMscope,Qebek, and proposed approach.

As mentioned earlier, kernel modification (Dornseif et al., 2004; Holz, 2005), memory usage (Wang et al., 2019), and the virtualization (Fu et al., 2006; Mukkamala et al., 2007), are three destructive factors in detection of Honeypot. The paper aims to reduce these three criteria so that Honeypot is resistant to existing detection methods. The Sebek tool was able to overcome an important problem facing network monitoring tools. Because it was able to access the data after decryption, and this is a big advantage. But it has drawbacks such as creating effects in the Honeypot's memory due to the use of Hooking to obtain system call data, not hiding the original driver in the Windows version so that it is clear in the list of loaded drivers. Instead, In VMScope and Qebek the monitoring tool has placed in VMM layer. Using a VMM has two major differences with a kernel-based monitoring tool Sebek. The first is that the report file is located outside the virtual machine and there is no need to transfer the report packets to another server. Second, it has access to the internal events of the system from the outside, which is more difficult to detect due to the isolation nature of the virtual machine described earlier. VMscope uses the binary translation technique for virtualization. This slows down the execution of VMscope, But because it does not change the kernel of the guest operating system, the monitoring tool does not leave a trace in the Honeypot's memory. This makes VMscope More stable than Sebek. The Qebek was created to improve the level of concealment of Sebek. This tool is also on the VMM and uses the binary translation method. Qebek hooks the implementation of system calls, So has some affects in Honeypot's kernel and memory. Fig. 13 shows a summary of the analysis performed in this case. In this analysis, we divided the level of virtualization, memory usage and Honeypot's kernel modifications into three categories: low, medium and high. Details of this classification are given in the Table 7. The results shown in Fig. 13, show that the proposed Hidden Honeypot could reduce all three parameters.

Table 8 shows the comparison of the proposed method with Sebek (Enemy, 2003), VMscope (Jiang and Wang, 2007), Qebek (Song et al., 2015), and Xen-based Honeypot presented in Kouba (2009). Because both of the proposed method and Sebek are kernel-based, they have access to the decrypted data in the Honeypot's kernel, and VMscope, Qebek, and Xen-based Tool have access to the events inside the kernel of the virtual machine due to the use of the VMM, and therefore they also access to the decrypted data. The scope of the data collected is not an objective in the proposed approach. The proposed approach only monitors the opening and creation events of the file system, but other solutions have more data scope. Sebek uses honeypot's memory and leaves traces in it, and this effect is a way to detection of it. The other approaches as well as the proposed approach have no effect on Honeypot's memory. And this has made the proposed hidden honeypot has been undetectable with the recognition ways provided through the effects of memory. In a Xen-based tool and Sebek substituting of system functions creates a suspicious honeypot's kernel and makes it possible to detect it. The proposed approach does not manipulate system functions. Sending of collected data to outside the honeypot is also one of the ways to detect it, in the proposed approach, the data are reported in real-time and are not sent outside the honeypot.

6. Conclusion and future work

The Honeypot detection problem by an attacker is one of the most important issues in the field of computer network security. Because if an attacker can detect a Honeypot on his target network, he can disable it. To solve this problem, a new solution is proposed to hide the kernel-based Honeypot. In this paper, for the first time, Honeypot was hidden using Rootkit hiding techniques. First, a data monitoring tool was created using the file system filter driver, and then it was hidden in the Honeypot's kernel using the DKOM hiding technique. The file system filtering driver allows monitoring of the events that occur in the kernel of the Honeypot from the kernel level, thus accessing the encrypted data of the attacker after they are decrypted at the kernel level. In this paper, the DKOM technique was implemented using the C programming language under the kernel level and applied on the file system filter driver in the Honeypot system. The performance of proposed approach was validated in a virtual network as a case study by simulating an attack.

No virtualization, no memory usage, and no kernel changes are the three main advantages of the proposed approach, which can make it resistant to Honeypot detection methods. In the proposed solution, we were able to hide the data monitoring tool inside the kernel of Honeypot system using the DKOM method used in modern rootkits, so that the monitoring work is done secretly inside the Honeypot without the VMM to be used. Table 8 Shows the result of this concealment. The VMM can be attacked similar to other systems on the network. Therefore, if it is used for data monitoring, the security of the host machine is also considered and measures must be taken to make it impenetrable to the host system, and this has its difficulties. In summary, the advantage of the proposed solution is that although it provides less information than the kernel-based data tool Sebek, on the contrary, it solves the kernel-based Honeypot detection problem and has almost all the features of VMM based monitoring tools.

In this paper, the main focus is on hiding the kernel-based Honeypot, so the expansion of this research in the future could be to increase the amount of data obtained from it. Also, offline analysis of data gathered by Honeypot is recommended for future researches.

CRediT authorship contribution statement

Maryam Mohammadzad: Writing – original draft, Methodology, Data curation, Investigation, Software, Validation. **Jaber Karimpour:** Conceptualization, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

Akshay, Akshat Divya, Bhushan, Anchit, Anand, Nihal, Khemka, Rishabh, K.A., Sumithra Devi, 2020. HONEYPOT: Intrusion detection system. Int. J. Educ. Sci. Technol. Eng. 3 (1), 13–18.

Almutairi, Abdulrazzaq, Parish, David, Phan, Raphael, 2012. Survey of high interaction honeypot tools: Merits and shortcomings. In: Proceedings of the 13th Annual PostGraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting. PGNet2012, PGNet.

Arnold, Thomas Martin, 2011. A Comparative Analysis of Rootkit Detection Techniques. University of Houston-Clear Lake.

Artail, Hassan, Safa, Haidar, Sraj, Malek, Kuwatly, Iyad, Al-Masri, Zaid, 2006. A hybrid honeypot framework for improving intrusion detection systems in protecting organizational networks. Comput. Secur. 25 (4), 274–288.

Christoph, Pohl, Meier, Michael, Hof, Hans-Joachim, 2015. Apate-a linux kernel module for high interaction honeypots. arXiv preprint arXiv:1507.03117.

Dornseif, Maximillian, Holz, Thorsten, Klein, Christian N., 2004. Nosebreak-attacking honeynets. In: Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004. IEEE, pp. 123–129.

Dowling, Seamus, Schukat, Michael, Barrett, Enda, 2018. Using reinforcement learning to conceal honeypot functionality. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, Cham, pp. 341–355.

Enemy, Know Your, 2003. Sebek, A kernel based data capture tool, The honeynet project.

Fu, Xinwen, Yu, Wei, Cheng, Dan, Tan, Xuejun, Streff, Kevin, Graham, Steve, 2006. On recognizing virtual honeypots and countermeasures. In: 2006 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing. IEEE, pp. 211–218.

Holz, Thorsten, 2005. Anti-honeypot technology.

Jayasinghe, K., Shane Culpepper, J., Bertok, Peter, 2014. Efficient and effective realtime prediction of drive-by download attacks. J. Netw. Comput. Appl. 38, 135–149.

Jiang, Xuxian, Wang, Xinyuan, 2007. Out-of-the-box monitoring of VM-based high-interaction honeypots. In: International Workshop on Recent Advances in Intrusion Detection. Springer, Berlin, Heidelberg, pp. 198–218.

Kapil, Divya, Mishra, Preeti, 2021. Virtual machine introspection in virtualization: A security perspective. In: 2021 Thirteenth International Conference on Contemporary Computing. IC3-2021, pp. 117–124.

Kouba, Tomáš, 2009. Virtual honeynet with simulated user activity.

Krishnaveni, S., Prabakaran, S., Sivamohan, S., 2018. A survey on honeypot and honeynet systems for intrusion detection in cloud environment. J. Comput. Theor. Nanosci. 15 (9–10), 2949–2953.

Lackner, Paul, 2021. How to mock a bear: Honeypot honeynet honeywall & honeytoken: A survey.

Li, Beibei, Xiao, Yue, Shi, Yaxin, Kong, Qinglei, Wu, Yuhao, Bao, Haiyong, 2020. Anti-honeypot enabled optimal attack strategy for industrial cyber-physical systems. IEEE Open J. Comput. Soc. 1, 250–261.

Liao, Hung-Jen, Lin, Chun-Hung Richard, Lin, Ying-Chih, Tung, Kuang-Yuan, 2013. Intrusion detection system: A comprehensive review. J. Netw. Comput. Appl. 36 (1), 16–24.

Lin, Zhi, Song, Yubo, Wang, Junbo, 2021. Detection of virtual machines based on thread scheduling. In: International Conference on Artificial Intelligence and Security. Springer, Cham, pp. 180–190.

Luckett, Patrick, Todd McDonald, J., Dawson, Joel, 2016. Neural network analysis of system call timing for rootkit detection. In: 2016 Cybersecurity Symposium. CYBERSEC, IEEE, pp. 1–6.

Miah, Mohammad Sujan, Gutierrez, Marcus, Veliz, Oscar, Thakoor, Omkar, Kiekintveld, Christopher, 2020. Concealing cyber-decoys using two-sided feature deception games. In: HICSS. pp. 1–10.

Miyamoto, Daisuke, Teramura, Satoru, Nakayama, Masaya, 2014. INTERCEPT: high-interaction server-type honeypot based on live migration. In: SimuTools. pp. 147–152.

Mukkamala, S., Yendrapalli, K., Basnet, R., Shankarapani, M.K., Sung, A.H., 2007. Detection of virtual environments and low interaction honeypots. In: 2007 IEEE SMC Information Assurance and Security Workshop. IEEE, pp. 92–98.

Mysliwietz, Egidius, Moonsamy, Veelasha, 2020. Identifying rootkit stealth strategies.

Nadim, Mohammad, Lee, Wonjun, Akopian, David, 2021. Characteristic features of the kernel-level rootkit for learning-based detection model training. Electron. Imaging 2021 (3), 31–34.

Pittman, M., Hoffpauir, Kyle, Markle, Nathan, Meadows, Cameron, 2020. A taxonomy for dynamic honeypot measures of effectiveness. arXiv preprint arXiv:2005.12969.

Qi, Duan, Al-Shaer, Ehab, Islam, Mazharul, 2019. CONCEAL: a strategy composition for resilient cyber deception: framework, metrics, and deployment. In: Autonomous Cyber Deception. Springer, Cham, pp. 101–124.

Rowe, Neil C., 2019. Honeypot Deception Tactics in Autonomous Cyber Deception. Springer, Cham, pp. 35–45.

Shi, Leyi, Jiang, Lanlan, Liu, Deli, Han, Xu, 2012. Mimicry honeypot: a brief introduction. In: 2012 8th International Conference on Wireless Communications, Networking and Mobile Computing. IEEE, pp. 1–4.

Singh, Baljit, Evtyushkin, Dmitry, Elwell, Jesse, Riley, Ryan, Cervesato, Iliano, 2017. On the detection of kernel-level rootkits using hardware performance counters. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. pp. 483–493.

Sohal, Amandeep Singh, Sandhu, Rajinder, Sood, Sandeep K., Chang, Victor, 2018. A cybersecurity framework to identify malicious edge device in fog computing and cloud-of-things environments. Comput. Secur. 74, 340–354.

Song, C., Ha, B., Zhuge, J., 2015. Know your tools: Qebek–conceal the monitoring—The honeynet project.

Su, Ming-Yang, 2010. Discovery and prevention of attack episodes by frequent episodes mining and finite state machines. J. Netw. Comput. Appl. 33 (2), 156–167.

Suratkar, Shraddha, Shah, Kunjal, Sood, Aditya, Loya, Anay, Bisure, Dhaval, Patil, Umesh, Kazi, Faruk, 2021. An adaptive honeypot using Q-learning with severity analyzer. J. Ambient Intell. Humaniz. Comput. 1–12.

Tsaur, Woei-Jiunn, Chen, Yuh-Chen, 2010. Exploring rootkit detectors' vulnerabilities using a new windows hidden driver based rootkit. In: 2010 IEEE Second International Conference on Social Computing. IEEE, pp. 842–848.

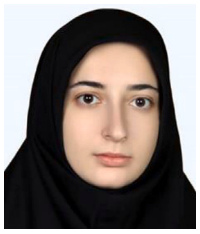
Tsikerdekis, Michail, Zeadally, Sherali, Schlesener, Amy, Sklavos, Nicolas, 2018. Approaches for preventing honeypot detection and compromise. In: 2018 Global Information Infrastructure and Networking Symposium. GIIS, IEEE, pp. 1–6.

Uitto, Joni, Rauti, Sampsa, Laurén, Samuel, Leppänen, Ville, 2017. A survey on anti-honeypot and anti-introspection methods. In: World Conference on Information Systems and Technologies. Springer, Cham, pp. 125–134.

Wang, Ping, Wu, Lei, Cunningham, Ryan, Zou, Cliff C., 2010. Honeypot detection in advanced botnet attacks. Int. J. Inf. Comput. Secur. 4 (1), 30–51.

Wang, Xiao, Zhang, Jianbiao, Zhang, Ai, Ren, Jinchang, 2019. TKRD: Trusted kernel rootkit detection for cybersecurity of VMs based on machine learning and memory forensic analysis. Math. Biosci. Eng. 16 (4), 2650–2667.

Xu, Zhixing, Ray, Sayak, Subramanyan, Pramod, Malik, Sharad, 2017. Malware detection using machine learning based analysis of virtual memory access patterns. In: Design, Automation & Test in Europe Conference & Exhibition. DATE 2017, IEEE, pp. 169–174.



Maryam Mohammadzad received the B.Sc. and M.Sc. degrees in Computer Science in 2015 and 2017 (Iran University of Tabriz). Now she is Ph.D. student in the CS Department at Tabriz University. Her research interests focuses in computer network and security. He has been member of talent students group in master degree of education.

Jaber Karimpour received the B.Sc. Degree in Computer Science and Applied Mathematics from the University of Tabriz (Iran) in 1998, the M.Sc. Degree specializing in the computer systems are of Applied Mathematics from the University of Tabriz in 2000, and the Ph.D. degree in Computer Systems from the University of Tabriz. He is currently an Associate Professor in the Department of Computer Sciences at the University of Tabriz and has been the manager of Information Technology of the university since 2011. His current research interests include cryptography, network security, formal specification and verification of computer systems.