Cyber Trap System Using Honeypots

Bhumish M Gajjar
Gujarat Technological University PG School, Ahmedabad, India
In collaboration with
Centre for Development of Advanced Computing, Pune, India
Email: bhumish.gajjar@gmail.com

Abstract—The domain of cyber security has become a concerning area for all types of industries which are connected to the Internet. There has been security solutions provided, which use Intrusion Detection Systems, Firewalls, or Honeypots to safeguard the network perimeter against attacks. The proposed structure is a combined security solution which uses Honeypots and IDS systems to provide security. The system is not resource intensive as it does not have initially deployed honeypots but creates them on-demand during attacks. The system acts as a trap for the cyber-attackers and malwares, as the attackers are directed to the virtual resources and malwares are captured, everything managed by the honeypots. The system is not meant to interact with normal users, but if there is an attack or binaries with malicious behavior, it becomes a trap for them. The main purpose of this proposed structure is to have a well structured and enhanced security solution through honeypots to redirect the attacker to virtually created honeypots while keeping the attacker busy and to deal with malwares being spread in the system. The solution deals with the cyber attacks and malwares by either redirecting them to the on-demand honeypots or capturing and dissecting the malwares according to their type.

Keywords—Intrusion Detection Systems, honeypots, malware detection, risk mitigation

I. INTRODUCTION

The corporate infrastructure in today’s world is dependent on the Internet for their business and communications. The dark side of the Internet is getting profit in this, by attacking soft target corporate that are unaware of cyber security. The corporate deploy basic security devices and get satisfied that their network is secure. But these devices are easy to bypass, and less amusing for the attackers. The attackers want more amusement by interacting and fiddling with more data and striking the corporate with more power. According to a recent report by Sophos [1], 96% of businesses are unprepared for a cyber attack.

The current state of honeypot devices is that they are used for either malware capture – where they are kept aside in a network to monitor the network; or to fake some critical resources – to amuse and track the attackers. In the paper, a unified and autonomous system is given which used honeypots and IDS for intrusion detection, resource management by creating on-demand honeypots and malware capture. Intrusion detection is performed at the entry level for detecting any network intrusion by using an IDS, and their logs are parsed for any severe alerts. Whenever there is an attack on the resource, the system creates a honeypot which is the replica of the victim resource by fetching the details of all the services and the attacker is redirected to the honeypot. In this manner, the resources stay safe and the attacker gets some amusement while playing with the honeypot. Malware capture is done on the basis of a toolkit which catches malwares by three different techniques.

II. SYSTEM OVERVIEW

Honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource, as defined by Lance Spitzner [2]. The proposed system is capable of detecting and mitigating network attacks on significant resources as well is continuously monitor to capture any malwares being spread in the network. The unified structure consists of various modules to automate the system monitoring after learning the basics through human interaction.

During the initial stages, the Malware Detector module learns the signatures and behaviours of malwares. Through human interaction, the module is taught the executables and binaries which harm the systems. The packet contents can be checked for any malicious match according to a research by Ahmed et al.[3], to check the packet contents later for any malwares.

The Intrusion Detector module is located at the gateway and watches for any intrusion attempts. Whenever any attack is detected, it generates an alert along with asking other modules to create a Honeypot which is a replica of the victim resource. All the services which were running on the victim machine are faked on the Honeypot, and the attacker is redirected here. The redirection is camouflaged to be a load balancing or other kind of redirection, so that the attacker does not sniff that he is being redirected. This provides a sandbox, which does not harm the functioning of the original resources on the network – along with providing an automated honeypot management system.

The system follows an algorithm for functioning. The algorithm takes reference from Honeydoop system as proposed by Kulkarni et al. [4] for creating on-demand honeypots whenever an attack is detected. In addition, automated backup and restoration for harmed systems is done.
The malware detection function runs continuously to monitor for any malware present in the network. The malware detection again follows the Honeybow toolkit as used by Zhuge et al. [5] for detecting and capturing malwares through different techniques.

III. ALGORITHM

The following algorithm is based on the Honeydoop [4] system, which creates on-demand honeypots whenever there is an attack detected. Along with the Honeydoop system, the algorithm powers the malware detection engine and the backup-and-restore procedures for better functionality.

```java
{  
  while (true):
    mw_detector()  #continuous malware detection
    x = detect_intrusion(i)  #IDS log parsing
    if x == 1:
      #detected intrusion
      create_honeypot(get_system_data())
      backup_original()
      backup_honeypot()  
      reconfigure_network()
    end if
    honeypot_fetch()  #get data from honeypot
    if sys == files_added():
      #sys = clean honeypot system
      mw_check()  #check for malware
      mw_analysis()
      mw_removal()
      if sys == files_added():
        restore_honeypot()
      manage_network()
    x = auditing_needed(j)
    if x == 1:
      start_auditor()  #log parsing
    end if
  end while
}
```

where i is a variable whose value is 0 if there is no intrusion, and 1 if there is any intrusion detected, j is a variable whose value is 0 if the auditing is not needed, and 1 if the network has been compromised and there is need of auditing.

From the given algorithm, the below mentioned states for the system are possible:

The Input States for the system can include:
- A network consisting of all the network devices managed by the system
- A network which is about to be compromised by an external attack

The Output States for the system can include:
- All attempts of intrusion are detected by the system, logged and proper actions are taken to mitigate the risk from the intrusion
- There are no attempts of intrusion and hence the system is safe
- The system is harmed by some attack which has gone undetected

The Success States for the system can include:
- The system is functioning and there are no intrusion attacks
- All the attempts of intrusion are logged, proper actions are taken by the modules to defend against them and the system is functioning without any harm
- None of the resources on the network are pre-compromised

The Failure States for the system can include:
- Malfunctioning of a system module
- The devices are pre-compromised, there can be no effect of the system
- The attacker is able to break the honeypot system and reaches the original resources, proving fatal

IV. SYSTEM FUNCTIONS

`mw_detector()`:
This function is provides the detection of any malware being spread in the network. It is dependent on three sub-functions, which are continuously monitoring the network for any malwares. The three sub-functions work as below:
- keep a watch at the entry point for malicious behavior
- periodically check the honeypots for any file modification
- extract the binaries entering into the network

`detect_intrusion()`:
This function will detect any intrusion attempts by parsing the logs produced by the Intrusion Detection System. If there are any attempts detected, it will generate an alert and further actions will be taken by the administrator and the honeypot manager module.

`update_system_data()`:
The logs created by the machines in the whole system are periodically updated and sent to the database to detect any anomalies and for storing the significant events.

`get_system_data()`:
This function fetches the data for running services of a victim resource from the logs and systems whenever there is any intrusion attempt for providing it to the next function to create a honeypot with the same services.

`create_honeypot()`:
Whenever there is any intrusion detected by the intrusion detector, this function is notified and asked to create a honeypot which is the replica of the victim resource - by fetching the system information from `get_system_data()` function.
backup_original():
During any attacks on a resource, it is backed up for security reasons. There is a periodical back-up of each resource, but for getting updated information about the attack and the traces of the attacker, the resource is backed up.

backup_honeypot():
The honeypots are played with and modified by the attacker, and hence they are backed up for getting any traces or evidence of the attack and attacker. The backup copy of the honeypot can be used next time for present it to a new attacker, which is a modified version of the honeypot and hence the new attacker is satisfied of getting something interesting.

reconfigure_network():
During any attacks, the resource is replicated by a honeypot and the attacker is redirected to the newly created honeypot. That part of the network needs to be reconfigured to redirect the attacker to the honeypot. The attacker does not need to know that it was a redirect to a trap, and hence it is shown as if a redirect took place for load balancing.

honeypot_fetch():
This function fetches data from the honeypot during an ongoing attack so that the attacker can be traced as easily and rapidly as possible.

files_added():
This function matches the honeypot with a previous version of the honeypot with the current version, and checks for any files modified by the attacker. The modified files are considered for later analysis.

mw_check():
The modified files are checked for any malicious content by matching them with available hashes of malwares and trojans.

mw_removal():
The modified files from the honeypot are then removed which show malicious content, by noting their hashes so as they can be matched with incoming malwares in future.

restore_honeypot():
This function restores the honeypot in case there is a recent attack. When the attacker is provided a dirty copy of the honeypot, they are happy to get something interesting out of it, and hence do not have suspicions of being landed to a honeypot.

manage_network():
The network is restored to the original after an attack has taken place and the system is functioning without any kind of harm. The unnecessary honeypots are removed from the systems and redirects are reconfigured to the original.

auditing_needed():
If the attack has left any traces of the attacker, these traces can be matched with logs from previous records and logs from the same day on other resources.

start_auditor():
The auditor module analyzes the honeypot logs, while extracting information like attacker’s location, exploits used, timestamps for providing a graph of the obtained details.

V. DESIGN

The design of the system can be divided into five modules, namely Malware Detector, Intrusion Detector, Honeypot Manager, Auditor and Backup Manager. For attracting the attackers to exploit our honeypots, they can be made to have high ranking in search and more requested servers which lure the attackers as researched by John et al. [6]. The whole network is continuously monitored with the Malware Detector module, and it is the module which is always running to check for any malwares. The Intrusion Detector is located at the entry point of the system and checks for any incoming threats. The other modules come into picture only when called for, the Honeypot Manager creates and manages a Honeypot whenever there is a need for having a honeypot to deal with an attacker, the Auditor checks the system for any malicious activities and for generated logs after significant events and the Backup Manager takes backup of the resources and honeypots whenever necessary.

Fig. 1: Overview of the Malware Detection Engine. The Watcher module constantly watches for any malicious activities, whereas the Hunter module looks for any Windows binaries. The Fetcher module periodically restarts the system and catches the modified files.

A. Malware Detector

The Malware Detection module, which works in collaboration with Nepenthes honeypot, runs independently on the network to detect and capture any malwares in the network. The module contains of various functions which search for malwares in different ways and later submit them for further research.

The research work by Halani [8] for capturing malwares by the Nepenthes honeypot is one of the pillars of the malware capture system. The Malware Detection is based on the
Honeybow toolkit as used by Zhuge et al. [5]. There are three different functions for catching malwares in the network - the fetcher, watcher and hunter. The MwFetcher is a service which does cross-matching to catch any malicious files. It is given a clean system for reference, and it matches the honeypots with the base clean system - and as all the traffic coming to honeypot and the files modified in the honeypot are malicious, it categorizes the added and modified files as malicious. The next service named MwWatcher watches the network for any malicious file to capture. The third service named MwHunter is a preprocessor which extracts Windows binaries incoming to the network and checks them for malicious behavior.

There are other functions namely Submitter and Checker which matches the hash of obtained malwares with known malwares, and verifies the identity of the malware. If the malware is not listed with the known malwares, but still does something malicious, its hash is submitted to the Antivirus engines through their APIs. The Collector function keeps all the malware hashes recorded for matching.

**B. Intrusion Detector**

The Intrusion Detector module is a pre-processor which controls the calling of other modules. Whenever there is any intrusion attempt in the network, the Intrusion Detector module checks it and shoots out any alert if necessary while calling other modules.

The Intrusion Detector module is basically dependent on Snort IDS, which is a tool for monitoring the network to detect any intrusion attempts. Snort is deployed at the entry point – gateway of the network and relevant logs are generated. The Intrusion Detector module contains a function for log parsing, which continuously watches the log for any intrusion attempts. The module generates an alert to inform the administrator if there is any serious intrusion attempt, e.g. brute-force attack to log-in to the FTP server.

The module keeps an eye for intrusion attempts, and when it catches any significant intrusion to harm the systems, it connects with the Honeypot Manager module to create a Honeypot and deal further with the attacker. The module watches for any intrusion attempt through the defined rules from Snort IDS, as well as any SQL Injection attempts through the URL parser function. The threshold value is set for SQL Injection attempts, and any attempt beyond the value will redirect the attacker to the honeypot.

**C. Honeypot Manager**

The Honeypot Manager is the backbone of the system, which provides the creation and management of the honeypots to mitigate the threat of the attack. The initial function of the module is to create honeypots whenever there is an attack detected, by fetching the data from the Intrusion detector module.

The logs of the honeypot, which record details like timestamps, IP address, type of request, username, are parsed and relevant information are recorded in the repository and parsed to extract important details for providing to the administrator. The module checks for the services running on the victim machine and tries to replicate all the services. It provides a sandbox, so that the attacker can play inside the honeypot without disturbing the original resources.

Through parsing the logs, it obtains the geo-location and ISP details of the attacker through web APIs. If a single proxy is used, it can detect the source IP address. By checking what was affected to the website (e.g., SQL Injection error messages), the type of the attacker can be identified. There has been a recent research by Sintsov [7] which proves the usefulness of honeypots for getting extensive information of the attacker by planting a backdoor to the attacker machine. This functionality is yet to be implemented in the proposed structure, which reveals the skills of the attacker as well as provides useful information from the attacker’s machine.

**D. Auditor**

The Auditor module functions to collect all data persisting in the system regarding the network, to check for any intrusion or network changes and to log every significant data to the repository. All the data collected by the auditor is logged to the repository with respective timestamps and details of the relevant devices.

The Auditor collects data like open ports, operating systems on the resources, hosted services, number of connections to the service, which are necessary to be provided to the Honeypot Manager for creating a replica of the resource being attacked. The module logs all the incoming connections to the various resources, but for the sake of saving space and resources, it overwrites the non-significant logs after a period of 10 days. However, the logs for attacks and other malicious activities are saved for a longer period for future reference.
The captured malwares and their respective hashes are also logged by the Auditor module.

All the data is logged to a secure repository which is backed-up regularly for better safety. The log repository cannot be reached by any other module or devices except the Auditor module; hence there are minimum chances of the repository being attacked.

E. Backup Manager

The function of the Backup Manager module is to provide the replicas of resources as well as honeypots and to manage the backup and restore process. This module takes regular backups of the systems and honeypots, to provide restoration facility in case of any uncalled events.

In case of attacks, the Honeypot Manager creates a honeypot which is the replica of an original resource. The Backup Manager provides a copy of the Honeypot which was modified previously by some other attacker. When instead of getting a clean Honeypot if an attacker gets a resource with some interesting data, he may want to play with it more and leave traces of the attack, without getting the real scenario that he is playing with a honeypot.

The Honeypot Manager contains the backup copies of each resource. Whenever a resource is attacked and modified by an attacker, the module matches the resource with a prior clean copy of the resource. All the modifications made to the resource can be matched and tracked. This provides a solution to restore a working copy of the resource and easily check for the traces of attacks on the resource.

VI. PROCESS OVERVIEW AND OUTCOME

The overall process takes place as described: Snort [9] IDS is located at the entry point and actively looking for any network attacks. If there are any attacks found, a honeypot will be created with the help of Honeyd [10], which is the replica of the original resource being attacked. All the processes, services, open ports on the victim resource are replicated by the newly created honeypot. The attacker is redirected to the honeypot, leaving the system safe. The changes made to the system are cleaned by the restore procedure. Along with the network attack redirection, the honeypot for malware capture is continuously monitoring the network perimeter for any malwares, by matching the binaries or any malicious behaviors.

The initial deployment of the system has proved to be an useful solution for mitigating the risk of network attacks and malwares. A malware binary created for testing, which acted as a backdoor for entering into a victim machine was captured by the honeypot capture service. The system was successful to deal with the Denial of Service attack on the Web-Server and port-scanning on the Database server, and redirect the attacker machine to a newly created honeypots.

VII. CONCLUSION

There are solutions available which work independently to detect attacks or capture malwares by utilizing Honeypots. The proposed solution combines these different aspects into a combined structure thus enhancing the security provided through honeypots. There have been honeypots deployed to individually do these mentioned security tasks, but a combined and automated solution has not been proposed.

The proposed solution is a combined structure, which can be complex during the initial stage of deployment. But as it is an automated system, it does not require more user interaction after its configuration. At the initial stages, during the learning phase it needs to have human interaction – but after that stage it works automatically on the basis of signatures and anomalies it has learnt. It provides a centralized management console and thus the management of the modules and resources is simplified. For organizations with large number of significant resources, this solution can prove ideal if deployed and managed in responsible manner.

Considering the scope of the unified system, future work can include a system with faster reaction to the attacks, with light weight honeypots and extensive information gathering is desired from the honeypots. Firewalls can be combined with the system for enhancement of security, filtering the malicious traffic. As there are advancements in the capabilities of Honeypots, the algorithm and the functionalities can be modified accordingly to provide an updated solution.

ACKNOWLEDGMENT

The author thanks Mr. Girish Khilari for providing their support and guidance during the research work. The author is also thankful to the Centre for Development of Advanced Computing (CDAC) Research and Development Department, where most of the proceedings of the research have been performed using the infrastructure.

REFERENCES