POLITECNICO DI TORINO

Master Degree in Communications and Computer Netowrks Engineering

Master Degree Thesis

From Honeypots to Distributed Deception Platforms

Theory and testing of emerging technologies for IT security.

Supervisors
prof. Antonio Lioy

Candidates
Vincenzo Viola

Internship Tutor
Intesa Sanpaolo S.p.A.
Ing. Davide Grangia

Academic Year 2018-2019

Contents

Abstract

The following thesis presents a research on the evolution of the deceptive technologies adopted in cyber-defense. In the first section honeypots are analysed in deep, evaluating their strengths and weaknesses, providing also some intriguing use-cases. The second part of the thesis focuses on the Distributed Deception Platforms (DDP), a brand-new paradigm which has the purpose to overcome honeypot limits and to allow an easier deployment of deceptive tools over the defender's network. Some of the most important distributed deception platforms present on the market are tested along with open-source solutions: the test involves the simulation of an Advanced Persistent Threat (APT) to be used against the platforms under analysis in order to evaluate their efficacy, the depth of information that it is possible to retrieve from an attacking pattern and the ease of deployment and management of the rising solutions.

Acknowledgements

I wish to thank my family that supported me to follow my ambitions and all my friends and colleagues that took part on this enjoyable journey.
The dedication of this thesis goes to my grandfather Vincenzo, gentle soul and silent teacher of the most precious of lessons.

Chapter 1

Introduction

1.1 What is Deception?

Despite the fact that new cybersecurity technologies are constantly introduced or improved, data breaches are events that appear far from cessation. Verizon 2019 Data Breach Investigation Report [1] states that from a sample of 41686 security incidents evaluated, 2013 were confirmed as data breaches: 69% of all the data breaches were conducted by outsiders, however the role of insiders cannot be understimated since 34% of the data breaches involved internal actors (including unintentional events). Cost of a Data Breach Report [2] analyzes how data breaches affect the economy of a company even years after the breach occurred, with an average loss of 1.42 Million $; the cost derived from data breaches depends also on the event that caused it: breaches caused by intentional attacks cost 27% more than breaches due to human error and 37 % more than system glitches, mainly because attacks can require more time to be identified.
In such scenario, it seems reasonable to look for a different paradigm in which it is assumed that the breach already occurred in the system, and the goal of the security team is now to delay the attack operation, to understand the tools and techniques that are being used, to deceive the attacker and to push the threat into safe areas: this approach is generally known as **Cyber Denial & Deception** (D&D). By such definition, deception's goals are two-fold: to provide to the attacker false or misleading information, capable to give him the belief and the confidence that the attack is ready to be performed and the situation is under attacker's control (Deception); to create uncertainty about the reality of the environment that the attacker is facing, in order to slow down the attacking operations or to lead the attacker to waste time or resources (Denial) [3]. In order to better understand the practical differences between denial and deception, it can be useful to introduce the distinction made by *Cyber Denial, Deception and Counter-Deception* [4] about the information (facts and fictions) to show and to hide to the attacker, dividing it in four categories:

- *Non-Essential Friendly Information* (NEFI), the real facts that don't require to be hidden from the attacker;
- *Essential Elements of Friendly Information* (EEFI), the facts that need to be

hidden from the attacker;

- *Essential Elements of Deception Information* (EEDI), fake information that needs to be presented to the attacker;
- *Non-Disclosable Deception Information* (NDDI), information that reveals the deception plan and therefore must remain hidden from the attacker.

Denial's scope is to **hide the EEFI**, while deception must be applied in order to **highlight the EEDI**.
D&D as a strategy did not originate in the cyber domain, but has its roots in the military field; one of the five foundations of *US Information Operations* known as *MILDEC* (Military Deception) enunciates six principles in order to plan and execute deception operations [5]:

- **Focus:** *The deception must target the adversary decision maker capable of causing the desired action(s) or inaction(s)*;
- **Objective:** *To cause an adversary to take (or not to take) specific actions, not just to believe certain things*;
- **Centralized planning and control:** *Military deception operations should be centrally planned and directed*;
- **Security:** *Deny knowledge of a force's intent to deceive and the execution of that intent to adversaries*;
- **Timeliness:** *A deception operations requires careful timing*;
- **Integration:** *Fully integrate each military deception with the operation that it is supporting.*

Those principles are revised in the cyber ground by the definition of the so-called **Deception Chain**, which is composed by three main phases: *planning*, *preparation* and *execution* [4]. The planning phase includes the purpose of the deception (i.e the objective), the intelligence gathering (i.e. the focus, how the opponent should react to deception) and the design of the deception cover story; the preparation phase consists in the exploration of the available resources to make deception possible and in the cooperation with other operators as network managers (i.e. integration principle); finally, in the execution phase the deception operation is monitored and eventually reinforced if the results are not as good as expected.

1.2 Why Deception?

As explained in the previous paragraph, Cyber Denial & Deception's aim is to provide another level of security after that the attacker has already been able to breach inside the system (or in the case of insiders); it could be observed that there are already components such Intrusion Detection Systems (IDS) that have

the purpose to detect suspicious activities inside the network, but those systems can hardly recognize previously unknown attacks, or malicious behavior hidden in actions which IDS can consider legitimate. The class of attacks for which D&D is called to take remediation is also known as **Advanced Persistent Threats (APT)**. Colin Tankard in its paper *Advanced Persistent threats and how to monitor and deter them* [6] defines APTs as "*a new breed of insidious threats that use multiple attack techniques and vectors and that are conducted by stealth to avoid detection so that hackers can retain control over target systems unnoticed for long periods of time*". The term Advanced may refer to the exploit of zero-day vulnerabilities or, more in general, to the use of sophisticated tools, tactics, techniques and procedures (TTTPs); the term Persistent refers to the (usually long) period of time under which the attacker conducts its malicious behavior inside the defender's system.
Another concept clarifies better the distinction between classic attack patterns and APTs: the report *M-Trends: advanced persistent threat* from 2010 by *Mandiant* [7] observes a recognisable, repetitive cycle which characterizes advanced persistent threats (it can be observed that the following phases are similar to the **Cyber Kill Chain** steps [8] defined by *Lockheed-Martin* corporation in 2011) :

1. Reconnaissance: the attacker identifies individuals of interest and develop methods of potential access to the target;

2. Initial intrusion: it may be achieved through social engineering techniques such as *Spear Phishing* against the target identified in the previous step;

3. Backdoor instalment: the attacker attempts privilege escalation, then installs multiple backdoors through lateral movements;

4. Obtain User credentials: the attacker targets domain controllers to obtain a large set of valid user credentials;

5. Install utilities: the user credentials previously obtained are now used to install utilities needed by the attacker (dump password, network monitoring);

6. Data exfiltration: the attacker steals data from the compromised network, encrypting and compressing it, then sending it to "staging servers" within the APT's command and control infrastructure in which stolen data is aggregated;

7. Persistence: as the victim detects intrusion, the attacker repeats the previous steps in order to maintain its presence in the compromised network.

Along with the aforementioned Cyber Kill Chain, it is worth to mention the MITRE ATT&CK Matrix [9] which provides a detailed taxonomy about APTs actions that can be conducted in all the cyber kill chain phases, and such matrix will be the ground floor in this thesis to simulate an APT to be tested against the deception solutions.

1.3 Objectives and Contributions

In the previous paragraphs were introduced the main actors involved in this research: indeed, this thesis will analyse the evolution of the main deception technologies adopted in the cyber scenario over the years and to test their efficacy, which will be evaluated basing on the depth of information that they are able to retrieve in an attacking scenario; in particular, the focus is on the emerging distributed deception platforms to be compared against traditional honeypots. The purpose of this thesis is to understand the value of distributed deception platforms to recognise Advanced Persistent Threats in comparison with the results obtained with traditional honeypot technologies, to evaluate their complexity in the deployment and management stages. Opensource platforms will be tested alongside proprietary solutions, which will be analysed in partnership with Intesa Sanpaolo which will perform a Proof of Concept of the adopted solutions.

Chapter 2

State of the art

Summary

This chapter will be dedicated to the chronological evolution of the main deception technologies; then the two principal approaches, honeypots and distributed deception, will be studied and evaluated.

2.1 Cyber Deception Background

2.1.1 Honeypots

One of the first relevant documents regarding a scientific deception approach applied to information systems is the paper from 1992 named *"An Evening with Berferd"* by Bill Cheswick [10]. The AT&T researcher added a set of fake services in its workstation, such as FTP, Telnet, SMTP DEBUG, and few others, properly configured in order to pretend that the services were actually running and the flaws they were attempting to exploit were present; at first, Cheswick monitored the cracker's attempts and decided in real-time which action to perform, but observing that the attacker was persistent in time, decided to develop a system called **The Jail**. It consisted in a *chroot* environment (the processes and their children are executed in a sub-directory different from the root directory, leading those processes to have restricted permissions) in which the processes with an higher risk of exploitation were taken out. Even if the attacker fell into the trap and did not appear to recognize it, Cheswick concluded that the Jail was a system too complicated to build and too dangerous to be worth it.
In November 1997, Fred Cohen developed one of the first honeypots available for free, the **Deception ToolKit** [11]. Just as the rudimentary Jail built by Cheswick, it emulates a system with several well known vulnerabilities, collects logs for every interaction and produces output response to attackers' inputs with the goal to waste their time and resources.
After DTK, new honeypot solutions arose in the market, but a true evolution of the technology was brought by the development of the honeynets.

2.1.2 Honeynets

This new architecture (figure 2.1) was introduced in 2005 by the paper *Know Your Enemy: Honeynets* from the HoneyNet Project [12]. The main concept about honeynets is to create a new network composed exclusively by honeypots: since there are no production servers inside the honeynet, every inbound or outbound connection which involves components within the network has to be considered as an action performed by an intruder. Furthermore, the honeynet is an example of **High Interaction** decoy, since it is composed by real systems providing real services and applications, while Cheswick's *Jail* and DTK are considered **Medium Interaction** decoys since they provide emulation of a real environment; the last category is composed by the **Low Interaction** decoys, which are the safest honeypots since they offer only emulated services with very little space for the attacker to exploit them, but on the other hand they provide little detail about the experienced attack. As mentioned before, honeynets define not a single device or product, but an architecture: it is crucial to separate the honeynet from the real environment, which is performed through a *honeywall*, a layer-2 bridging device (which allows it not to have a MAC address and packets to route so it is very difficult for the attacker to detect it) that has to collect traffic data but most of all has to perform data control (e.g. limit the bandwidth, the number of outbound connections) in order to limit attacker's actions. It has to be pointed out that the use of real machines makes the honeynet more believable but, on the other hand, it requires plenty of attention in the configuration of the honeywall since the attackers have now more potential to do harm if they manage to take control of the honeypots.

2.1.3 Tripwire

A further step towards recent distributed deception approach was made by a smart use of integrity check programs as *Tripwire*, which gives the possibility to monitor file systems and to check if they are modified, deleted, moved or added. The integrity check works by the use of two inputs: a *configuration* file in which the files and directories that need to be monitored are indicated, and a *database* file which is generated by Tripwire and contains the filenames, signature information and the configuration file that generated it; to check if modifications to files listed in the configuration file have been made, the program generates another database file and compares it with the already present database.
Obviously, it cannot be considered a stand-alone deception product, but specific use-cases of Tripwire anticipated the use of lures and baits in Distributed Deception Platforms (which will be analysed in deep afterwards). In the paper *"Experiences with Tripwire: using integrity checkers for intrusion detection"* [13], Kim and Spafford suggested a possible use of Tripwire by monitoring false sensitive data specifically created in order to catch stealth intrusions; this kind of data is generally known as **honeytokens**. [14] The two researchers also point-out that the first advantage given by the use of Tripwire is that it "cannot be turned against a system to identify or exploit weaknesses or flaws". This observation needs some clarification, since it is true that the monitoring operation conducted by Tripwire cannot be exploited, but on the other hand the use of monitored bait files need to

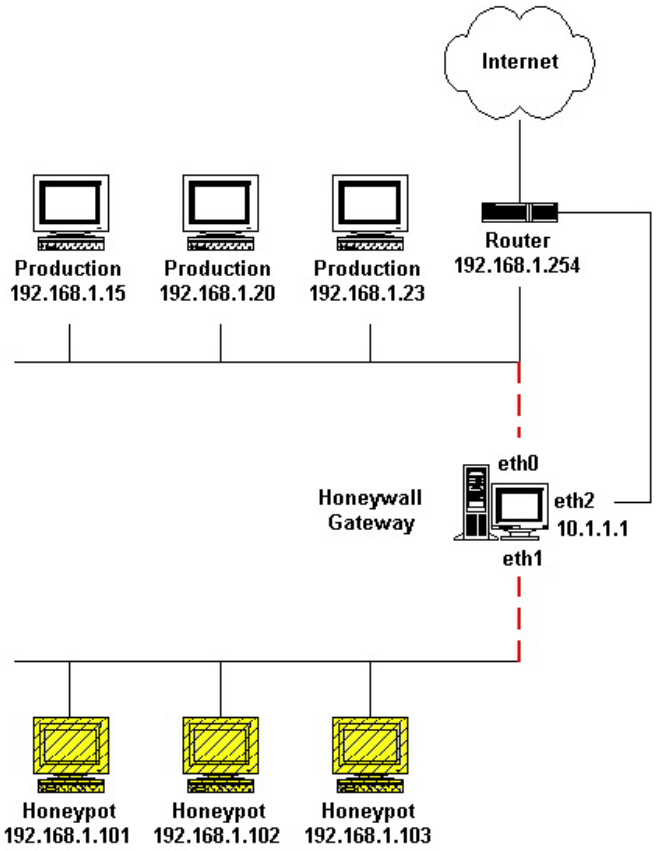


Figure 2.1: Honeynet architecture. (source:[12])

be carefully designed considering that an eventual leak of fake files could create as damage as the disclosure of the real ones.

2.1.4 Introduction to Distributed Deception Platforms

In recent years a new kind of products arose in the market which inherited a lot of the features from the previously discussed tools: indeed, distributed deception platforms (DDP) are systems which allow the distribution, configuration and monitoring of the deception elements, which fundamentally are **decoys**, **breadcrumbs** and **honeytokens**. The term decoy refers to honeypots, which can be high, medium or low interaction with the already discussed differentiation in section 2.1.2; honeytokens have already been introduced in section 2.1.3; breadcrumbs are data such user credentials which point to a particular decoy, and their scope is to tempt the attacker to interact with such decoy.

2.2 From Honeypots to Distributed Deception

In the previous section the chronological steps that led to the development of the distributed deception platforms starting from the very first honeypot prototypes have been briefly analyzed. Now the two paradigms, honeypots and distributed deception, will be deepened in order to better understand their strength and weaknesses, similarities and differences.

2.2.1 Honeypots review

Classification

The decoys introduced in the previous paragraph are well-known examples of honeypots, but it is quite intuitive that there are several variants of such technology depending on the environment in which it has to be deployed: considering that this thesis focuses on gaining more knowledge as possible about the simulation of an APT, it is necessary to introduce the following distinction taken from the book *Honeypots: Tracking Hackers* [15], which subdivides honeypots in two categories: **production** honeypots and **research** honeypots.
Production honeypots are honeypots which have the goal to enhance the defensive capacity of the system, mainly by detecting the threat and being able to take the proper counter-measures. Research honeypots instead have the goal to gain knowledge about the tools, techniques used by attackers and which are the motives that drive them. In other words, the goal of production honeypots is to defend the system, and they need to be easy to implement and maintain; the goal of research honeypots is to gain knowledge about threats, but they are not particularly adapt to defend the system (honeynets represent high interaction research-honeypots).
As mentioned before, honeypots can also be classified in high, medium and low interaction honeypots depending on the depth of information that they are capable to retrieve and on the degree of freedom that they leave to the attacker.
From the early projects of the 90's seen in the previous paragraph, honeypot evolution took many directions depending on the systems it needed to protect and the attacks it had to face; are now reported some of the most interesting honeypot deployments in the recent years.

Use-Cases

Up until now, honeypots have been presented as virtual or real machines to put inside the network waiting to be probed; the research *Detecting targeted attacks using shadow honeypots* from 2005 [16] proposed the use of a particular architecture, called **shadow honeypot**, to detect malicious traffic both at client (in the scenario in which the client has been convinced to download malicious data) and at server sides: the architecture is composed by a filter, anomaly detection sensors and the honeypot. The filter blocks the well-known attacks which don't need any further investigation; anomaly detectors can include different components which have to decide if the traffic can be considered acceptable and so it can be forwarded to the real server, or if it needs to be redirected to the honeypot if it is considered

suspicious; the honeypot has to be tuned in order to detect failures or sensitive worsening in the system after that the traffic was accepted. The architecture has the nice property to be adaptive, since if the traffic is considered non-malicious by the honeypot, the sensors can be re-tuned in order to allow the previously analyzed traffic to be immediately accepted.

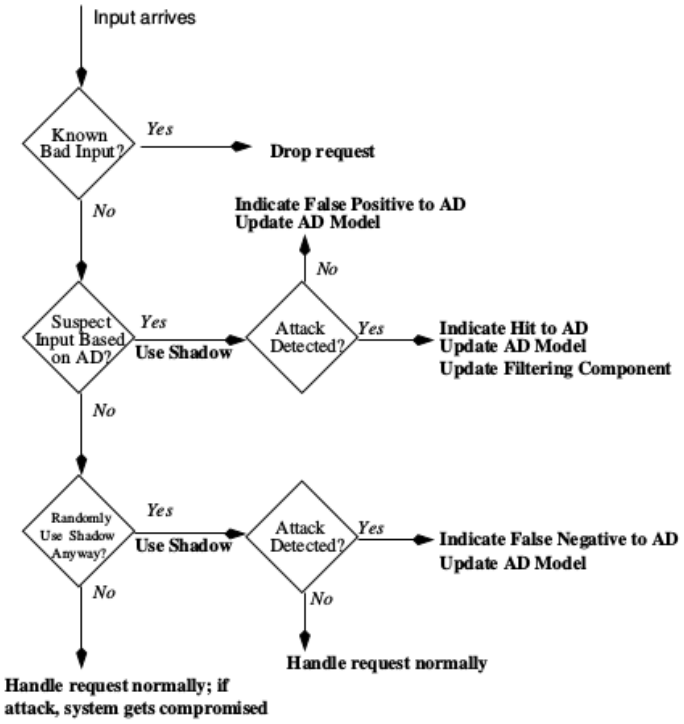


Figure 2.2: Shadow-honeypot workflow. (source:[16])

In the report *Honeypot Router for routing protocols protection* [17] it is proposed another approach in which the honeypot, called **Honeypot Router**, emulates a router in order to analyze attacks directed to the routing protocols. It works using the open-source routing software *Quagga* (that implements intra-domain routing protocols such as Routing Information Protocol which uses hop-count as routing metric, Open Shortest Path First (OSPF) which executes the SPF algorithm inside an Autonomous System domain and exterior-gateway algorithms such as Border Gateway Protocol) and other monitoring tools to capture packets, gain information about the flow and eventually stop it to prevent exploitation of the honeypot. One of the attacks that were studied against the honeypot was the MaxAge attack in the Open Shortest Path First algorithm: OSPF routers use Link-State Advertisement (LSA) packets to share information about their neighbors and the associated

routing metrics; MaxAge attack exploits the fact that routers running OSPF ignore LSA packets with age higher than 3600 s (1 hour) [18]; a common attacking pattern is to intercept the LSA packet, to modify its age to the maximum and re-injecting it towards the destination router with the purpose to cause a Denial of Service, excluding the victim router from the computation of the routing table of the destination router. In the test conducted by Ghourabi, this kind of attack was recognized by the honeypot by controlling the Wireshark capture, in which the malicious packets were identified by unexpected Time To Live value (which in this case has to be equal to 1 since LSA is sent towards neighbors) and checking the LSA age value, that revealed the nature of the attack.
All the previous cases analyzed so far involved static honeypots, in which its efficacy could be proven only if the attacker tried to engage an interaction with it; will be now discussed the case of a mobile honeypot, in which is the honeypot itself to seek for the attacker. This particular scenario is taken from the document *Intelligent Honeypot Agent for Blackhole Attack Detection in Wireless Mesh Networks* [19] in which it is discussed the application of a mobile honeypot in a hierarchical Wireless Mesh Network (WMN), which is composed by three levels: the top-level is composed by Internet Gateways wired to the Internet, below there is the mesh of static wireless routers which form the backbone of the WMN and allow the connection of the clients (that can be static or dynamic) to Internet (figure 2.3). The attack for which the honeypot is called to take remediation is named

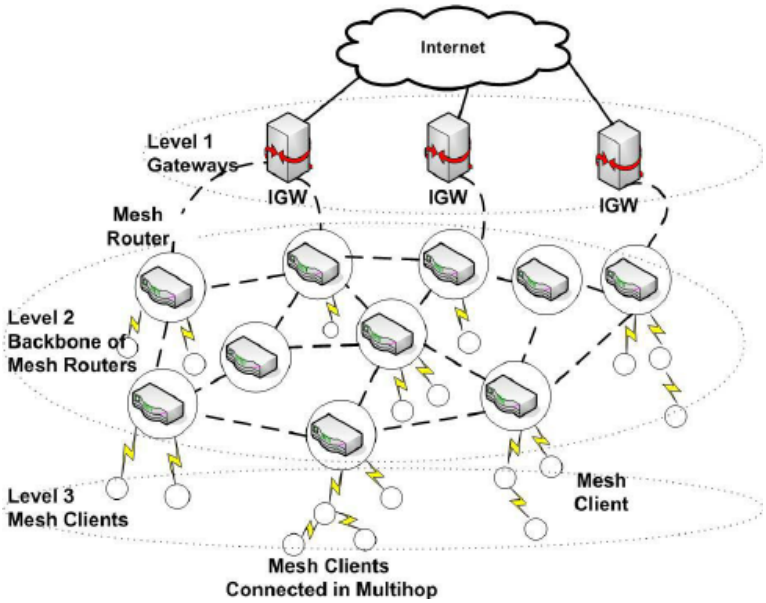


Figure 2.3: Hierarchical WMN architecture. (source:[19])

Black-hole attack, in which in the set of the mesh routers in the backbone there is

a malicious router which advertises itself as the best possible route and then drops all the received traffic in order to create Denial of Service. The test conducted by Prathapani involved routers using the On-Demand Routing Protocol in which the mesh router which wants to send traffic broadcasts a Route Request and if the neighbors have a fresh route toward the destination will reply with a Route Reply, else they will send a NULL packet to the source router and re-broadcast the received query; the black-hole router exploits the forge of Route Replies packets setting low hop count in order to be selected as the best node to traverse. As anticipated, in this environment the honeypot is supposed to move through the mesh, generating a Route Request to a known destination for which the best path to traverse is already known; after that the honeypot receives the Route Reply, it checks its validity forwarding a masked data packet to the mesh router which is unable to understand that it was sent by the honeypot; then, the honeypot sends a Query packet (which contains information to be compared with the masked packet that will eventually arrive) to the destination through the already known route; then the destination sends a Query Reply to the honeypot containing information regarding the last packet received by the mesh router under analysis; eventually, the honeypot decides if the mesh router has regular behavior or it is a black-hole router basing on the match between the information received in the Query Reply and the masked packet sent at the beginning of the process. Honeypot mobility is in this case required in order to check all the regions on the mesh of routers.

Honeypots and insider threats

The implementations seen so far made clear the fact that honeypots hold their value only when the attacker interacts with them. In the case of external threats it is more difficult for the attackers to distinguish between a real production machine and a honeypot, since they are moving inside a (presumably) unknown environment; the scenario changes dramatically in case of insider threats, in which the attacker is assumed to know very well the internal setting. Also, insiders are more difficult to detect since their goal usually is not to compromise or to take control of a machine, but to gain sensitive information. *Honeypots: Catching the Insider Threat* [20] observes that since insider's scope is to obtain useful information with low risks of being caught, it should be considered the possibility that the attacker is using packet sniffers which are hardly detectable and could expose sensitive information if the communication within the network is not ciphered. Spitzner's suggestion in order to discover such behavior is to send into network traffic some honeytokens (e.g. user credentials): once the honeytoken content is used, it will intuitively reveal the presence of the insider inside the environment. Another strategy analyzed by Spitzner is to exploit research engines in order to mislead the insider: the main idea is to create an honeytoken web-link for which the insider has no authorization to use. This idea is particularly valuable when the web search concerns topics that only an insider could know, such the development of a new prototype within the enterprise.
It needs to be remarked how the deployment of honeypots is a strategy that needs

to be handled very carefully especially in the case of the presence of insiders, since such deployment must remain undisclosed (known by as less people as possible) within the enterprise in order to be effective.

Strengths and weaknesses

By definition, honeypots hold their value when attackers attempt an illegal or unauthorized interaction with them, which implies that every data that the honeypot collects indicates a possible malicious behavior. It leads to the first advantage of honeypots with respect to traditional detection systems such IDS or firewalls: honeypots collect little but very **meaningful data**, which allows a much simpler analysis of the attack. Also, little data to process implies **little waste of resources**, while IDS and firewalls are very likely to saturate since they have to process much more data, and if saturation happens it can lead to harmful consequences. The third advantage of honeypot technologies is given by their **simplicity**, since they are basically devices which only need to run and monitor their status.[15]
The little amount of data to process is certainly a good feature, but what if honeypot collects no data? Is it possible to be sure that there are no threats inside the system? Unfortunately it is not so immediate to jump to such conclusion since one of the disadvantages of this technology is the **limited sight**: indeed, honeypots are able to detect only attacks that are directed to them, without knowing what else is happening in the other machines of the network. Moreover, honeypots are **identifiable**: errors on the emulation of a certain system or detection instruments such OS fingerprint could reveal to the attackers that they are facing an honeypot (e.g. the honeypot shows itself as a Windows server but it is emulated from a Linux machine); furthermore, some tools specifically designed to detect honeypots have been introduced in the market, such *Honeypot Hunter* by Send-Safe [21]. This product has been developed to discover if a proxy server is safe, is not working or it is an honeypot. The tool opens a false mail server on the system, then attempts a connection to its own false mail server through the proxy server: if the proxy server claims that the connection was successfully established but the mail server doesn't observe such connection, then the proxy server is an honeypot.
The paper *NoSEBrEaK - Attacking Honeynets* [22] illustrates several techniques to detect and avoid Sebek, a kernel rootkit which is used in honeynets to record all the actions performed by the intruder. Sebek is based on a customized `read()` system call which replaces the standard `read()` system call in the kernel, that allows it to have access to all the data passing through it, making it capable even to have access to encrypted messages; the recorded data is then sent to the Sebek server, always exploiting a modification at kernel level in order to hide the communication to the attacker. Since Sebek operates at kernel level, it is sufficient to observe anomalies in the kernel memory in order to have clues about the presence of that rootkit: when Sebek replaces the `read()` system call, replace the function pointer with its own pointer of the new inserted function in the `NR_read` entry of the system call table; in a normal environment, the `read()` and `write()` pointers are supposed to be located nearby, whereas in the presence of Sebek it is very probable that they are at distant slots.

Additionally, the same document [22] provides two simple ways to detect a honeywall. The first is based on the strict traffic limiter that honeywalls implement: sending a reasonable amount of outgoing connections requests makes possible to see if, at a certain point, the requests are blocked by the honeywall. The second method exploits the fact that some honeywalls modify outgoing packets that are considered potentially malicious in order to make them harmless; in this case it is sufficient for the attackers to initiate a connection in which they are in control of the packets at the source and at the destination and to check if the communication has been altered.
Detection of Virtual Environments and Low Interaction Honeypots [23] illustrates how timing analysis can be a valid technique to discover if a low-interaction honeypot is being employed. Empirically, it resulted that connections with low-interaction honeypots running on virtual environments on the same physical machine led to higher delays than connections with a normal machine; in particular, using a threshold delay of `0.44 ms` it is clearly possible to distinguish a regular host from a virtual honeypot. This behavior can be explained by the fact that packets (the tests were conducted with `ping`) have to pass through the physical machine link layer before being moved to the virtual environment; furthermore, if in the same physical environment lay different virtual machines, the operating system has to route packets through different processes which introduces further delays.
In the scenario in which the honeypot is detected, the attacker could decide to simply **bypass** the honeypot, or could **compromise** the honeypot in order to perform internal attacks or directed to the internet, or otherwise could **poison** the honeypot flooding it with false information which would make real attacks harder to detect under the generated noise [21]. Such disadvantages are the reason why honeypots cannot be employed as stand-alone security measure but need to be integrated with the traditional security mechanisms as IDS and firewalls.

Legal Issues

The employment of honeypot technologies could arise not only technical problems as seen in the previous paragraph, but also legal ones. Lance Spitzner [15] observes that the legal issues with honeypots (given for granted that the security administrator is respecting its enterprise's policies) may be due to the type of information that they retrieve from intruder's action, or by the actions that the intruder conducts from the honeypot itself: in particular, the three aspects to be focused on are *privacy, entrapment, civil liability*. Privacy aspect refers to the boundaries that need to be respected to protect user's privacy, even of intruders; high-interaction honeypots are able to capture intruder's communications, even if those communications do not involve malicious intentions. Entrapment is the *"inducement of a person to commit a crime, by means of fraud or undue persuasion, in an attempt to later bring a criminal prosecution against that person"* [24]; Spitzner observes that entrapment is mostly used for legal defense but does not really concerns honeypots since the attacker exploits the opportunity to commit a crime without knowing that the victim machine is a real host or an honeypot. Civil liability concerns the

responsibility of the enterprise in case the intruder performs illegal actions exploiting the honeypot: in particular, the company deploying the honeypot which is then exploited to conduct attacks to third party should have a duty to warn the third party about its system security; Spitzner suggests to keep the honeypot as secure as possible in order to avoid its exploitation, to monitor and eventually block outgoing traffic, and to pay attention to the data that is stored inside the system in order to prevent those circumstances. Scottberg [25] observes that in such scenario the system administrator could be persecuted for negligence as a *"hazard that was deliberately set up and not properly supervised"*.
In *Computer Security Education and Research: Handle with Care* [26] are considered relevant also the reasons behind the deployment of the honeypot: if the honeypot is used for research purposes and contains no valuable information inside, its employment can hardly be persecuted; instead, seeking for the attacker's computer is not advised, since it would legitimate eventual legal actions or the attacker could have used machines belonging to innocent third parties.
Those previous considerations where built upon the Federal laws of United States of America. According to Italian laws [27], the employment of honeypots can hardly be considered as entrapment; regarding privacy, Italian laws regulates personal data collection and treatment, not privacy as a more general term: as long as honeypots are employed to collect data for prevention and research, it is not considered law infringement.

2.2.2 Distributed Deception

Beyond Honeypots

One of the main features regarding honeypot technologies is that they are independent from the other security mechanisms present on the system; it can be considered an advantage since if an attack occurs they can be disconnected in order to study the collected data, but on the other hand studies suggest that it is better to have security devices which cooperate in order to have an orchestration view of the situation. *Ensuring security in depth based on heterogeneous network security technologies* [28] proposes an architecture to fully integrate honeypots with the other security technologies:

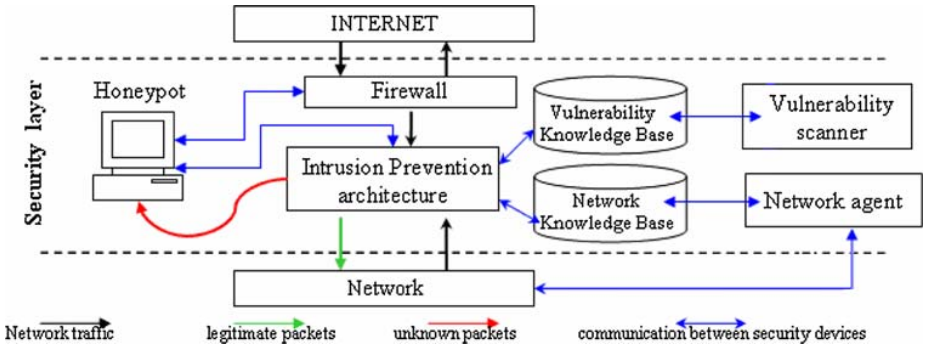


Figure 2.4: Source:[28].

- The first control of the traffic is made by the firewall which, in addition to regular traffic inspection, communicates with the honeypot and adapts its policies depending on honeypot's feedback;
- The intrusion prevention architecture is composed by a load balancer that divides the traffic toward several IPS and it receives information from the vulnerability scanner which analyzes the vulnerabilities that are present in the system. If the IPS architecture is not able to take a decision about some traffic pattern, it will be forwarded to the honeypot in order to gain some knowledge;
- The information collected by the vulnerability scanner are stored inside the Vulnerability Knowledge Base and updated at each new scan;
- The Network Agent collects all the information regarding the hosts present in the network (OS, applications, permissions) and stores it on the Network Knowledge Base.

Despite such architecture deploys the most popular security instruments designed to co-operate in order to achieve a coordinated security mechanism, yet it does not guarantee that the system is protected from stealthy or undetected intrusions.
Furthermore, the use-cases discussed in 2.2.1 involved mainly *research* honeypots, which purpose is to discover new information regarding the currently employed attacking patterns; on the other hand, enterprises are interested in keeping secure their assets, and honeypots often resulted too expensive and too difficult to manage [29].

The Deception paradigm

Honeypots can be a valid support for deception operations, but their isolated deployment cannot be sufficient in order to realize a full cyber denial and deception campaign. In the introductory paragraph 1.2 it has been introduced the Cyber Kill-Chain, namely the typical steps which characterize an advanced persistent

threat; cyber deception can be applied at every step of the cyber kill-chain, as it is summarized in the table below and analyzed in depth in the 2.2.2 section.

Cyber Kill-Chain phase	**Deception action**
Reconnaissance	Artificial ports, fake sites
Weaponization and delivery	Artificial bouncing back
Exploitation and installation	Artificial exploitation response
Command and Control (C2)	Honeypots
Lateral Movements	HoneyTokens, HoneyFiles
Exfiltration	HoneyTokens, endless files, fake keys

Table 2.1: Mapping deception to the kill-chain model. (source: [30])

In the reconnaissance phase the attacker tries to gain knowledge about target's vulnerabilities, hence the deception strategy should be focused on providing misleading information about the defending system or monitor the knowledge of the attacker using web footprints; the weaponization phase can be efficaciously neutralized if the attacker wrongly believes that the defender's system has certain vulnerabilities; if an exploitation attempt is recognized, the attacker could be fed with false information in order to make him/her believe that the operation has been successful (also honeypots could be valuable) ; if the attackers have reached the Command And Control stage, they should be redirected to an honeypot in order to understand their goals and to keep the intruders in an isolated environment; in the exfiltration phase, the goal of the deception strategy is to delay attackers' operations and to exhaust their resources [31].
A cyber denial & deception strategy is not meant to be applied to defend the system from a single incident, but it is a cyclic process that aims at protecting the system against a full attacking campaign, not an isolated attacking effort. The **Deception Chain** [4] is a ten-step process which gives guide-lines in order to properly design and implement a deception campaign:

- **Purpose:** The goal of the deception operation. In this phase must be indicated the criteria that define the deception's success;
- **Intelligence gathering:** this phase is focused on the study of the adversaries, which could be their possible objectives, which information will be shown or will be hidden, how the intruders may react, how to monitor their actions;
- **Design of the Cover Story:** it is defined as "the deception version of what the target of the deception should perceive and believe"[4]. Since the D&D strategy is based on the provisioning of false information, it must be considered that the intruders may notice incongruities which could lead the full deception operation to failure. In this phase must be decided what needs to be hidden (EEFI and NDDI) and what needs to be created and revealed (EEDI and NEFI) to make the environment safer and the cover story believable;
- **Plan:** in this phase are developed the denial methods to hide the undisclosable elements and are analyzed the environment characteristics in order to create

deceptive components able to properly mimic them. Since not everything can be hidden, it is suggested to rest on the non-essential friendly information in order to make the deception story more believable;

- **Prepare:** it analyzes the feasibility of the deception operation, considering the available resources and the coordination with other operators in the system in which deception strategy has to be applied in order to avoid conflicts between the real and the false environments;

- **Execute:** if the preparation phase did not signal conflicts between the real and the deceptive operations, the deception strategy can be now executed. This phase represents the ending of the planning phases and the beginning of the monitoring phases;

- **Monitor:** the full environment and intruder's actions are monitored and, if needed, operational mistakes are fixed;

- **Reinforce:** After the monitoring phase it is possible to see if the deception story is effective or it needs improvements. If the deception strategy does not lead to the expected results programmed in the Purpose phase, it could be needed to re-plan another deception operation.

Taxonomy

The steps presented above represent a general guide-line for the implementation of a deception plan, but the strategies, the goals and the technologies can be very different from case to case. *Design Considerations for Building Cyber Deception Systems* defines **explicit deception** the scenario in which the deception story has voluntarily low credibility: this strategy can be useful to raise doubts about the validity of the data that the intruder has already obtained.
If the deception strategy is scheduled to generate same responses to certain interactions, then it is named **static deception**; if instead the response changes over time as to mimic a real-time changing environment, then it is called **dynamic deception**.
The deception campaign can be put in place before that an attack is detected (**proactive** approach) or after that an intrusion is detected in the system (**reactive** approach); these two approaches may be complementary, since D&D plan is usually thought to cover attacks at different steps in the cyber kill chain.

Use-cases

Cyber Denial, Deception and Counter-Deception [31] provides two valuable cases in which the D&D techniques and methodologies prove their efficacy both at the attacking and at the defending sides.
The first scenario which is analyzed is the infamous Stuxnet campaign, a cyber attack conducted against the Iranian's government centrifuge facilities for uranium

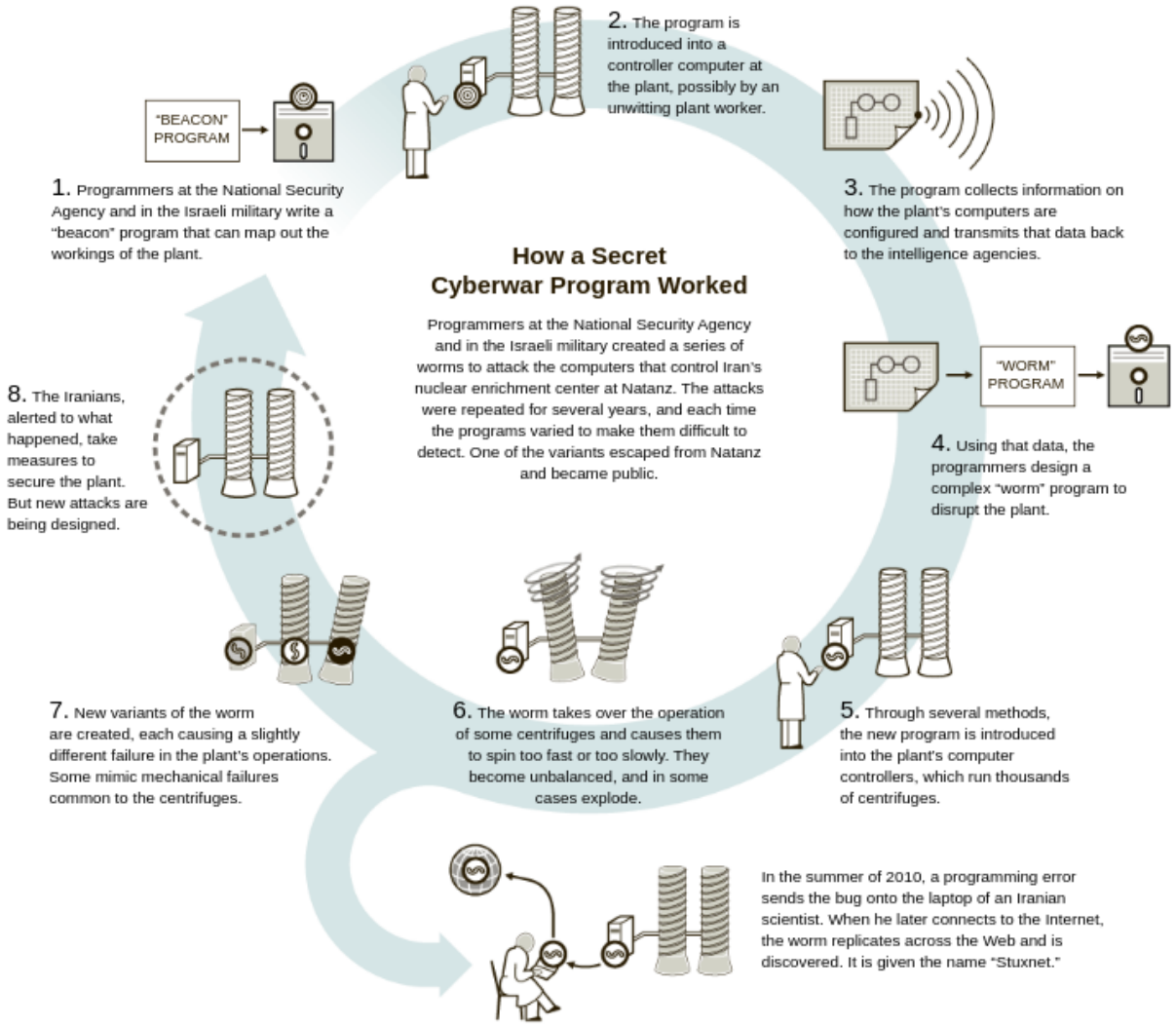


Figure 2.5: Source:[32]

enrichment. The attackers developed the Stuxnet worm in order to physically destroy the centrifuges (figure 2.5 shows the steps of the Stuxnet strategy). Stuxnet employed some deceptive techniques which allowed the worm to secretly propagate and replicate itself, to bypass and avoid intrusion-protection systems, to block alerts before reaching the operators and to spoof monitoring systems to indicate that the system was not manipulated. Furthermore, the attack was conducted with operations which had the purpose to attribute the causes of the attack to engineering and design mistakes (the centrifuges were disassembled many times by Iranian operators in order to understand the failures). The starting point of the attack was the

malware Duqu, which was used in the intelligence gathering phase of the deception chain to collect useful data and assets, exploited later-on to conduct the offensive operation; the detailed intelligence gathering about control systems and procedures allowed Stuxnet to work in a well-known environment and to be highly adaptive and diverse in the attacking procedure against the centrifuges; furthermore, it was programmed to be self-erasing in order to not be identified even after that the malfunctioning in the system was detected (in some system the self-deletion failed, allowing the reverse-engineering of the malware).
The Stuxnet case is helpful in order to comprehend the potential of a cyber D&D campaign properly orchestrated, but since the scope of this thesis is to evaluate defensive deception strategies, it is worth to examine the defensive use-case against a fictional APT espionage attempt proposed by [4], where it shows that the deception chain can be applied in all the steps of the cyber kill-chain: it is assumed that the defender is developing a prototype which information has to be protected through deception, and that the attacker will follow the cyber kill-chain steps (not necessarily in a linear fashion) introduced in 1.2, and has the ability to breach inside the defender's private network, steal credentials and operate as legitimate user.
During the **reconnaissance** phase, the attacker uses search engines and social networks in order to gather information about employers and enterprise's organizational information; in order to anticipate the attacker, the defender builds the deception strategy as follows:

- *purpose*: defender's scope is to make the intrusion attempts ineffective and to create a model able to catch attacker's methods;
- *intelligence gathering:* knowing a priori that the attacker will go through the kill chain and will eventually breach inside the private network, the defender installs a honeynet in order to get information about attacker's TTTPs and creates fictitious, exploitable user profiles and accounts;
- *cover story design:* fictional, misleading elements about the prototype have to be designed;
- *plan:* the data that supports the cover story is made accessible to attacker through the honeynet or search engines;
- *prepare:* additional material is designed in order to cover and compensate the potential leak of EEFI and to make more believable the cover story as to add confusion and misconception to the offender about the real disclosed information;
- *execute:* honeynet is put in place together with the fictitious documents supporting the cover story;
- *monitor:* activity on the fictitious created accounts is monitored, along with the honeynet.

Once the attacker obtained information about target's vulnerabilities through the recon phase, the proper exploits will be prepared through the **weaponization**

phase; the defender's strategy is now different from the previous one since the attacker is on a further step of the kill chain:

- *purpose:* the defender wants to redirect the attacker to the wrong targets and detect attacker's weaponization choices;
- *intelligence gathering:* basing on the current technology used for exploitation and on the misinformation voluntarily leaked about its own system, the defender can get some hints about the possible attacker's actions;
- *prepare and execute:* defender deploys other decoy systems with well-known vulnerabilities inside the internal network;
- *monitor:* it allows the defender to understand if the false vulnerabilities are believed by the offender.

The next step in the cyber kill chain is the **delivery phase**, which strongly depends on the victim's environment and security architecture. Also in this phase the defender has the opportunity to develop a deceptive strategy:

- *purpose:* the primary goal at this stage is to detect the deliver attempt; once the attempt is detected, the defender may decide to deny the access, or to give a feedback as to make believe that the delivery was successfully performed;
- *intelligence gathering:* the fake user profiles are monitored in order to see social engineering attempts;
- *execute:* the defender interacts with the attacker's efforts;
- *Reinforce:* in case the intruder has obtained access into the network, new deception elements may be deployed as to conceal EEFI or to lure him/her towards misleading information.

In the **exploit phase**, the intruder may leave some traces about the exploitation attempt. This phase is very sensitive from defender's point of view since the attacker is now inside the network and the damages could be significant:

- *purpose:* the defender must keep the system safe, which means that the attacker must be redirect toward a controlled environment;
- *plan:* the deployment of low-interaction and high-interaction honeypots could give believable feedback to the attacker and could improve the knowledge about attacker's weapons;
- *execute:* defender interacts with the attacker in order to make him/her believe that the exploitation succeeded and that the defender is relying on weak defending systems;
- *reinforce:* the feedback provided should tempt the attacker to use weaker tools.

In the **control/execute phase**, the attacker installs additional malware, obtains valid credentials and performs lateral movements. The deception strategy in the depicted scenario is based on the following steps:

- *purpose:* as for the previous phases, the attacker should be redirected to monitored environments;

- *execute:* new user credentials may be revealed to the attackers along with EEDI in order to give them access to the safe environment and to strengthen their confidence about the efficacy of the used exploits;

The last step in the kill chain is the **maintain phase**, where the attacker's goal is to preserve presence inside the victim's network; in this phase the intruder's actions may be very rare, and the deception should allow the defender to keep the intruder under control:

- *purpose:* the defender's goal is to control attacker's interactions and disrupt the control that has gained inside the network;

- *monitor/reinforce:* the EEDI content is continuously fed to the attacker in order to keep the cover story believable;

- *design cover story:* the attacker has been inside the system for a while, therefore the previously deception story could be not so believable; in this phase, another deception level could be added in order to give a more solid foundation to the weaker sides of the previous story and to lead the attacker further from the true valuable data.

Distributed Deception Platforms

In the very recent years, the improvement of the technology in terms of virtualization capabilities led to the rise of Distributed Deception Platforms (DDP) as a new model for cyber-deception, potentially capable to overcome honeypots' limitations. The research studies regarding those platforms are very scarce, presumably for two reasons: distributed deception platforms have been introduced very recently in the market, mainly by startups; the DDP architecture does not really implements new technologies from the ones that have been debated so far, but aspires to allow an easier deployment, monitoring and management of the deception components, now distributed all over the network [33]. Deception platforms have a centralized management system to create and distribute the deception elements, which fundamentally can be:

- **decoys:** low-interaction or high-interaction honeypots;

- **breadcrumbs and lures:** fake information used to address the attacker towards the decoys or to provide him/her misleading information. Example of lures are user-credentials, application shortcuts, browser history, log files, user certificates, database files etc.;

- **honeytokens:** data similar to lures, but while lures lead the attacker towards the decoys but do not generate an alarm if there is an interaction, honeytokens are monitored in order to trigger an alert in case of interaction;

Analyzing the deployment of those three categories, decoys operate within the network, therefore their deployment is not complex to manage; lures or breadcrumbs have to be positioned within the endpoints, which leads to a more challenging deployment; honeytokens have to be positioned within endpoints too, but their degree of complexity is even higher since they play a central role in the credibility of the deception story, therefore they require more attention in the design phase (DDPs may include some features which analyze the internal environment in order to create concealing deception components). In order to make the deception story believable, it is also fundamental to keep the centralized management system hidden from the attacker, since its disclosure would reveal the deceptive strategy (it is a *NDDI*, introduced in 1.1).
The architecture introduced in *Automating the Injection of Believable Decoys to Detect Snooping* [34] can be considered a prototype of distributed deception architecture which allows to create automatic decoy traffic in order to tempt potential network sniffers to reveal their actions. The false traffic injected into the the network has to respect five principles, enunciated in *Baiting inside attackers using decoy documents* [35]: believability, non-interference, detectability, variability and enticement. Believability means that decoy data must be unrecognizable with respect to regular traffic; non-interference refers to the likelihood that regular users can access to normal documents without impediment; detectability means that decoy traffic should generate an alert if some action involves their use; variability indicates "the quality of being subject to variation", which means that decoys should not have some identifiable pattern which differentiate them from the regular traffic; enticement indicates the decoys' appeal and strongly depends on attacker's purposes or preferences.
The architecture proposed in [34] is composed by a decoy traffic generator, a distribution platform and several broadcasters to inject different traffic patterns. The model claims to generate believable decoy traffic, called *honeyflow*, starting from recorded network traffic (template or real captures) as input data. The automated **traffic generator** separates each session/protocol into individual trace files; each trace is passed through a traffic identifier to find the best match with the traffic known by the system (if no match is found, it is marked as unknown traffic); for the identified flows it is inserted decoy information on headers (e.g. making the IP addresses conform to the subnet in which the decoy traffic is going to be injected) or on payloads (e.g. web page content, email body); then, the different traces are combined into a single trace, sent to the **decoy distributor** which delivers the trace to the different **decoy broadcasters**, which are general-purpose wireless routers that must be placed near an Access Point and have to be able to inject the traffic (therefore monitor mode must be enabled, since in other modes the injection fails or is limited). The same authors developed a monitoring system which can be integrated to collect the data generated by the misuse of the decoys [35]: the monitoring platform is able to control the activity related to the fake accounts that are created and which credentials are injected into the network, to store suspicious IP addresses and send an alert for each of their operation in the network; an alert is also generated whenever it is not possible to log into a bait account, since it indicates that its password has been changed.

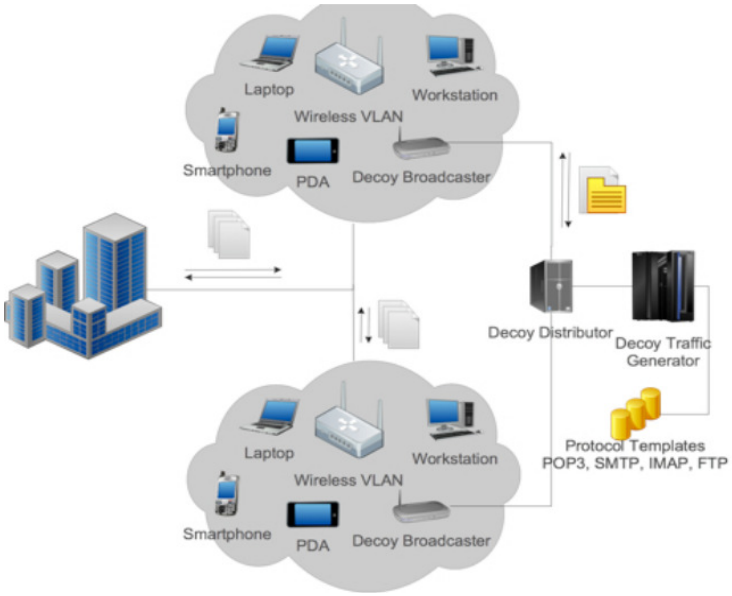


Figure 2.6: Injection Platform. (source:[34])

The distributed platforms developed few years later are more general purpose than the previous case which only scope was to catch eavesdroppers; DDPs aim at providing security through deception against the Advanced Persistent Threats phases discussed in 1.2. The next chapter treats the methodology used to test the platforms, with an in-depth analysis on the category of attacks that will be simulated.

Chapter 3

Methodology

Summary

Here it is reported the methodology regarding the testing phase of some of the deception technologies available on the market: the test will involve the simulation of an advanced persistent threat through the use of a Breach and Attack Platform, then the deception platforms will be evaluated by the use of APT observables.

3.1 Advanced Persistent Threats

In the introductory section 1.2 have been illustrated the six typical steps characterizing advanced persistent threats: *reconnaissance, intrusion, backdoor instalment, lateral movements, install utilities, data exfiltration.* The implementation of those stages is now analyzed using *A study on advanced persistent threats* [36] as main reference. The reconnaissance phase refers to the collecting of information usually exploiting social engineering techniques in order to gain access inside the target's network and to know in advance the environment in which the intruder will be in. Social engineering techniques are usually employed to gain access inside the network exploiting psychological manipulation, but the attackers are also eager to obtain information about the environment, configurations and assets that they are willing to breach. For this purpose they rely on Open-Source Intelligence (**OSINT**), data and resources freely available on the internet.

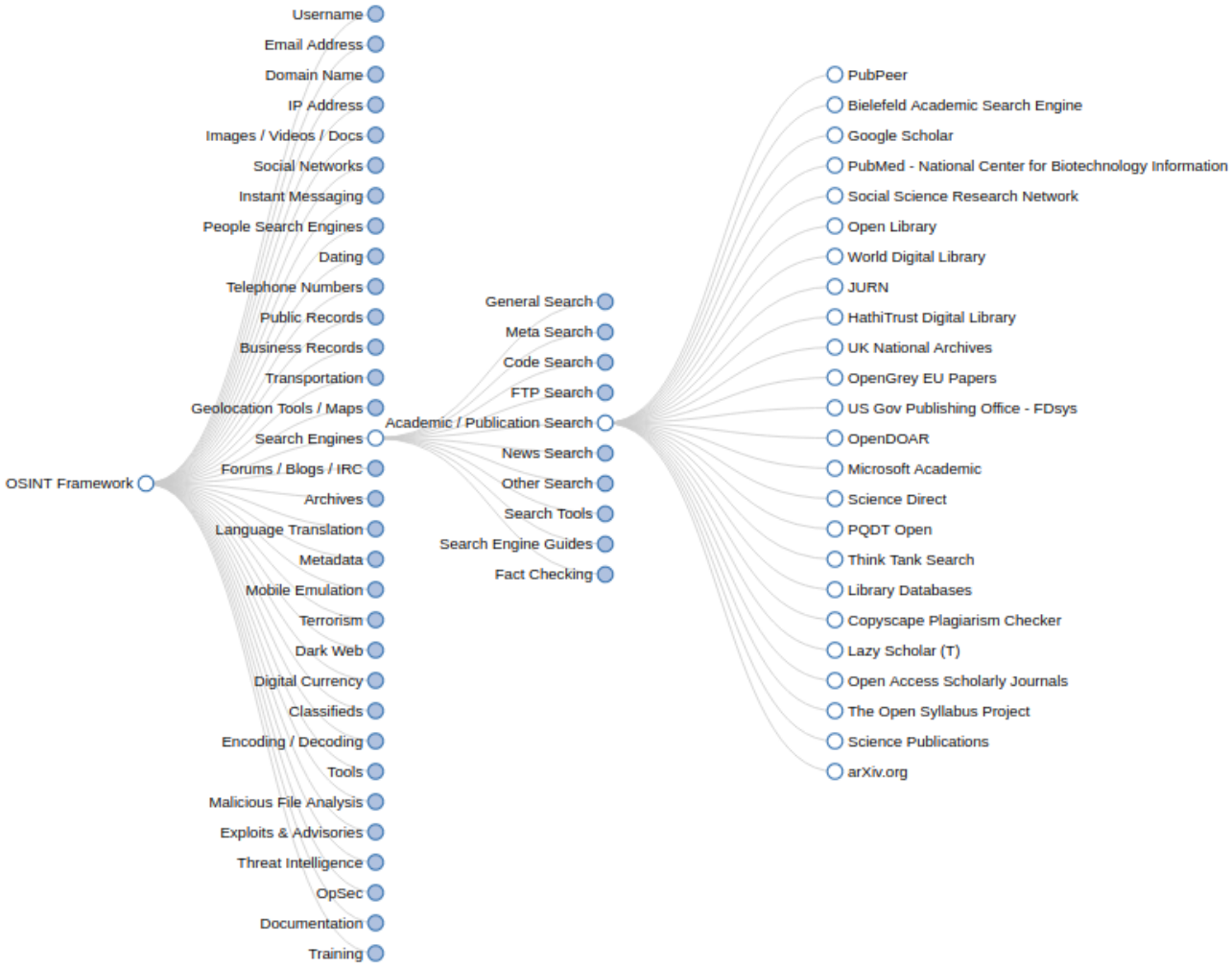


Figure 3.1: OSINT Framework. (source:[37])

The intrusion phase is executed mainly in two ways: with a legitimate access using user credentials obtained through the social engineering procedure, or by exploitation of vulnerabilities discovered in the reconnaissance phase. Once the intrusion has been performed, the attacker installs a backdoor in the machine in order to have access to the target's network.
In order to expand their presence inside the network, attackers perform lateral movements: lateral movements consist in the progressive unauthorized access (possibly achieved also through legitimate credentials) of devices inside the victim's network. The attacker exploits vulnerabilities relating the authentication mechanism, which could be the use of single sign-on, brute-force password stealing, software vulnerabilities or even simple packet sniffing.[38] Usually the goal of lateral movements is to reach the Domain Controller hashed password in order to obtain the `ntds.dit` file which contains all the hashed passwords of all the users of that domain. The activities regarding lateral movements are usually slow and can last

long periods of time since the attacker has to gain knowledge about the the next systems to exploit and has to avoid raising suspects. In order to avoid detection, it is common that the attacker installs different tools and utilities typically used by IT administrators in order to make their actions more believable. The final step of advanced persistent threats involves data exfilitration: when the attackers have been able to establish their presence inside the network and have reached sensitive information, they may want to to destroy it or to exfiltrate it by sending (with secure protocols as SSL or relying on Tor Network) the undisclosed data on an external server which is no longer under the victim's control. In the next paragraph will be enunciated the configuration choices in order to simulate the actions of an advanced persistent threat.

3.2 Observables

In this section are discussed the metrics used to evaluate the deception tools employed in the Intesa Sanpaolo network. The products will be evaluated basing on the APT's 9 **observables**, defined in the text *Reverse Deception: Organized Cyber Threat Counter-Exploitation*. [39] The information that the deception technology under evaluation is able to retrieve will be used to fill the observables table; seven observables can be evaluated on a scale from 1 to 10 danger level, being 1 very low danger and 10 very threatening.

Attack origination points	1-10
Victim involved in the attack	1-10
Risk tolerance	1-10
Timeliness	1-10
Skills and methods	1-10
Actions	1-10
Objectives	1-10
Resources	N/A
Knowledge source	N/A

Table 3.1: APT's observables table. (source:[39])

3.2.1 Attack origination points

The "attack origination points" observable allows to define if the network is subject to a random attack or it is specifically tailored for the victim's system. Below it is provided a table which categorizes the seriousness of the attack:

1	Accidental opening of an infected file
2	Digital device infection
3	Random client-side exploit against a browser
4	Exploit via a social networking site
5	Custom server-side exploit kit (professional)
6	Custom tailored client-side exploit against a browser
7	Insider implemented infection
8	Custom-tailored attachments with embedded infectors
9	tailored spear phishing (horizontal and vertical)
10	direct-tailored whaling e-mail

Table 3.2: Evaluation table for attacking origination points. (source:[39])

The attacks caused by casual or "general-purpose" infection are considered low-level threats since no peculiarities regarding the enterprise are exploited to conduct the attack; are considered medium-level threats the ones involving professionals attackers which have the purpose to infect as many networks as possible in order to gain some sensitive information which could lead to economic advantage; the high-level threats are the threats involving specifically tailored attacks such whaling, in which are targeted the top-executives of the organization.

3.2.2 Victim involved in the attack

This observable aims at gaining knowledge about the goal of the intruder by analyzing which component within the enterprise was the victim of the attack:

1	The system of a low-level employee
2	The system of a low-level manager
3	The system of a network administrator
4	The forward-facing systems (DMZ, web-application servers)
5	The system of an administrator's assistant (exposed to sensitive information)
6	The DNS servers of the organization
7	The mail servers of the organization
8	The primary file or database servers of the organization
9	Security team's systems
10	The systems of core stakeholders or organizational leadership

Table 3.3: Evaluation table for Victims involved in the attack. (source:[39])

3.2.3 Risk tolerance

Risk tolerance observable refers to the effort that the attacker has put into not getting caught. This observable allows to understand if the attacker has the will to remain inside the victim's network ans therefore if the organization is facing an Advanced Persistent Threat or not:

1	No logs were altered
2	Login/access logs were altered
3	Connection logs and times were altered
4	Entire systems logs were wiped
5	Entire system logs were corrupted
6	Operating system security services were disabled
7	Specific security services were disabled
8	Specific applications were corrupted
9	Operating system was corrupted
10	Entire system was corrupted or disabled

Table 3.4: Risk tolerance evaluation table. (source:[39])

3.2.4 Timeliness

Timing analysis allows to comprehend the depth of information retrieved by intruders regarding the information: if the attack was conducted in a brief amount of time, hitting specific sensitive components of the enterprise may indicate an high knowledge by attackers of the internal structure of the organization; another element that is worth considering is at which time of the day the attack is conducted in order to presume the possible geographic location or habits of attackers.

1	Multiple systems accessed for long periods of time
2	Multiple systems accessed for long periods of time in specific locations
3	Multiple systems accessed for long periods of time involving specific applications
4	Few systems accessed for long periods of time involving specific information
5	Few systems accessed on regular basis targeting specific file types
6	Few systems accessed on regular basis within a specific internal team
7	Few systems accessed a few times within a specific internal team
8	A single system of a specific member of a team accessed on a regular basis briefly
9	A single system of a specific member of a team accessed a few times and briefly targeted
10	A single system was accessed directly and briefly

Table 3.5: Timeliness evaluation table. (source:[39])

3.2.5 Skills and methods

The Skills and methods observables is needed to evaluate the attacker's competences. The aspects that have to be considered involve the tools used, the vulnerabilities of the organization, the exploitation technique, the attacking patterns, the data transfer techniques and the logging alteration/deletion techniques; other elements such keystroke analysis and behavioral profiling can be helpful in order to get additional information (age, gender, culture) about the adversary.

1	Open source publicly available tools accessible using basic techniques
2	Open source publicly available tools accessible using some custom techniques
3	Open source publicly available tools accessible using completely custom techniques
4	Customized open source tools accessible using completely custom techniques
5	A combination of customized open source and commercial tools using custom techniques
6	A combination of customized open source and commercial tools using professional techniques
7	customized and commercial tools using professional techniques and observable patterns of previous intrusions
8	Completely customized tool suite with medium knowledge of operating system commands tailored for the victim environment
9	Completely customized tool suite with deep knowledge of operating system commands tailored for the victim environment
10	Customized tools never seen in previous attacks tailored for the victim environment

Table 3.6: Skills and methods evaluation table. (source:[39])

3.2.6 Actions

Actions observable aims at identify the systems that were compromised and how, looking for a possible pattern to be identified along with the Skills and methods observable and adding another layer of knowledge about the motives that led the intruders to perform their actions.

1	Threat uses the system as training point without causing harm
2	Threat uses the system to store peer-to-peer files for torrent seeds on the system
3	Threat is a worm spreading itself on the system
4	Threat is a standard malware from a random infector site
5	Threat uses the system as part of a botnet
6	Threat uses the system as a part of a criminal network to steal information
7	Threat uses the system to coordinate attacks against external systems
8	Threat uses the system to coordinate attacks against internal and external systems to gain advantageous position across the enterprise, partners, customers or third parties
9	Threat uses the system to coordinate attacks against internal and external systems to gain specifical information within the enterprise
10	Threat uses the system to coordinate attacks against internal and external systems to gain specifical information within the enterprise and to sell it to criminal groups

Table 3.7: Actions evaluation table. (source:[39])

3.2.7 Objectives

The Objectives observable is based on the analysis of the stolen/deleted data on the defending enterprise.

1	Seemingly curiosity
2	Targeting login information
3	Targeting organizational information (e-mails, logins)
4	Targeting organizational, partner and customer information
5	Targeting organizational user's personally identifiable information (PII)
6	Targeting organizational user's financial information
7	Targeting organizational financial information
8	Targeting organizational operational, financial and research information
9	Targeting specific high-profile organizational members' information
10	Targeting specific high-priority classified information AND all of the above

Table 3.8: Objectives evaluation table. (source:[39])

3.2.8 Resources

This is one of the two observables which do not allow a quantitative measure. Understanding the resources of the threat requires an orchestration view of the previous observables in order to determinate if the attacker is at a low level on the criminal food chain or it is a state state-sponsored cyber threat. The most important features that need to be evaluated are: the skills and methods which give an hint about the threat's budget and may also allow to understand if the attacker has been trained or is self-educated; the timeliness of the attack, which can discriminate if the attack is performed during a spare-time or it follows typical work-time schedule.

3.2.9 Knowledge source

As discussed in 3.1, the sources of information cover a very wide range, which makes it difficult to understand from which source the information has been retrieved. The most common sources of information are security sites, forums, hackers' private sites, social networks, search engines and many more.

3.3 Attack Simulation

In order to re-create a believable scenario in which the deception technologies have to prove their worth, it will be used the Guardicore open-source Breach & Attack platform, *Infection Monkey*.[40] The platform comprises two components: the *monkey* which executes the attacks, and the *Infection Monkey Island*,a Command and Control Platform that allows to configure and monitor the attacks. This tool has been chosen in order to create a post-breach attack which simulates typical Advanced Persistent Threats behavior as lateral movements; also, the choice of such platform allows to create an attacking pattern exactly reproducible for all the deception tools that will be tested. The machine from which the monkey is executed scans for the IPs that were indicated in the configuration file, then attempts OS fingerprinting and tries different exploits in order to take possess of the machines. The

Mitre ATT&CK Matrix [9] provides an overview of all the tactics and techniques used by the attackers in order to achieve the goals after the exploitation stage. The infection Monkey integrates some of the attacks enhunciated in the Mitre matrix in order to re-create a believable APT behavior.

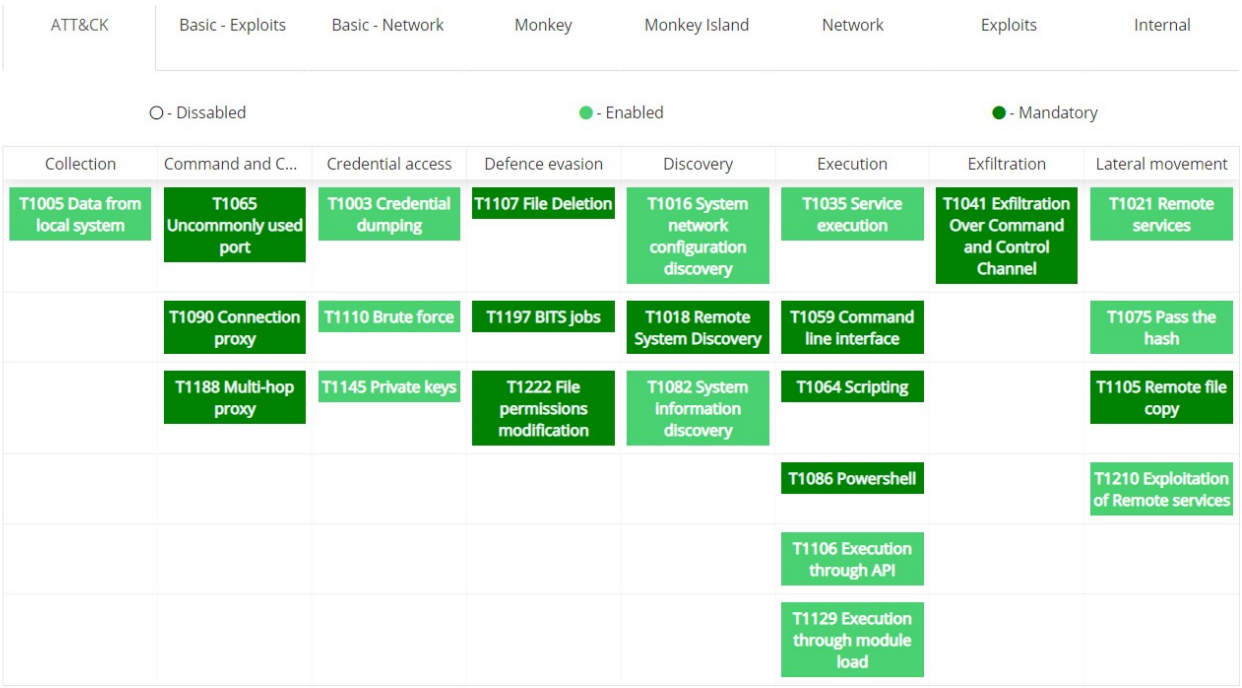


Figure 3.2: Mitre Att&ck Matrix implemented in Infection Monkey. (source:[3])

The implementation of such matrix allows to collect sensitive data from local systems, to establish a Command and Control communication through proxies over non-standard ports to bypass firewalls, to delete footprints of the ongoing attack, to discover the systems within the network, to exploit the C2 communication to execute services and to exfiltrate data, and to perform lateral movements using valid user credentials or by the Pass the Hash method, which exploits the hashed password stored locally to gain access without possessing the clear-text password.

3.3.1 Exploiting user credentials

The reconnaissance and intrusion stages will be simulated from within the victim's network, supposing that the attacker is already inside the system (it could be the case of insider threats) and has obtained a set of possible user credentials that can be used to perform lateral movements. Infection Monkey allows to make use of stolen user credentials through `SSH`, or `SMB` and `WMI` if the victim is using a Microsoft system with specific unpatched vulnerabilities.

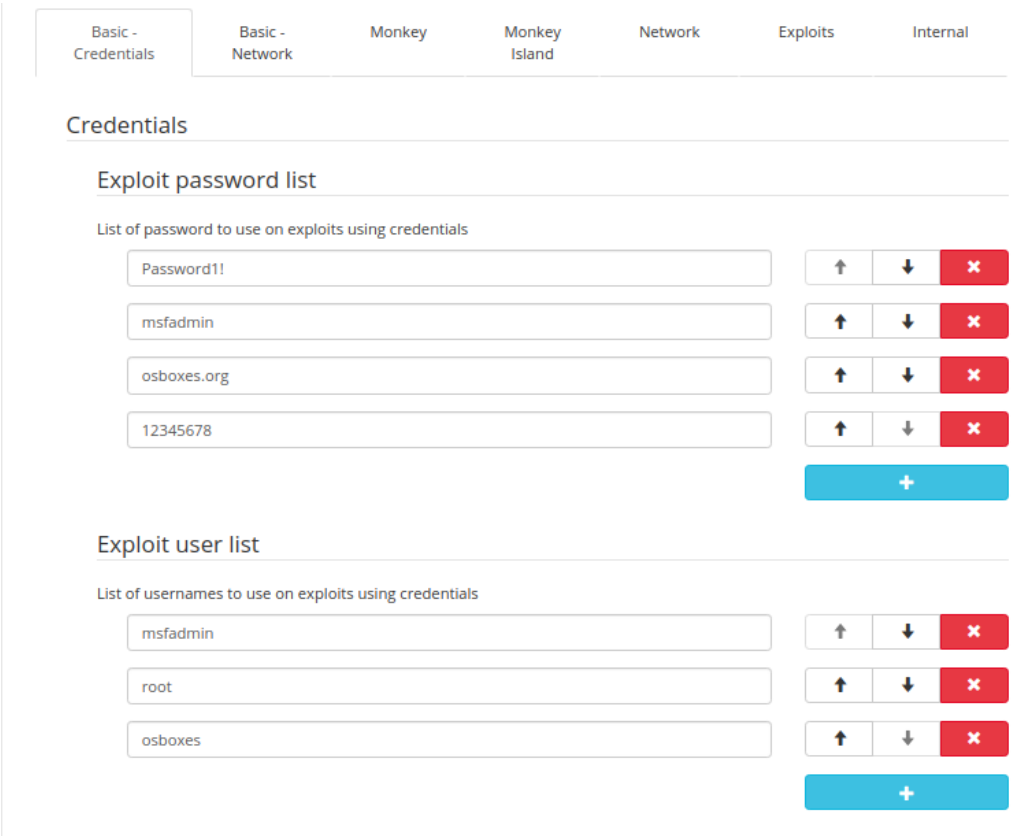


Figure 3.3: Infection Monkey's configuration page for the exploitation through basic credentials.

3.3.2 Backdoor instalment

The next step in the kill chain that will be executed if the attack has been able to have access to the machine is the backdoor instalment: if in the network are present Windows machines which are vulnerable to the `MS08_067 Conficker` exploit (Windows 2000 or Windows XP SP2) present in the Infection Monkey database, it will allow to install a backdoor setting the backdoor credentials in the monkey configuration page and to dump a monkey inside the machine to continue the intrusion through lateral movements exploiting the protocols described in 3.3.1.

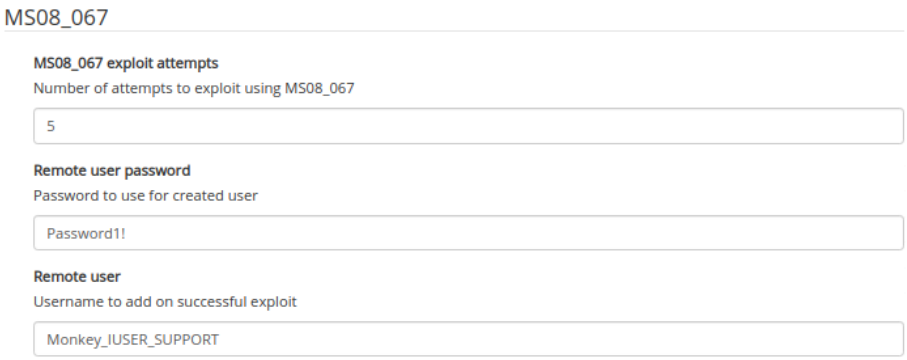


Figure 3.4: Infection Monkey's settings for the MS08_067 Exploit.

In the case of a Linux machine, the monkey will run an executable which will establish a connection with a remote server and will be now able to send commands to be executed from the victim machine and to receive the corresponding output. The code of the custom backdoor has been written in Python and is reported in 6.1.

3.3.3 Install utilities

The infection monkey allows the simulation of the download and execution processes of a generic file by exploiting the `RDP` protocol, after attempting a brute-force login with the provided credentials in the configuration stage of the simulation.

3.3.4 Data Exfiltration

The data exfiltration stage is implemented through the backdoor communication between the infected machine and the Command and Control machine.

Chapter 4

Test and Results.

4.1 Cowrie Honeypot.

4.1.1 Description.

The first test is performed on an opensource medium-high interaction honeypot named Cowrie, which will allow to better comprehend the differences and the possible enhancements brought by the distributed deception platforms with respect to classic honeypots. Cowrie [41] is a SSH/Telnet honeypot, capable to collect logs about brute-force login attempts and the entire shell actions performed by the attackers. It provides a false filesystem that allows to create bait files to which the attacker could be interested in, and has inspection capabilities about the files downloaded through the `wget` utility. Cowrie is very simple to implement since it needs only a script to be executed, and it has been confirmed compatible both with Linux and Windows distributions; for the purposes of this thesis, it has been executed over a Ubuntu 16.04 distribution.
Cowrie is configured by default to accept any login credentials that are provided by the attackers, presenting them a **false shell** to which they can interact with.

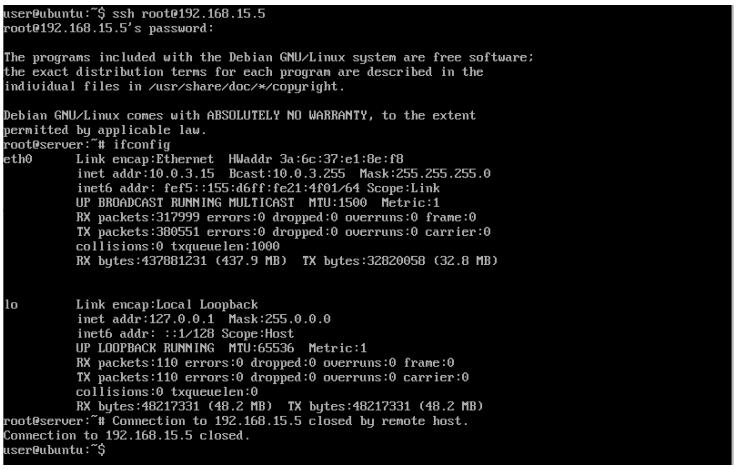


Figure 4.1: Interaction with Cowrie honeypot. The execution of ifconfig command leads to a false output.

Since this configuration could lead to an easy recognition of the honeypot, the configuration file is changed in order to allow access only to the `root:Admin123` credentials.

4.1.2 Results and comments.

The test is executed in a local network environment, composed by 15 production devices and the Cowrie honeypot. The attack is performed by Infection monkey as described in section 3.3. The actions of the Infection Monkey platform are resumed in the following graph:

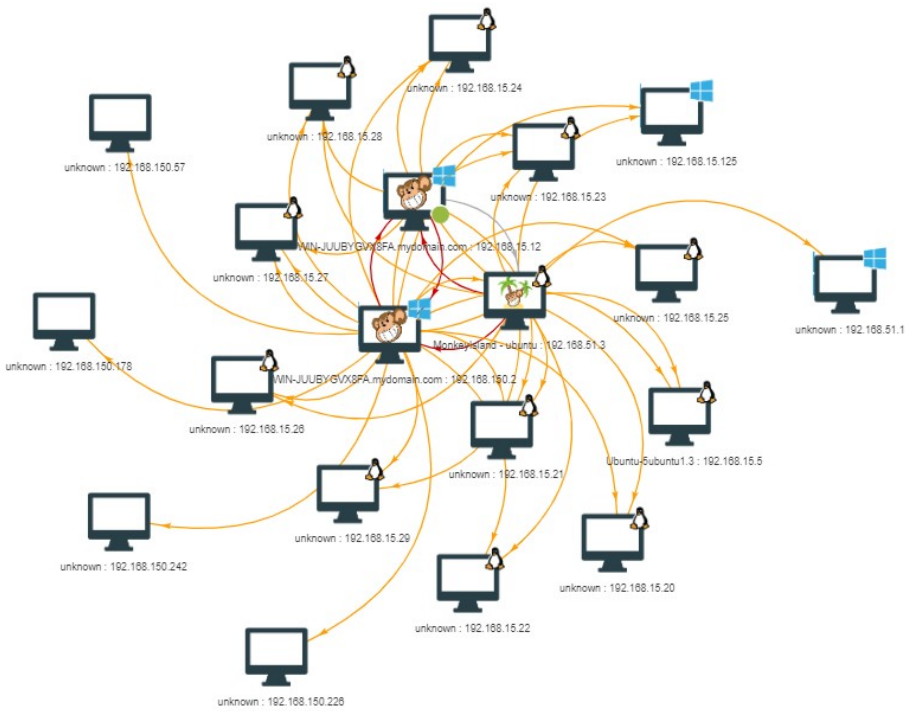


Figure 4.2: Infection monkey interactions within the network.

The Breach & Attack platform was configured as to search for sensitive data after that a system was exploited, and then to display its content:

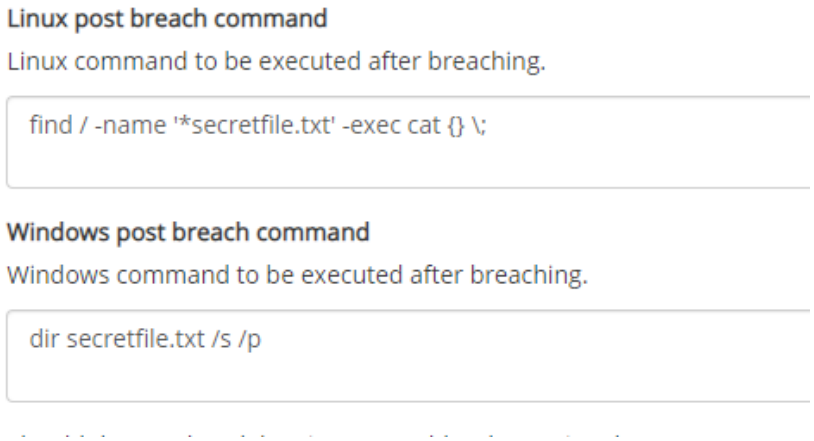


Figure 4.3: Post breach actions setup in Infection Monkey.

The attacker behavior has been properly captured by the Cowrie honeypot which shows that, as the Infection Monkey attempts the execution of the `find` command, it is denied by the honeypot since the attacker has not access to the real server's shell.

```
root@ubuntu:/opt/cowrie# bin/playlog  var/lib/cowrie/tty/6c17588eba694b14b058c707cbf6623a134aaacd979
7b1fbf6b6e2ff22589812

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@server:~# find / -name '*secretfile.txt' -exec cat {} \;
-bash: find: command not found
```

Figure 4.4: Keylog of the commands executed by the attacker, displayed through the Playlog functionality.

Cowrie implements **Playlog** as log visualizer, which can be a very effective tool to perform forensics analysis about the attackers, since the collected keystroke logs are displayed in the same exact way as they were prompted, which allows to observe interesting features as deletions, typing speed etc.
With the information collected by the Cowrie honeypot is now filled the observables' table in order to evaluate the knowledge gained by the defender through the deployed tool, compared with the real observables of the APT, known in advance:

Observables	Real Score	Perceived Score
Attack origination points	7	4 - 7
Victims involved in the attack	8	4
Risk tolerance	4	1
Timeliness	7	7
Skills and methods	4	1
Actions	6	6
Objectives	3	3
Resources	medium	low
Knowledge source	Undefined	Unknown

The honeypot allows to understand that the attacker is acting within the internal network, which indicates an exploitation of the machine or that the malicious actions are perfomed by an insider; since the attack is programmed to replicate itself once a machine is exploited, Cowrie's logs about exploitation attempts suggest that the attack was not targeted against a single machine, and the keylog of the attacker's commands indicate that the attacker targeted sensitive information, but the Cowrie honeypot is not able to determine that the full network is under attack; since the log deletion is not performed through the SSH channel by the Infection Monkey, the honeypot is not able to individuate this attacking feature; the Timeliness observable is marked as correct since the logs collect the timestamp of the malicious activities; about the skills and methods, the honeypot is only able to see the SSH connection request, which does not indicate advanced skills in the attacking attempt; the keylogging function allows to correctly individuate the attacker's intentions, as well as the Objectives observable.

4.2 Modern Honey Network.

4.2.1 Description.

The next test involves an hybrid architecture in-between honeypots and distributed deception platforms: indeed, Modern Honey Network (MHN) [42] is an opensource project which allows to create a centralized server that handles honeypots and collects the generated alerts, but the deployment of the honeypots has to be performed manually in already existing machines, since the MHN console does not have virtualization capabilities to automatically deploy virtual decoys; moreover, the complete logs are not collected by the server, but must be checked inside the involved honeypot. The installation of the MHN console is compatible with Ubuntu and CentOS servers only, same goes for the supported honeypots, with the additional support on Raspberry-Pi for few honeypots.

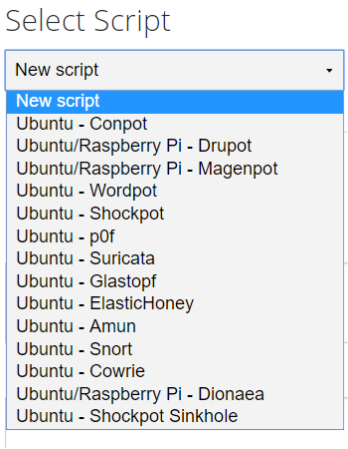


Figure 4.5: MHN supported honeypots.

- Conpot [43] is a low-interaction honeypot to be employed in industrial systems to simulate ICS/SCADA systems;
- Drupot honeypot [44] emulates a Drupal platform, used for the creation and distribution of dynamic web-sites;
- MagenPot [45] stands for Magento Honeypot, a fake web Content Management System for e-commerce;
- Wordpot [46] allows detection of fingerprinting attempts over the server running Wordpress;
- Shockpot [47] allows the detection of Shellshock exploitation attempts;
- p0f [48] performs passive fingerprinting on the probing machine;
- Suricata [49] includes an Intrusion Detection System, Intrusion Prevention System, Network Security Monitoring and pcap files processing;

- Glastopf [50] is a web application honeypot which implements vulnerability type emulation, allowing the analysis of unknown attacks;
- ElasticHoney [51] has the purpose to discover Remote Code Execution exploitation attempts over the Elastic Search service;
- Amun [52] is a low-interaction honeypot with enhanced log collection capabilities;
- Snort [53] is an Network Intrusion Detection System;
- Cowrie is a SSH/telnet medium-interaction honeypot;
- Dionaea [54] is a honeypot for malwares, which purpose is to block and analyse the captured malware;
- Shockpot Sinkhole [55] implements the sinkhole feature to Shockpot.

Once the selection is made, Modern Honey Network allows the customization of the honeypot's script and provides the command to be executed in the machine that has to be turned into a honeypot.

Figure 4.6: MHN honeypot's customization interface.

4.2.2 Configuration.

In order to test the functionalities of Modern Honey Network, 4 different honeypots types have been deployed: the first one is the **Cowrie** SSH/Telnet honeypot already discussed in the previous section; a second server runs **P0f**, a passive fingerprinting tool which monitors the incoming traffic of the selected network interface of the server; the third server runs **Amun**, a low-interaction python honeypot hosting several services and providing wide log collection; into the last server is implemented **Snort**, to implement control at network level. Once the deployment scripts are executed on the selected servers, the honeypots are bound to the MHN monitoring console:

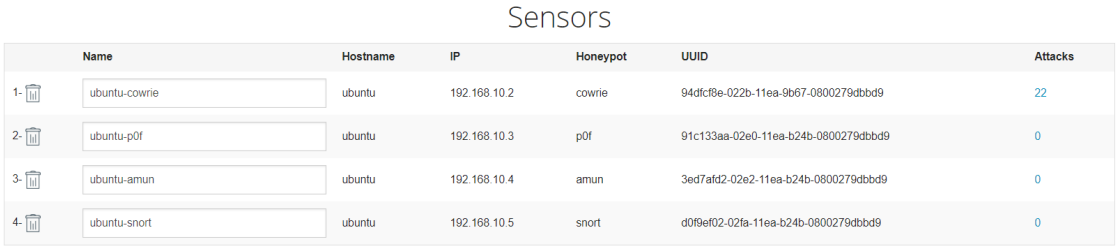


Figure 4.7: Deployed honeypots through the MHN console.

4.2.3 Results and comments.

The attack is performed through Infection Monkey in order to emulate Advanced persistent Threat's stages, as explained in section 3.3. The map of the attacker's interaction within the victim's network is reported below:

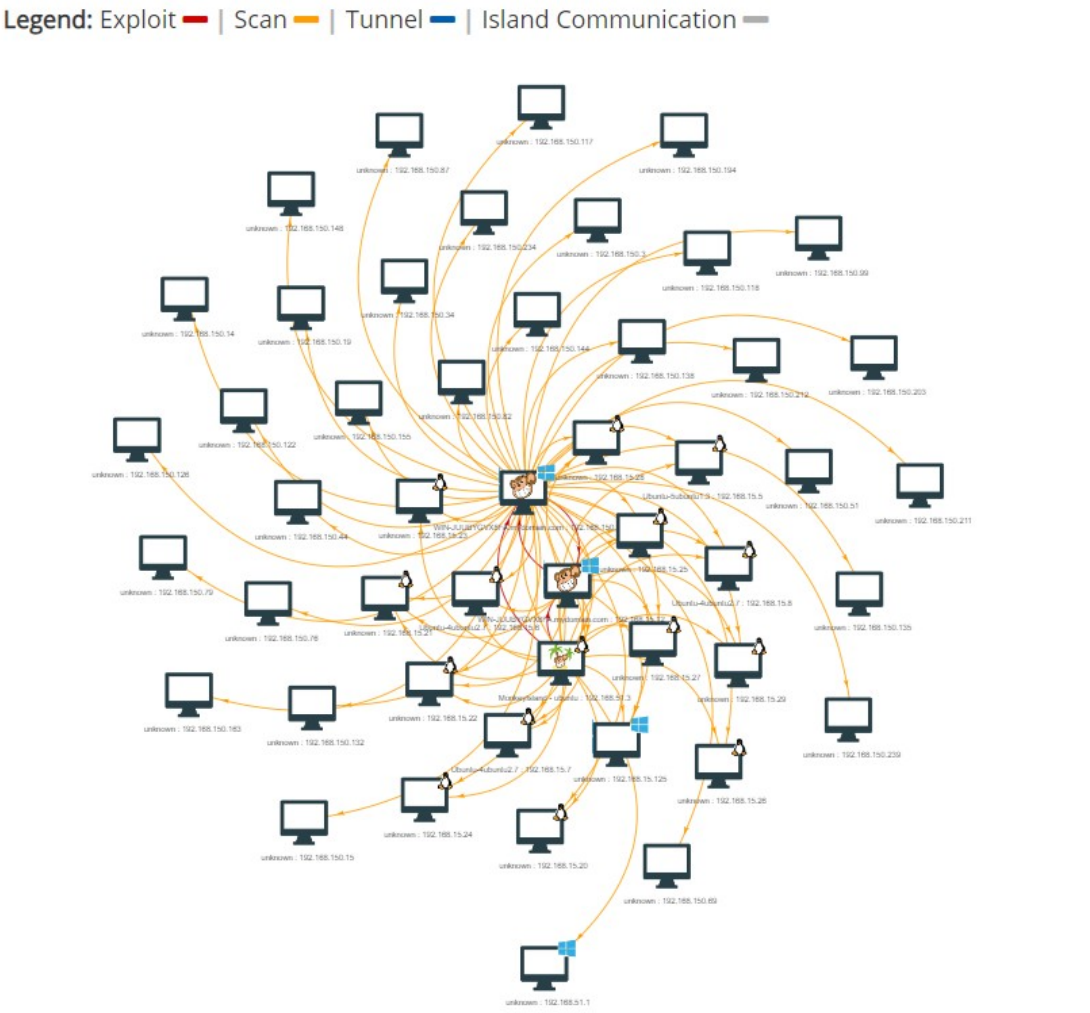
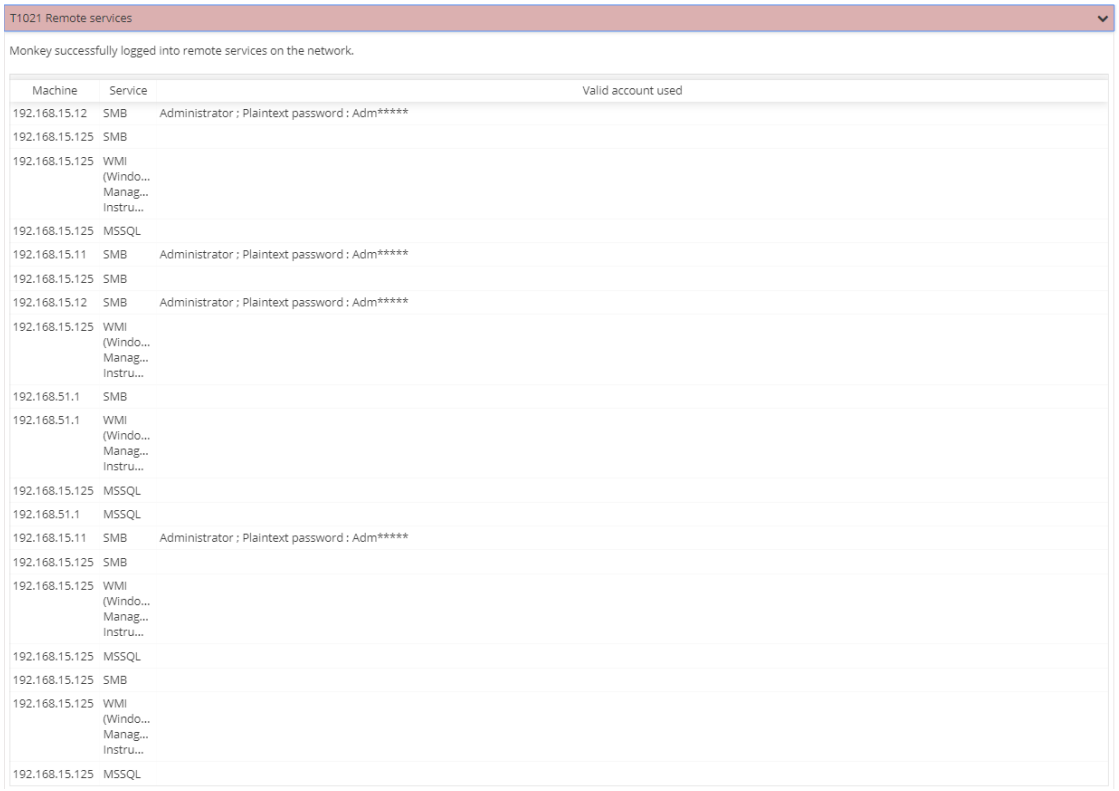


Figure 4.8: Infection Monkey interaction map.

The map shows that the network has been scanned multiple times, since the Infection Monkey was configured to replicate the actions within the exploited machines (the windows servers, exploited through the SMB service). The breach and attack platform discovered the following services within the network:



T1021 Remote services

Monkey successfully logged into remote services on the network.

Machine	Service	Valid account used
192.168.15.12	SMB	Administrator : Plaintext password : Adm*****
192.168.15.125	SMB	
192.168.15.125	WMI (Windo... Manag... Instru...	
192.168.15.125	MSSQL	
192.168.15.11	SMB	Administrator : Plaintext password : Adm*****
192.168.15.125	SMB	
192.168.15.12	SMB	Administrator : Plaintext password : Adm*****
192.168.15.125	WMI (Windo... Manag... Instru...	
192.168.51.1	SMB	
192.168.51.1	WMI (Windo... Manag... Instru...	
192.168.15.125	MSSQL	
192.168.51.1	MSSQL	
192.168.15.11	SMB	Administrator : Plaintext password : Adm*****
192.168.15.125	SMB	
192.168.15.125	WMI (Windo... Manag... Instru...	
192.168.15.125	MSSQL	
192.168.15.125	SMB	
192.168.15.125	WMI (Windo... Manag... Instru...	
192.168.15.125	MSSQL	

Figure 4.9: Services discovered by Infection Monkey.

Additionally, since the SSH exploit of the Infection Monkey did not worked properly, the bruteforce login attempts and the post-breach actions have been simulated manually from the infected machine toward the deployed honeypots through the **Hydra** [56] opensource tool.
The Modern Honey Network console displays the overall statistics collected by the deployed honeypots through the main dashboard:

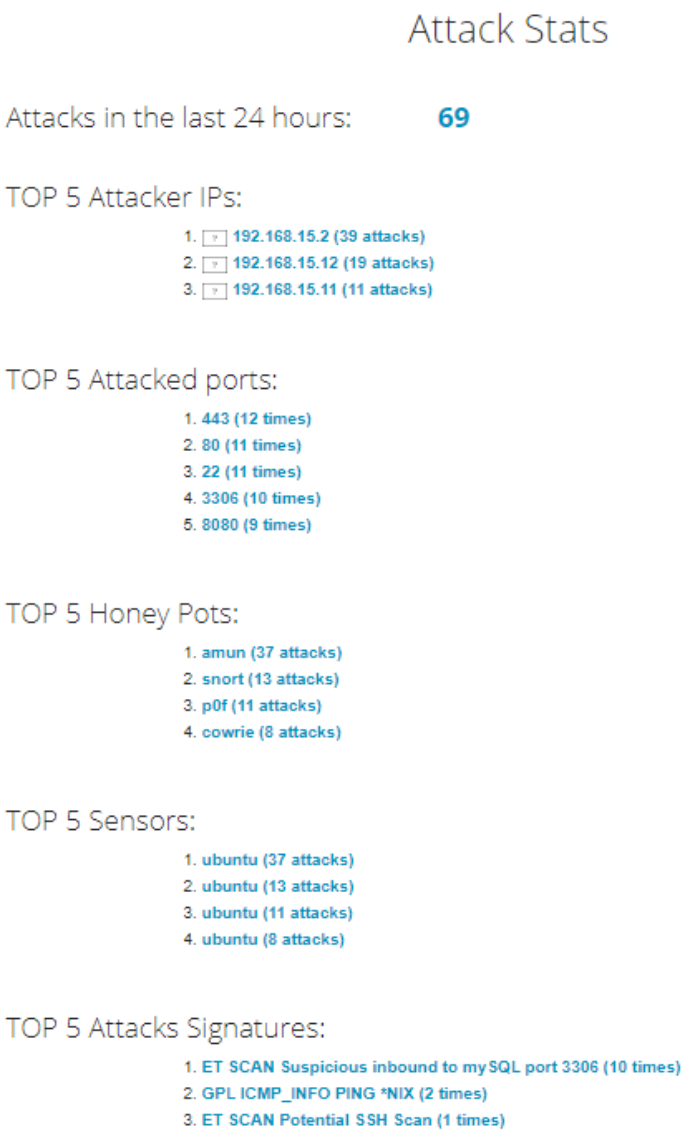
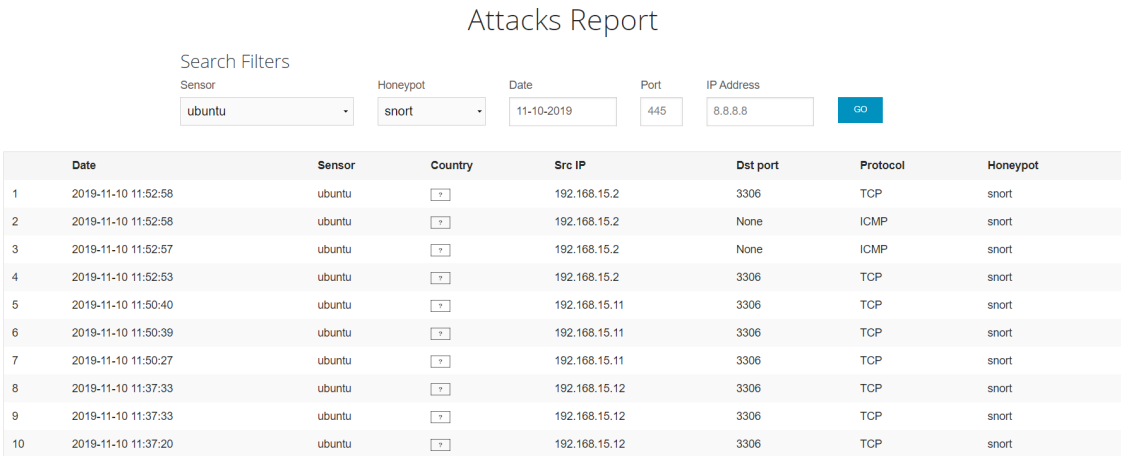


Figure 4.10: Modern Honey Network main dashboard.

The dashboard displays the IP addresses of the malicious attempts, which belong to the Infection Monkey machine and to the exploited Windows servers which replicate the attack; the attacked ports displayed are the open ports of the deployed honeypots, the port 22 hosts the SSH service, the ports 80, 8080 and 443 host a web-application and the port 3306 hosts mySQL; then the dashboard displays that all the honeypots have been triggered, and reports some signatures which suggest which kind of actions have been performed by the intruders. The logs collected by the MHN interface display which honeypot collected the data, the timestamp, the IP interacting with the honeypot, the protocol and the triggered port:



	Date	Sensor	Country	Src IP	Dst port	Protocol	Honeypot
1	2019-11-10 11:52:58	ubuntu	?	192.168.15.2	3306	TCP	snort
2	2019-11-10 11:52:58	ubuntu	?	192.168.15.2	None	ICMP	snort
3	2019-11-10 11:52:57	ubuntu	?	192.168.15.2	None	ICMP	snort
4	2019-11-10 11:52:53	ubuntu	?	192.168.15.2	3306	TCP	snort
5	2019-11-10 11:50:40	ubuntu	?	192.168.15.11	3306	TCP	snort
6	2019-11-10 11:50:39	ubuntu	?	192.168.15.11	3306	TCP	snort
7	2019-11-10 11:50:27	ubuntu	?	192.168.15.11	3306	TCP	snort
8	2019-11-10 11:37:33	ubuntu	?	192.168.15.12	3306	TCP	snort
9	2019-11-10 11:37:33	ubuntu	?	192.168.15.12	3306	TCP	snort
10	2019-11-10 11:37:20	ubuntu	?	192.168.15.12	3306	TCP	snort

Figure 4.11: Log display from the MHN monitoring interface.

The Charts section displays the most used combinations of username and password to login into the Cowrie honeypot and the IP addresses which attempted the attack:

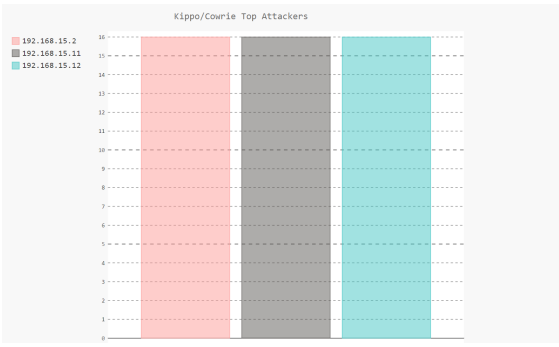


Figure 4.12: IP addresses chart.

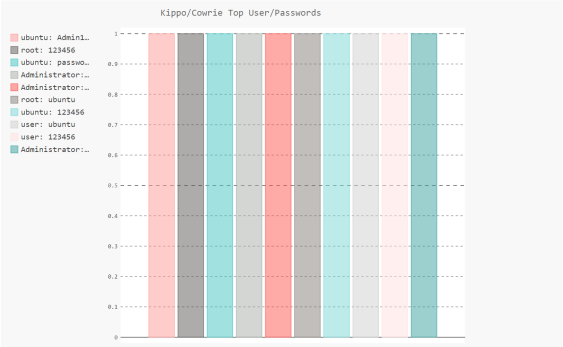


Figure 4.13: Username-password chart.

P0f allows to perform passive fingerprinting of the attacking machines, discovering that the machine from which the attack began is a Linux machine, which then propagated the attacks to the Windows 2008 servers of the network:

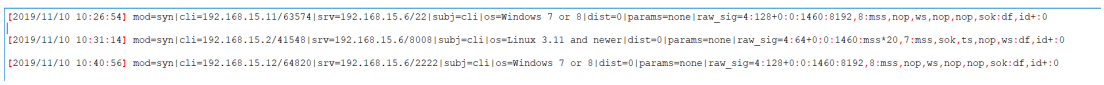


Figure 4.14: P0f's OS fingerprinting.

Snort logs traced the ICMP, UDP and TCP traffic generated by the attacking machines: in particular, the TCP connection is established in the SSH exploitation event, while ICMP has been used to discover hosts in the network and UDP to perform port scan:

```
11/10-12:05:40.699860 192.168.15.2:40766 -> 192.168.15.8:22
TCP TTL:64 TOS:0x0 ID:41671 IpLen:20 DgmLen:60 DF
******S* Seq: 0xB6E18DD2  Ack: 0x0  Win: 0x7210  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 208661 0 NOP WS: 7
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

11/10-12:05:40.699877 192.168.15.8:22 -> 192.168.15.2:40766
TCP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:60 DF
***A**S* Seq: 0xC70E8452  Ack: 0xB6E18DD3  Win: 0x7120  TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 194659 208661 NOP WS: 7
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

11/10-12:05:40.701144 192.168.15.2:40766 -> 192.168.15.8:22
TCP TTL:64 TOS:0x0 ID:41672 IpLen:20 DgmLen:52 DF
***A**** Seq: 0xB6E18DD3  Ack: 0xC70E8453  Win: 0xE5  TcpLen: 32
TCP Options (3) => NOP NOP TS: 208662 194659
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Figure 4.15: 3-Way Handshake of the SSH connection, capture from Snort.

Logs collected from Amun honeypot are able to recognise the port scanning and detect the attempt to exploit the Web-Server through port 8080, without being able to categorize this kind of attack:

```
2019-11-10 10:28:03,905 INFO [amun_request_handler] PortScan Detected on Port: 8080 (192.168.15.11)
2019-11-10 10:28:18,906 INFO [amun_request_handler] unknown vuln (Attacker: 192.168.15.11 Port: 8080, Mess: [...]
2019-11-10 10:28:24,927 INFO [amun_request_handler] unknown vuln (Attacker: 192.168.15.11 Port: 8080, Mess: [...]
2019-11-10 10:31:06,481 INFO [amun_request_handler] PortScan Detected on Port: 445 (192.168.15.2)
2019-11-10 10:31:06,483 INFO [amun_request_handler] PortScan Detected on Port: 135 (192.168.15.2)
2019-11-10 10:31:06,483 INFO [amun_request_handler] PortScan Detected on Port: 443 (192.168.15.2)
2019-11-10 10:31:06,484 INFO [amun_request_handler] PortScan Detected on Port: 3389 (192.168.15.2)
2019-11-10 10:31:06,485 INFO [amun_request_handler] PortScan Detected on Port: 8080 (192.168.15.2)
2019-11-10 10:31:18,482 INFO [amun_request_handler] unknown vuln (Attacker: 192.168.15.2 Port: 8080, Mess: [...]
2019-11-10 10:31:27,481 INFO [amun_request_handler] unknown vuln (Attacker: 192.168.15.2 Port: 8080, Mess: [...]
```

Figure 4.16: Amun log events.

Cowrie's detection capabilities have already been discussed in section 4.1, therefore with the aforementioned logs the observables' table can now be filled, comparing the results with the real characteristics of the APT:

Observables	**Real Score**	**Perceived Score**
Attack origination points	7	**4 - 7**
Victims involved in the attack	8	**8**
Risk tolerance	4	1
Timeliness	7	**7**
Skills and methods	4	4
Actions	6	**6**
Objectives	3	**3**
Resources	medium	**medium**
Knowledge source	Undefined	Unknown

The Modern Honey Network observable table inherits the correct scores obtained

in the Cowrie analysis of section 4.1, and enhances the score of the Victims involved in the attack, since Snort and Amun detect ICMP and UDP traffic which aims at gaining information on the full network, while Cowrie had limited sight about these events; the trace captured by Amun honeypot determines also an uncategorized attacking pattern, which could possibly indicate more advanced techniques adopted by the attacker, while the Cowrie honeypot only detected SSH attempts.

4.3 Dejavu Deception Framework

4.3.1 Description

The first distributed deception platform to be tested is the *Dejavu Deception Framework*, an open-source project developed by Bhadresh Patel and Harish Ramadoss [57]. The architecture is built upon the Management interface and the Decoy interface: the management interface is a centralized console which allows to handle the distribution of the decoys, honeytokens and controls the alerts triggered in case of suspicious activities against the distributed components; the decoy interface is a virtual sensor which communicates directly with the deployed decoys, forwards the notifications to the management interface allowing it to be detached from the monitored network.

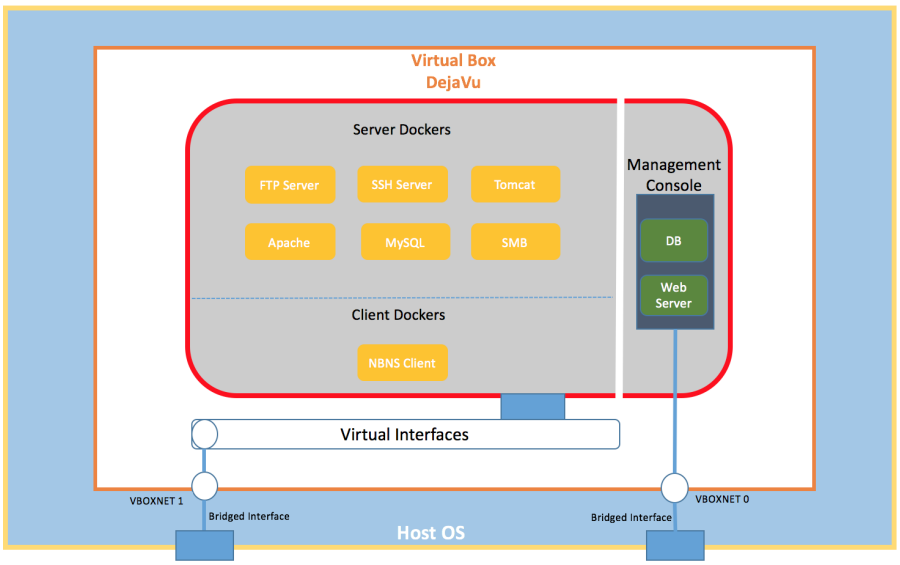


Figure 4.17: Dejavu Architecture. (source [57]

The Management console allows to create new client and server decoys, but does not allow to specify the distribution of such decoys; it is possible to indicate the services that the server decoys must provide and it is possible to create fake user credentials associated to existing domains thorugh a shell script to be run inside the domain controller; finally, the Log Management interface provides a graphical view and raw logs of the incidents that were detected by the sensor.

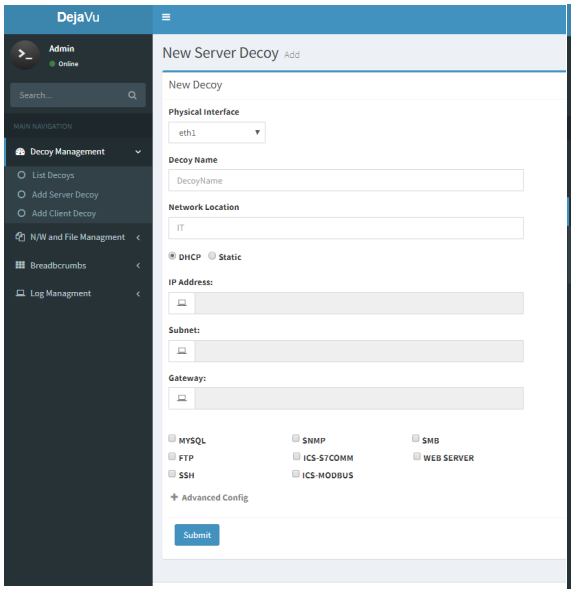


Figure 4.18: Server configuration.

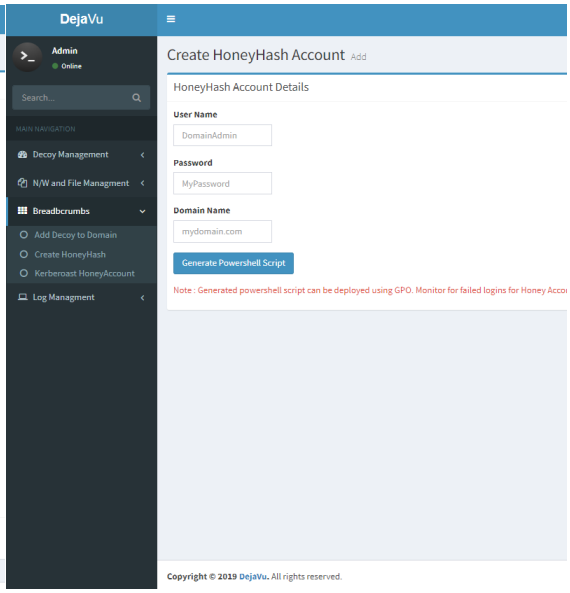


Figure 4.19: Honeytoken configuration.

4.3.2 Configuration.

In order to test the performances of the Dejavu framework, the decoys are deployed respecting the 1:2 ratio with respect to the number of clients and servers present inside the network (i.e. there is one decoy for every 2 real systems). The server decoys are configured to have the `MySQL`, `FTP`, `SSH` and `Apache Web Server` services.

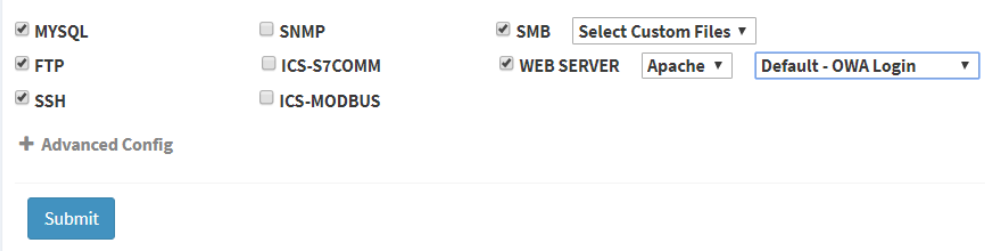


Figure 4.20: Setup choice for server decoys.

4.3.3 Results and comments.

Are now reported and commented the results of the attack performed by the Infection Monkey platform and the information collected by the Dejavu deception platform.
The attack starts from the Infection Monkey sensor located inside the network, and after attempting the procedures described in section 3.3, the overview of the network from the attacker point of view is the following:

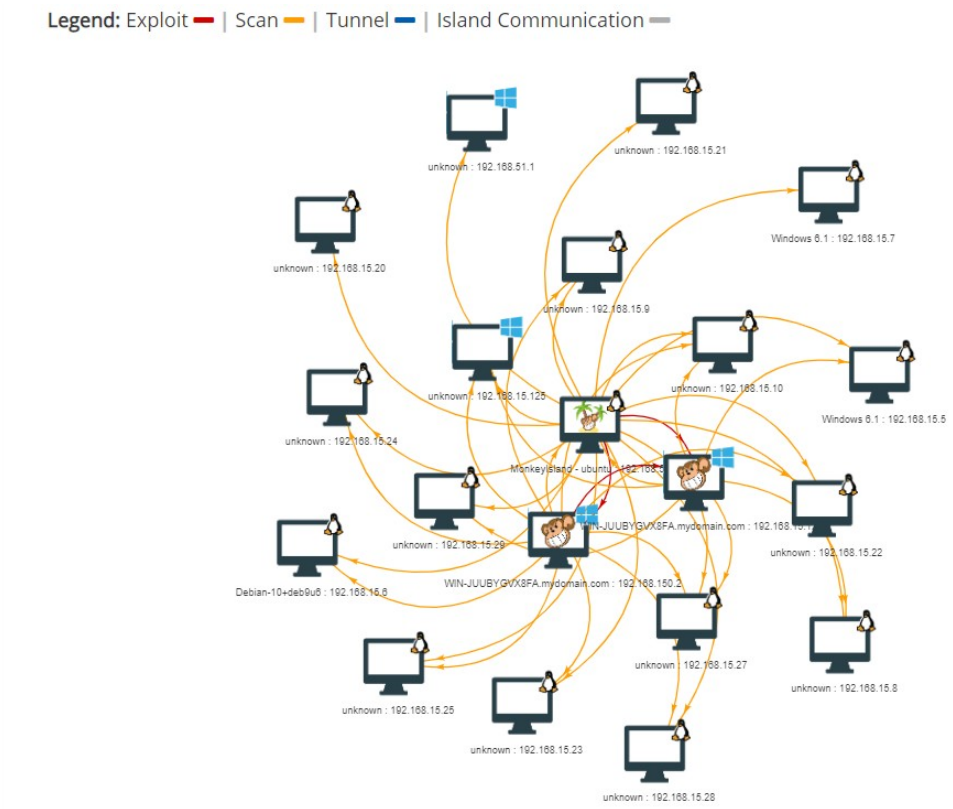


Figure 4.21: Infection Monkey interaction map.

The report of the attack shows that the machines present on the network and their services were all discovered; the backdoor instalment was successfuly performed and communication with the Command and Control server has been established; two machines have been exploited through SMB service by brute-force of the credentials, logs related to Infection Monkey's actions have been deleted and that Mimikatz was successfully installed and used inside the exploited systems.

T1021 Remote services

Monkey successfully logged into remote services on the network.

Machine	Service	
192.168.15.125	SMB	
192.168.15.125	WMI (Windo... Manag... Instru...	
192.168.15.125	MSSQL	
192.168.15.6	SSH	
192.168.15.11	SMB	Administrator ; Plaintext password : Adm*****
192.168.15.5	SSH	
192.168.15.7	SSH	
192.168.15.12	SMB	Administrator ; Plaintext password : Adm*****
192.168.15.5	SSH	
192.168.15.125	SMB	
192.168.15.5	SSH	
192.168.15.125	WMI (Windo... Manag... Instru...	
192.168.51.1	SMB	
192.168.51.1	WMI (Windo... Manag... Instru...	
192.168.15.6	SSH	
192.168.15.125	SMB	
192.168.15.125	WMI (Windo... Manag... Instru...	
192.168.15.125	MSSQL	
192.168.51.1	MSSQL	
192.168.15.6	SSH	

Figure 4.22: Services discovered by Infection Monkey.

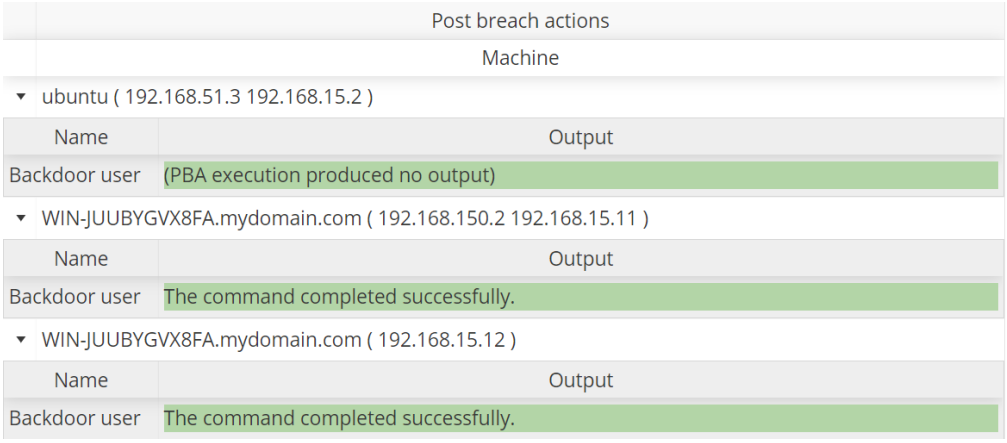


Figure 4.23: Backdoor instalment in the exploited machines.

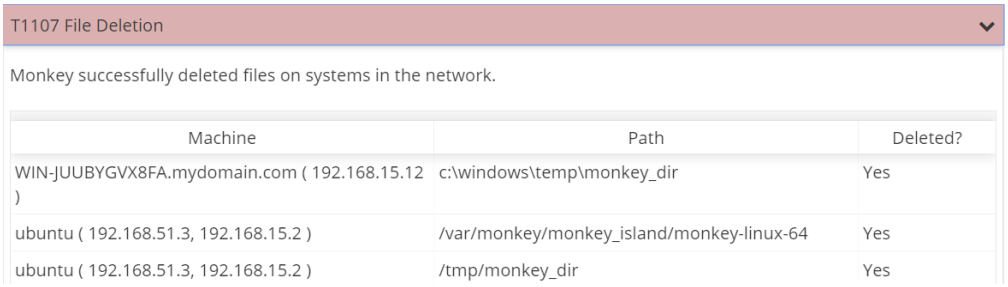


Figure 4.24: Deletion of the logs involving attacker's actions.

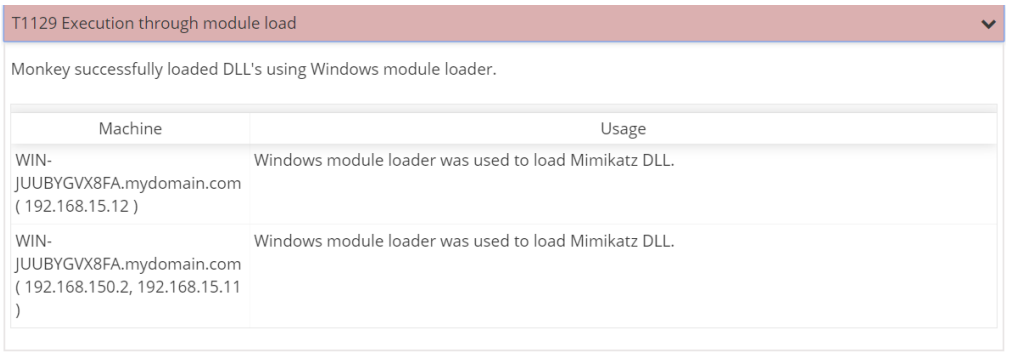


Figure 4.25: Utilities instalment inside the victim's machines.

Dejavu has been able to collect 17 logs of events related to the interaction of the Infection Monkey with the decoys deployed inside the network, and the overall malicious activities are resumed in the following graph.

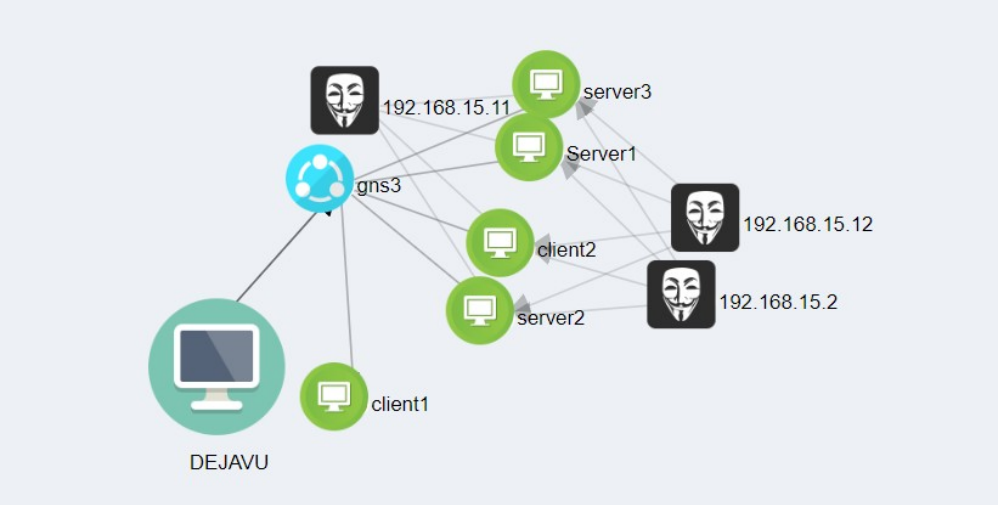


Figure 4.26: Attacker's actions from the Dejavu platform's point of view.

The collected logs are now used to fill the Observables table in order to understand the depth of information obtained by the deception platform.

Observables	Real Score	Perceived Score
Attack origination points	7	4 - 7
Victims involved in the attack	8	4 (Generic servers)
Risk tolerance	4	1
Timeliness	7	7
Skills and methods	4	1
Actions	6	3
Objectives	3	2
Resources	medium	low
Knowledge source	Undefined	Unknown

Since the attack starts within the network and the actions involved cannot be mislead with accidental causes, the "attack origination points" observable has to be considered of medium risk. The collected logs indicate that the attacker exploited general-purpose Windows servers to continue the attack within the network, which indicate a medium risk in the "victims involved in the attack" observable, but since there is little possibility of customization of the Dejavu servers, it is not possible to determine exactly the real target of attackers. The "risk tolerance" observable indicates the effort made by the attacker to remain undetected inside the network depending on the corruption of the logs in the exploited machines; since the attack was programmed to delete the logs involved in the attack, the seriousness of this observable is categorized as medium, but from Dejavu's perspective, no log corruption has been detected. About the "timeliness" observable, the logs show that the attack is conducted in a brief amount of time and it involves multiple Windows systems, which could suggest that the attacker was interested in some specific

information regarding the exploited machines. The logs show that the skills and methods engaged in the attack are not tailored for the victim's environment, which leads to a low score on this observable: the Dejavu logs reported below show that the attacker only performed a TCP and an HTTP scan, tried to exploit the SMB service and attempted brute force login through SSH protocol.

Decoy Name	Network Location	Service Deployed	Event Type	Decoy IP	Attacker IP	Timestamp
Server1	gns3	TCP	TCP Request: 192.168.15.12:51388 --> 192.168.15.5:2222	192.168.15.5	192.168.15.12	2019-10-29 03:07:05
Server1	gns3	TCP	TCP Request: 192.168.15.12:51390 --> 192.168.15.5:7001	192.168.15.5	192.168.15.12	2019-10-29 03:07:05
Server1	gns3	TCP	TCP Request: 192.168.15.12:51391 --> 192.168.15.5:445	192.168.15.5	192.168.15.12	2019-10-29 03:07:05
Server1	gns3	TCP	TCP Request: 192.168.15.12:51387 --> 192.168.15.5:8080	192.168.15.5	192.168.15.12	2019-10-29 03:07:02
Server1	gns3	TCP	TCP Request: 192.168.15.12:51386 --> 192.168.15.5:22	192.168.15.5	192.168.15.12	2019-10-29 03:07:01
Server1	gns3	TCP	TCP Request: 192.168.15.12:51396 --> 192.168.15.5:135	192.168.15.5	192.168.15.12	2019-10-29 03:06:59
Server1	gns3	TCP	TCP Request: 192.168.15.12:51397 --> 192.168.15.5:9200	192.168.15.5	192.168.15.12	2019-10-29 03:06:59
Server1	gns3	TCP	TCP Request: 192.168.15.12:51396 --> 192.168.15.5:135	192.168.15.5	192.168.15.12	2019-10-29 03:06:59
Server1	gns3	TCP	TCP Request: 192.168.15.12:51397 --> 192.168.15.5:9200	192.168.15.5	192.168.15.12	2019-10-29 03:06:59
Server1	gns3	TCP	TCP Request: 192.168.15.12:51396 --> 192.168.15.5:135	192.168.15.5	192.168.15.12	2019-10-29 03:06:58

Figure 4.27: TCP open ports discovery.

Decoy Name	Network Location	Service Deployed	Event Type	Decoy IP	Attacker IP	Timestamp
Server1	gns3	TCP	TCP Request: 192.168.15.12:51432 --> 192.168.15.5:8008	192.168.15.5	192.168.15.12	2019-10-29 03:16:39
Server1	gns3	TCP	TCP Request: 192.168.15.12:51430 --> 192.168.15.5:443	192.168.15.5	192.168.15.12	2019-10-29 03:16:39
Server1	gns3	TCP	TCP Request: 192.168.15.12:51432 --> 192.168.15.5:8008	192.168.15.5	192.168.15.12	2019-10-29 03:16:36
Server1	gns3	TCP	TCP Request: 192.168.15.12:51432 --> 192.168.15.5:8008	192.168.15.5	192.168.15.12	2019-10-29 03:16:36
Server1	gns3	TCP	TCP Request: 192.168.15.12:51431 --> 192.168.15.5:8008	192.168.15.5	192.168.15.12	2019-10-29 03:16:36
Server1	gns3	TCP	TCP Request: 192.168.15.12:51431 --> 192.168.15.5:8008	192.168.15.5	192.168.15.12	2019-10-29 03:16:36
Server1	gns3	TCP	TCP Request: 192.168.15.12:51430 --> 192.168.15.5:443	192.168.15.5	192.168.15.12	2019-10-29 03:16:36
Server1	gns3	TCP	TCP Request: 192.168.15.12:51430 --> 192.168.15.5:443	192.168.15.5	192.168.15.12	2019-10-29 03:16:36
Server1	gns3	TCP	TCP Request: 192.168.15.12:51431 --> 192.168.15.5:8008	192.168.15.5	192.168.15.12	2019-10-29 03:16:36
Server1	gns3	TCP	TCP Request: 192.168.15.12:51429 --> 192.168.15.5:443	192.168.15.5	192.168.15.12	2019-10-29 03:09:39

Figure 4.28: HTTP ports discovery.

Decoy Name	Network Location	Service Deployed	Event Type	Decoy IP	Attacker IP	Timestamp
Server1	gns3	SSH	SSH Connection	192.168.15.5	192.168.15.12	2019-10-29 03:07:52
Server1	gns3	SSH	SSH Connection	192.168.15.5	192.168.15.12	2019-10-29 03:07:51
Server1	gns3	SSH	Failed Authentication for username "Administrator " using password "Admin123"	192.168.15.5	192.168.15.12	2019-10-29 03:07:51
Server1	gns3	SSH	Failed Authentication for username "Administrator " using password "1234"	192.168.15.5	192.168.15.12	2019-10-29 03:07:51
Server1	gns3	SSH	SSH Connection	192.168.15.5	192.168.15.12	2019-10-29 03:07:51
Server1	gns3	SSH	Failed Authentication for username "Administrator " using password "password"	192.168.15.5	192.168.15.12	2019-10-29 03:07:51
Server1	gns3	SSH	SSH Connection	192.168.15.5	192.168.15.12	2019-10-29 03:07:51
Server1	gns3	SSH	Failed Authentication for username "Administrator " using password "12345678"	192.168.15.5	192.168.15.12	2019-10-29 03:07:51
Server1	gns3	SSH	SSH Connection	192.168.15.5	192.168.15.12	2019-10-29 03:07:51
Server1	gns3	SSH	Failed Authentication for username "root " using password "Admin123"	192.168.15.5	192.168.15.12	2019-10-29 03:07:51

Figure 4.29: SSH login attempts.

The repetitive pattern of the attack against the decoys (once exploited, the internal machines will attempt to perform the same actions of the source of the attack) could suggest that the attack is a worm that spreads itself throughout the network, but the seriousness of the attack strongly depends on the actions performed on the compromised systems, which are out of the sight of the deception

platform (e.g. there is no possibility for Dejavu to detect the backdoor instalment since the deployed servers are low-interaction decoys which allow no access inside the machines). As for the previous observables, also the "objectives" show a gap between the actions performed inside the exploited systems and the logs collected by the Dejavu platform: the logs show that the attack targeted and exploited the servers inside the network, but in order to obtain real knowledge about the seriousness of the attack, the real exploited servers need to be investigated. The "resources" and "knowledge source" fields are filled taking into account all the considerations made for the previous observables.
In conclusion, it can be stated that the Dejavu Framework can be a useful tool to distribute low and medium-interaction decoys across the network, but on the other hand it presents some limits: it is not possible to customize the decoy images which could lead to evident incongruities between the production environment and the fake decoys; the absence of high interaction decoys leads to a superficial knowledge about the attacker's actions that could lead to misleading conclusions.

4.4 Commercial solutions.

This thesis has followed the Proof of Concept conducted by Intesa Sanpaolo of two commercial Distrbuted Deception Platforms: here are reported the main features of the platforms and the overall evaluation of the two solutions.

4.4.1 Commercial Deception Platform #1

Characteristics

The first platform that has been tested presents a similar architecture with respect to Dejavu Deception Framework described in section 4.3, since it is composed by a centralized console connected to one or many sensors deployed in the subnets which need to be covered by the deception strategy. The platform claims to provide the following protections: **Ransomware detection** through the use of bait files which, if encrypted, generate an alarm to the console; **data breach discovery** through the deployment of honeytoken credentials placed in the DMZ of the enterprise which, if used, determine the presence of intruders; **internal network intrusion**, by the deployment of client and server honeypots alongside with honeytokens; **data exfiltration**, through the use of fake sensitive data which if manipulated triggers an alert to the central console. The server decoys can be deployed as low-interaction, medium-interaction or high-interaction honeypots: low-interaction honeypots are triggered only by ICMP traffic, medium-interaction honeypots emulate several services but without allowing access to the attackers, high-interaction honeypots allow remote authentication and access in order to monitor and to gain knowledge about attacker's actions. The decoys that can be deployed comprise not only client and server machines, but also IoT devices, routers, printers and several other network components; the platforms provides also a recognition feature of the real devices in the network and suggests the choice of deceptive elements that may fit better to the analysed environment, and allows to upload custom images to be deployed within the network in order to create a believable and fitting deception scenario. This deception solution provides an uploading section which can be useful not only to customize the decoys and the honeytokens, but also to upload network configurations such as IP whitelists, IP ranges of the deployed decoys, hostname mapping etc.
The monitoring interface classifies the seriousness of the incident and individuates to which stage of the attacking chain does it belong: recon, authentication, lateral movement, data access, exfiltration, high interaction; moreover, incidents are grouped depending on the type of activity and event correlation.

Evaluation

Unlike the previously tested platforms, for both the commercial solutions it was not possible to perform the simulation of the Advanced Persistent Threat by the use of Infection Monkey breach and attack platform, but the actions that have been performed are the same as the ones described in section 3.3.
A first scan of the decoys is executed with Nmap: the platform categorizes the

scan as Recon attempt, and indicates the severity of the attack as Medium since this action was performed in other decoys in the same time interval. The platform generates two kind of logs from this event, separating the ICMP frames used to check if the host is up from the TCP packets generated by Nmap to scan the ports of the decoys; here is reported the capture of the event performed with Wireshark.

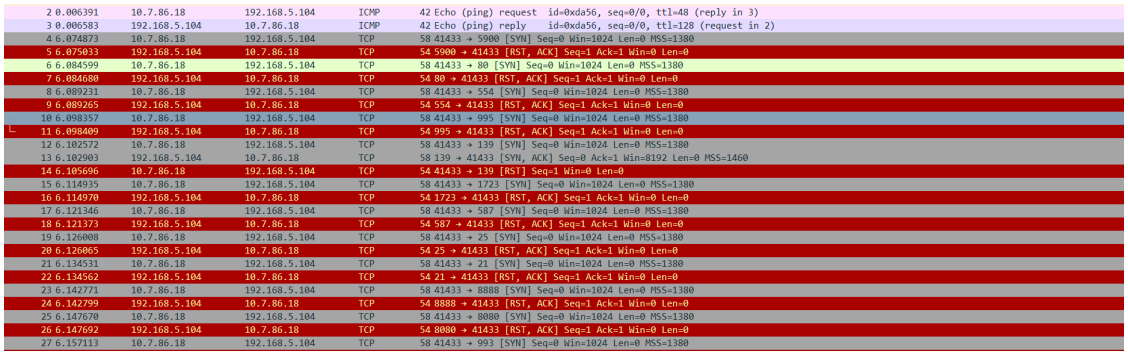


Figure 4.30: Capture of the scan performed with Nmap.

Through the Nmap scan the attacker has discovered that one server hosts the SSH service (port 22), a second server hosts Samba (through port 445) and a third server hosts a web-configuration page of an IoT camera (port 80). The servers hosting SSH and the web-page are low-interaction decoys which do not allow access, while the server providing the Samba service is an high-interaction honeypot which monitors the commands and actions performed by the intruders once they managed to gain access. The low-interaction honeypot's detection capabilities have been tested by attempting to login into the SSH decoy and with simple HTTP requests to IoT camera decoy: in both cases, the severity of the alert is reported as Medium, and the monitoring interface displays which credentials have been used. The SSH and the web-application honeypots differ in the categorization of the intruder's attempts: the web-application categorizes the opening of the IoT camera web page as Recon activity, the SSH server indicates the login efforts as Compromised credential and Lateral movement risk.

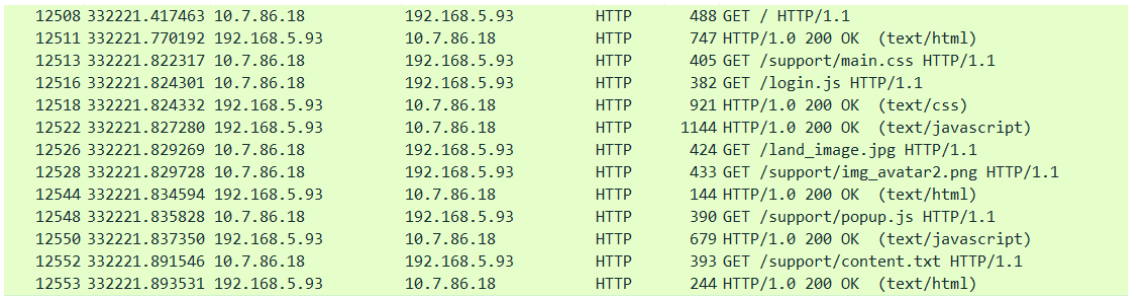


Figure 4.31: Capture of the HTTP interaction with IoT Camera webpage.

The SSH communication has been captured with Wireshark but the attempted credentials are displayed only in the monitoring interface of the platform since the

established communication between the attacker and the decoy is ciphered and the login credentials are then sent by the decoy to the main platform.

4021	347050.078890	10.7.86.18	192.168.5.15	SSHv2	82 Client: Protocol (SSH-2.0-PuTTY_Release_0.71)
4023	347054.265970	192.168.5.15	10.7.86.18	SSHv2	97 Server: Protocol (SSH-2.0-OpenSSH_7.2.p2 Ubuntu-2ubuntu2.10)
4024	347054.289785	10.7.86.18	192.168.5.15	SSHv2	1222 Client: Key Exchange Init
4026	347054.291400	192.168.5.15	10.7.86.18	SSHv2	638 Server: Key Exchange Init
4027	347054.298819	10.7.86.18	192.168.5.15	SSHv2	134 Client: Elliptic Curve Diffie-Hellman Key Exchange Init
4028	347054.308587	192.168.5.15	10.7.86.18	SSHv2	710 Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys
4030	347059.191300	10.7.86.18	192.168.5.15	SSHv2	134 Client: New Keys, Encrypted packet (len=64)
4031	347059.192712	192.168.5.15	10.7.86.18	SSHv2	118 Server: Encrypted packet (len=64)
4032	347059.196423	10.7.86.18	192.168.5.15	SSHv2	134 Client: Encrypted packet (len=80)
4033	347059.198358	192.168.5.15	10.7.86.18	SSHv2	198 Server: Encrypted packet (len=144)
4041	347666.271221	10.7.86.18	192.168.5.15	SSHv2	82 Client: Protocol (SSH-2.0-PuTTY_Release_0.71)
4043	347666.550154	192.168.5.15	10.7.86.18	SSHv2	97 Server: Protocol (SSH-2.0-OpenSSH_7.2.p2 Ubuntu-2ubuntu2.10)
4044	347666.555970	10.7.86.18	192.168.5.15	SSHv2	1222 Client: Key Exchange Init
4046	347666.557426	192.168.5.15	10.7.86.18	SSHv2	638 Server: Key Exchange Init
4047	347666.564800	10.7.86.18	192.168.5.15	SSHv2	134 Client: Elliptic Curve Diffie-Hellman Key Exchange Init
4048	347666.571422	192.168.5.15	10.7.86.18	SSHv2	710 Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys
4049	347666.582652	10.7.86.18	192.168.5.15	SSHv2	134 Client: New Keys, Encrypted packet (len=64)
4050	347666.583758	192.168.5.15	10.7.86.18	SSHv2	118 Server: Encrypted packet (len=64)
4051	347666.587213	10.7.86.18	192.168.5.15	SSHv2	134 Client: Encrypted packet (len=80)
4052	347666.589230	192.168.5.15	10.7.86.18	SSHv2	198 Server: Encrypted packet (len=144)
4054	347670.395671	10.7.86.18	192.168.5.15	SSHv2	326 Client: Encrypted packet (len=272)
4056	347671.400197	192.168.5.15	10.7.86.18	SSHv2	134 Server: Encrypted packet (len=80)

Figure 4.32: Capture of the SSH communication between the attacker and the decoy.

Then the Samba file sharing decoy server is tested: through the correct credentials the attacker is able to navigate inside the server through GUI, all the visited folders and files are registered by the decoy and displayed by the monitoring interface, alongside with the exploited credentials to obtain access inside the server.
The figure below shows the TCP three-way handshake between the attacker and the decoy, then the attacker sends the request to access to the Samba service, but fails in the authentication phase:

4033	337417.994593	10.7.86.18	192.168.5.17	TCP	66 49962 → 445 [SYN] Seq=0 Win=64240 Len=0 MSS=1380 WS=256 SACK_PERM=1
4034	337417.995154	192.168.5.17	10.7.86.18	TCP	66 445 → 49962 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
4035	337417.997763	10.7.86.18	192.168.5.17	TCP	54 49962 → 445 [ACK] Seq=1 Ack=1 Win=66048 Len=0
4036	337417.998253	10.7.86.18	192.168.5.17	SMB2	232 Negotiate Protocol Request
4037	337418.053526	192.168.5.17	10.7.86.18	TCP	54 445 → 49962 [ACK] Seq=1 Ack=179 Win=131072 Len=0
4038	337418.225555	192.168.5.17	10.7.86.18	SMB2	228 Negotiate Protocol Response
4039	337418.229497	10.7.86.18	192.168.5.17	SMB2	220 Session Setup Request, NTLMSSP_NEGOTIATE
4040	337418.231614	192.168.5.17	10.7.86.18	SMB2	401 Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHALLENGE
4041	337418.235430	10.7.86.18	192.168.5.17	SMB2	681 Session Setup Request, NTLMSSP_AUTH, User: SPIMI\U087734
4042	337418.237555	192.168.5.17	10.7.86.18	SMB2	131 Session Setup Response, Error: STATUS_LOGON_FAILURE
4043	337418.241079	10.7.86.18	192.168.5.17	TCP	54 49962 → 445 [RST, ACK] Seq=972 Ack=599 Win=0 Len=0

Figure 4.33: Capture of the unsuccessful login attempt into SMB service.

When the correct credentials are provided (`sede\it-usr` in the capture below), the attacker is able to access successfuly to SMB:

4055	337462.975799	10.7.86.18	192.168.5.17	TCP	66 53079 → 445 [SYN] Seq=0 Win=64240 Len=0 MSS=1380 WS=256 SACK_PERM=1
4056	337462.976759	192.168.5.17	10.7.86.18	TCP	66 445 → 53079 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
4057	337462.980709	10.7.86.18	192.168.5.17	TCP	54 53079 → 445 [ACK] Seq=1 Ack=1 Win=66048 Len=0
4058	337462.981186	10.7.86.18	192.168.5.17	SMB	213 Negotiate Protocol Request
4059	337463.037864	192.168.5.17	10.7.86.18	TCP	54 445 → 53079 [ACK] Seq=1 Ack=160 Win=131072 Len=0
4060	337463.253422	192.168.5.17	10.7.86.18	SMB2	228 Negotiate Protocol Response
4061	337463.256698	10.7.86.18	192.168.5.17	SMB2	232 Negotiate Protocol Request
4062	337463.258762	192.168.5.17	10.7.86.18	SMB2	228 Negotiate Protocol Response
4063	337463.302136	10.7.86.18	192.168.5.17	TCP	54 53079 → 445 [ACK] Seq=338 Ack=349 Win=65792 Len=0
4064	337463.536019	10.7.86.18	192.168.5.17	SMB2	220 Session Setup Request, NTLMSSP_NEGOTIATE
4065	337463.538075	192.168.5.17	10.7.86.18	SMB2	401 Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHALLENGE
4066	337463.541675	10.7.86.18	192.168.5.17	SMB2	677 Session Setup Request, NTLMSSP_AUTH, User: sede\it-usr
4067	337463.544198	192.168.5.17	10.7.86.18	SMB2	159 Session Setup Response
4068	337463.547805	10.7.86.18	192.168.5.17	SMB2	168 Tree Connect Request Tree: \\Share-FF-519\IPC$
4069	337463.549195	192.168.5.17	10.7.86.18	SMB2	138 Tree Connect Response

Figure 4.34: Capture of the successful SMB access.

With the provided captures it is possible to fill the observables' table of the

attack described in section 3.3.

Observables	Real Score	Perceived Score
Attack origination points	7	**4 - 7**
Victims involved in the attack	8	**8**
Risk tolerance	4	**4**
Timeliness	7	**7**
Skills and methods	4	**4**
Actions	6	**6**
Objectives	3	**3 - 4**
Resources	medium	**medium**
Knowledge source	Undefined	Unknown

As for the previous analysed cases, also this deception platform cannot clearly indicate the attacking origination point; since the platform allows the deployment of high interaction honeypots, it is possible to see that the attacker is seeking for sensitive data within the internal network, which leads to a correct evaluation of the second observable, but it needs to be stated that, in order to correctly individuate this attack characteristic, the decoys and the breadcrumbs need to be deployed in the internal network, since if they are deployed within the DMZ this attacking feature will not be recognized; differently from the previous cases, this platform is able to recognize that the attacker has performed log wiping, since the platform allows to deploy high-interaction decoys, capable of recording all the data manipulation performed by the attackers; the timeliness is correctly evaluated by the log-collection capabilities of the platform; the platform is able to recognize all the methods used in the attack since the actions have been monitored by the exploited decoys; considering the objectives observable, the correct analysis of the attack strongly depends on the bait files that are deployed in the design stage of the deception strategy: indeed this functionality needs to be carefully managed, since if the attackers have the goal to collect only some specific information they may be attracted by the luring files and therefore to seek for more sensitive information.

4.4.2 Commercial Deception Platform #2

Characteristics

The second commercial distributed deception platform tested in Intesa Sanpaolo environment presents a simpler infrastructure, since the platform is composed only by a physical or a virtual appliance, which corresponds to the central management interface; from the central console, it is possible to distribute the decoys over different subnets, and it is possible to distribute the honeytokens by running a script in the device of interest. The central console provides also an Artificial Intelligence functionality which allows to automate incident responses of the deception platform. The main use-cases for which the platform holds his value are: to provide **static deception** by deploying decoys and breadcrumbs all over the network; to provide a useful tool to the **Security Operation Centers** in order to analyse in deep ongoing suspicious activities; to allow an automate **incident response** technology, simple to manage in complex environments.
As for the first commercial platform tested above, also this one supports the deployment of Linux and Windows machines, including the possibility to upload custom images. The honeytokens are the mean to lead the attackers toward the decoys, and are divided in endpoint honeytokens and network honeytokens: example of endpoint honeytokens are credentials, browser cookies, SSH connection details, OpenVPN configuration files; network honeytokens are credentials which are periodically sent across the network channels, or NTLM traffic to tempt the attacker to perform the Pass-The-Hash attack. The main services supported by the decoys are: Git, MySQL, OpenVPN, SSH, Web-applications, FTP, RDP for Windows decoys, Intel AMT interface. The central interface allows to decide if the creation of decoys should be customized or if the security analyst wants to deploy the most common decoy architectures, such as Linux endpoints or servers running SSH and SMB, or VPN servers running VPN and SSH services, or internal website servers running web-application, SSH and MySQL.
Differently from the previously tested platforms, this one does not generate an alarm for every interaction between the attacker and the decoy: minor events such as port scans are documented but do not generate an alert. The events that cause an alert are: **port access**, indicating that a decoy has been probed; **interaction events** such as login attempts; **code execution**, which indicates that the attacker has executed a program already present in the decoy machine; **unsigned code execution**, indicating that the attacker is running a program that was not present into the probed decoy; **unrecognized machine**, when the decoys perform network scans and individuate machines which do not belong to the legitimate environment.

Evaluation

The first server under test is the one hosting the SSH and SMB services, to which two honeytoken credentials are bound, one for each service. Once the SSH request is performed and the correct honeytoken credentials are used, the decoy shows to the attackers that they have been able to access to the system; some simple commands like `ipconfig` are executed, and differently from the Cowrie honeypot anaylsed in

section 4.1, this platform displays the real output of the command:

```
C:\windows\system32>ssh administrator@192.168.174.2
administrator@192.168.174.2's password:
Last login: Wed Nov 13 15:57:54 2019 from 192.168.174.1
administrator@backupserver:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:b0:a8:f1
          inet addr:192.168.174.2  Bcast:192.168.174.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:feb0:a8f1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:28153 errors:2 dropped:0 overruns:0 frame:2
          TX packets:51030 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3127706 (3.1 MB)  TX bytes:120746656 (120.7 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:7846 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7846 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:61758646 (61.7 MB)  TX bytes:61758646 (61.7 MB)
```

Figure 4.35: Execution of ipconfig command through SSH: the decoy displays the actual network configuration.

The central console is able to display every interaction between the attacker and the decoy server, showing the commands executed by the attacker, including the log wiping attempt, and categorizing them as "code execution", while the opening and closing of the SSH communication are categorized as "Port access" and "SSH interaction". Then the SMB service is accessed through the honeytoken credentials, generating the "Share access" alert on the management console.

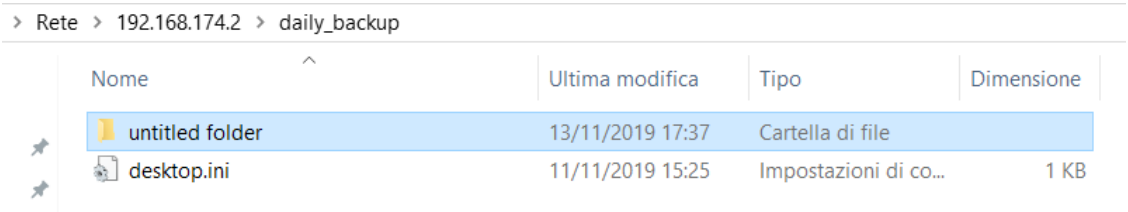


Figure 4.36: SMB decoy share folder access.

As for the SSH service, all the actions performed in the shared folder have been tracked by the main console. The last decoy that has been tested is the web-server application, a low-interaction honeypot (no possibility to bind login credentials to obtain access to the database) and it has been deployed without the Secure Socket Layer option; during the test, the page has been attempted to be reached with the ssl option, but destination resulted unreachable; reaching it through port 80, the site is accessible:

No.	Time	Source	Destination	Protocol	Length	Info
59	14.496331	192.168.174.1	192.168.174.3	TCP	66	56441 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
60	14.496671	192.168.174.1	192.168.174.3	TCP	66	56442 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
61	14.497710	192.168.174.3	192.168.174.1	TCP	60	443 → 56441 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
62	14.497710	192.168.174.3	192.168.174.1	TCP	60	443 → 56442 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
63	14.498951	192.168.174.1	192.168.174.3	TCP	66	56443 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
64	14.500342	192.168.174.3	192.168.174.1	TCP	66	80 → 56443 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
65	14.500494	192.168.174.1	192.168.174.3	TCP	54	56443 → 80 [ACK] Seq=1 Ack=1 Win=131328 Len=0

Figure 4.37: Capture of the https and http attempts to establish a TCP connection with the web-site.

After that the connection was correctly established, to the attacker is prompted the `PhpMyAdmin` login page, without the possibility to gain access:

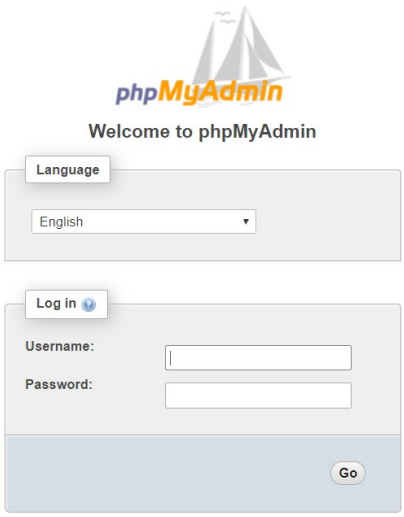


Figure 4.38: PhpMyAdmin login interface.

From the platform management monitoring interface, the attempt to reach the web-site through port 443 is categorized as "Port access" alert, when the attacker interacts with the web interface through port 80 the alert is of "HTTP request" type. The last action that is performed is the instalment of the backdoor described in section 6.1: this action was categorized by the deception platform as "code execution".
With the data collected above is now possible to fill and comment the Observables table, following the parameters described in section 3.2:

Observables	Real Score	Perceived Score
Attack origination points	7	**7**
Victims involved in the attack	8	**8**
Risk tolerance	4	**4**
Timeliness	7	**7**
Skills and methods	4	**4**
Actions	6	**6**
Objectives	3	**3**
Resources	medium	**medium**
Knowledge source	Undefined	Unknown

The platform is the first one that implements custom forensic tools to be executed in the attacking machine in order to understand the origin of the attack; the final target of the attackers is evident by the actions performed inside the decoys, since they were searching for sensitive data within the network; as stated above, the SSH decoy was able to intercept the log wiping attempt, allowing the correct score about the third observable; the Timeliness observable is evaluated correctly since the attack is meant to be executed in a narrow time window, but in general the accuracy of this observable strongly depends on the frequency of interaction by the attacker with the decoys; the Skills and methods observable is properly filled by the high-interaction capabilities of the decoys, as well as the Actions observable; about the objectives observable, since no stand-alone bait data was deployed, the attackers were not mislead by additional (even if false) data, which lead to the pursuit of the original goal without looking for other information about the system, allowing the high interaction decoys to properly capture all the performed actions, including the backdoor instalment; the resources exploited by the attackers are evaluated by taking into account all the observables above.

Chapter 5

Conclusions

The scope of the testing section of this thesis was to evaluate if the emerging Distributed Deception Platforms improve the detection of APT's tools, tactics, techniques and purposes with respect to the deployment of traditional deception technologies. The results obtained in section 4 allow to make a comparison between the deployment of a single medium-interaction honeypot, a central console which handles the deployment of already existing open-source honeypots, an open source and two commercial distributed deception platforms. The filled observables' tables suggest that the implementation of a medium-interaction honeypot leads to better results than the open-source distributed deception platform Dejavu, due to the low customization of the honeypots implemented by the distributed platform; the Modern Honey Network implements no virtualization capabilities but achieves better results than the Dejavu platform since it offers a reasonable possibility of choosing the proper honeypots to implement in the network, but on the other hand it cannot be ignored the higher difficulty in the analysis of the logs collected by the deployed decoys; the commercial solutions led to the best results in terms of depth of information collected, ease of management and deployment of the decoys and customization. In particular, the first solution allowed an easier deployment of a large number of decoys (useful in large networks), but the deployment of the breadcrumbs, bait files and honeytokens has to be performed manually on every machine, which could lead to scarce or bad usage of these features; on the other hand, the commercial platform #2 provided an easy way to link the breadcrumbs and the honeytokens to the desired decoys, but the creation of new decoys requires more time than the commercial platform #1, which could become significantly challenging in the deployment over large networks.
In the evaluation of distributed deception platforms must be considered that, in order to obtain comparable results, this research has been conducted with a repeatable attacking pattern that tried to mimic the characteristics of an Advanced Persistent Threat, but in the real scenario DDPs could represent an even more effective solution against attackers; on the other hand, they still require high carefulness in the designing phases of the deception strategy, since a wrong implementation could bring vulnerable elements inside the network easily exploitable by attackers.
Focusing on the management side, it is understandable why the open-source deception solutions are rarely adopted in enterprise environments, since companies look

for "plug-and-play" products, easy to handle and with technical support in case of faults; instead open-source solutions were complicated to deploy, often due to projects not updated, and even the log analysis was much easier in the distributed deception platform implementing a monitoring interface.
As explained above, this thesis' focus is limited on understanding the "research" value of the distributed deception paradigm, which is meant to be used in production environments; in order to take a step further on the comprehension of the real value of distributed deception platforms, it is suggested a deeper analysis performed by the employment of red teams unaware about the presence of the deception solutions, in contrast with blue teams which should take the proper counter-measures by deception platforms to face ongoing attacks.

Chapter 6

Appendix A

6.1 Backdoor

The following code provides a simple TCP client-server communication in which the server is handled by the attacker and the client is the victim's machine. The server is supposed to send commands to be executed in the client machine, and the client machine has to send back the resulting outputs.

6.1.1 Client Architecture

```
#!/usr/bin/sh
""":"
exec python3 $0 ${1+"$@"}
"""

import socket
import subprocess
target_host="server address"
target_port=25332
client=socket.socket(socket.AF_INET,socket.SOCK_STREAM) #definition of
    the TCP communication
client.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1) #allows to
    immediately reuse the socket
client.connect((target_host,target_port))
while True:
    command=client.recv(1024)
    if command=='exit':
        client.close()
        break
    else:
       proc=subprocess.Popen(command,shell=True,stdout=subprocess.PIPE,stderr=subprocess.P
           #info of command
       output=proc.stdout.read()+proc.stderr.read()
       client.send(output)
```

6.1.2 Server Architecture

```
#!/usr/bin/env python
import socket
server=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
bind_ip="server address"
bind_port=23442 #random
server.bind((bind_ip,bind_port))
server.listen(1) #server listens only one connection
while True:
   conn, addr=server.accept()
   print('Connection with the client',addr)
   while True:
      command=input("Shell>>")
      if command=='exit':
        conn.send('exit')
        conn.close()
        break
   else:
        conn.send(command.encode())
        output=conn.recv(1024)
        print(output)
```

Bibliography

[1] Verizon. Verizon 2019 data breach report. https://enterprise.verizon.com/resources/reports/2019-data-breach-investigations-report.pdf, 2019.

[2] IBM Ponemon Institute. Cost of a data breach report. https://databreachcalculator.mybluemix.net/, 2019.

[3] Frank J Stech, Kristin E Heckman, and Blake E Strom. Integrating cyber-d&d into adversary modeling for active cyber defense. In *Cyber deception*, pages 1–22. Springer, 2016.

[4] Kristin E Heckman, Frank J Stech, Roshan K Thomas, Ben Schmoker, and Alexander W Tsow. *Cyber denial, deception and counter deception*. Springer, 2015.

[5] US Information Operations. Military deception. https://jfsc.ndu.edu/Portals/72/Documents/JC2IOS/Additional_Reading/1C3-JP_3-13-4_MILDEC.pdf, 2012.

[6] Colin Tankard. Advanced persistent threats and how to monitor and deter them. *Network security*, 2011(8):16–19, 2011.

[7] MANDIANT. M-trends: the advanced persistent threat. https://content.fireeye.com/m-trends/rpt-m-trends-2010, 2010.

[8] Eric M Hutchins, Michael J Cloppert, and Rohan M Amin. Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains. *Leading Issues in Information Warfare & Security Research*, 1(1):80, 2011.

[9] Mitre att&ck enterprise matrix. https://attack.mitre.org/matrices/enterprise/.

[10] Bill Cheswick. An evening with berferd in which a cracker is lured, endured, and studied. In *Proc. Winter USENIX Conference, San Francisco*, pages 20–24, 1992.

[11] Deception toolkit homepage. http://all.net/dtk/index.html.

[12] Know your enemy: Honeynets. http://old.honeynet.org/papers/honeynet/index.html, 2005.

[13] Gene H Kim and Eugene H Spafford. Experiences with tripwire: Using integrity checkers for intrusion detection. 1994.

[14] Fabien Pouget, Marc Dacier, and Hervé Debar. White paper: honeypot, honeynet, honeytoken: terminological issues. *Rapport technique EURECOM*, 1275, 2003.

[15] Lance Spitzner. *Honeypots: tracking hackers*, volume 1. Addison-Wesley Reading, 2003.

[16] Kostas G Anagnostakis, Stelios Sidiroglou, Periklis Akritidis, Konstantinos Xinidis, Evangelos Markatos, and Angelos D Keromytis. Detecting targeted attacks using shadow honeypots. 2005.

[17] Abdallah Ghourabi, Tarek Abbes, and Adel Bouhoula. Honeypot router for routing protocols protection. In *2009 Fourth International Conference on Risks and Security of Internet and Systems (CRiSIS 2009)*, pages 127–130. IEEE, 2009.

[18] John T Moy. *OSPF: anatomy of an Internet routing protocol*. Addison-Wesley Professional, 1998.

[19] Anoosha Prathapani, Lakshmi Santhanam, and Dharma P Agrawal. Intelligent honeypot agent for blackhole attack detection in wireless mesh networks. In *2009 IEEE 6th International Conference on Mobile Adhoc and Sensor Systems*, pages 753–758. IEEE, 2009.

[20] Lance Spitzner. Honeypots: Catching the insider threat. In *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, pages 170–179. IEEE, 2003.

[21] Neal Krawetz. Anti-honeypot technology. *IEEE Security & Privacy*, 2(1):76–79, 2004.

[22] Maximillian Dornseif, Thorsten Holz, and Christian N Klein. Nosebreak-attacking honeynets. In *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.*, pages 123–129. IEEE, 2004.

[23] S Mukkamala, K Yendrapalli, R Basnet, MK Shankarapani, and AH Sung. Detection of virtual environments and low interaction honeypots. In *2007 IEEE SMC Information Assurance and Security Workshop*, pages 92–98. IEEE, 2007.

[24] Henry Campbell Black, Bryan A Garner, Becky R McDaniel, David W Schultz, and West Publishing Company. *Black's law dictionary*, volume 196. West Group St. Paul, MN, 1999.

[25] Brian Scottberg, William Yurcik, and David Doss. Internet honeypots: Protection or entrapment? In *IEEE 2002 International Symposium on Technology and Society (ISTAS'02). Social Implications of Information and Communication Technology. Proceedings (Cat. No. 02CH37293)*, pages 387–391. IEEE, 2002.

[26] Bradley S Rubin and Donald Cheung. Computer security education and research: handle with care. *IEEE security & privacy*, 4(6):56–59, 2006.

[27] Honeypot: tra agente provocatore e privacy. https://www.ilnuovodiritto.it/2017/02/23/honeypot-tra-agente-provocatore-e-privacy/, 2017.

[28] Meharouech Sourour, Bouhoula Adel, and Abbes Tarek. Ensuring security in depth based on heterogeneous network security technologies. *International Journal of Information Security*, 8(4):233–246, 2009.

[29] Honeypot architecture vs. deception technology. https://go.illusivenetworks.com/wp-honeypot-v.-deception-tech-lp?hsCtaTracking=a01a63e9-c2a9-4a7a-a70c-13ef840a0358%7C86bc81ae-decf-4352-bdf3-064696847616.

[30] Mohammed H Almeshekah and Eugene H Spafford. Cyber security deception. In *Cyber deception*, pages 23–50. Springer, 2016.
[31] Kristin E Heckman, Frank J Stech, Ben S Schmoker, and Roshan K Thomas. Denial and deception in cyber defense. *Computer*, 48(4):36–44, 2015.
[32] How a secret cyberwar program worked. https://archive.nytimes.com/www.nytimes.com/interactive/2012/06/01/world/middleeast/how-a-secret-cyberwar-program-worked.html.
[33] Introduction to cyber deception. https://www.ciosummits.com/Online_Assets_Cymmetria_Whitepaper_-_Introduction_to_Cyber_Deception.pdf.
[34] Brian M Bowen, Vasileios P Kemerlis, Pratap Prabhu, Angelos D Keromytis, and Salvatore J Stolfo. Automating the injection of believable decoys to detect snooping. In *Proceedings of the third ACM conference on Wireless network security*, pages 81–86. ACM, 2010.
[35] Brian M Bowen, Shlomo Hershkop, Angelos D Keromytis, and Salvatore J Stolfo. Baiting inside attackers using decoy documents. In *International Conference on Security and Privacy in Communication Systems*, pages 51–70. Springer, 2009.
[36] Ping Chen, Lieven Desmet, and Christophe Huygens. A study on advanced persistent threats. In *IFIP International Conference on Communications and Multimedia Security*, pages 63–72. Springer, 2014.
[37] Osint framework. https://osintframework.com/.
[38] Emilie Purvine, John R Johnson, and Chaomei Lo. A graph-based impact metric for mitigating lateral movement cyber attacks. In *Proceedings of the 2016 ACM Workshop on Automated Decision Making for Active Cyber Defense*, pages 45–52. ACM, 2016.
[39] Sean Bodmer, Max Kilger, Gregory Carpenter, and Jade Jones. *Reverse deception: organized cyber threat counter-exploitation*. McGraw Hill Professional, 2012.
[40] Guardicore infection monkey breach & attack platform. https://github.com/guardicore/monkey.
[41] Cowrie - ssh/telnet honeypot. https://github.com/cowrie/cowrie.
[42] Modern honey network. https://github.com/pwnlandia/mhn.
[43] Conpot - ics/scada honeypot. http://conpot.org/.
[44] Drupot - drupal honeypot. https://github.com/d1str0/drupot.
[45] Magenpot - magento honeypot. https://github.com/Creare/magento-honeypot.
[46] Wordpot - a wordpress honeypot. https://github.com/gbrindisi/wordpot.
[47] Shockpot. https://github.com/pwnlandia/shockpot.
[48] P0f. http://lcamtuf.coredump.cx/p0f3/.
[49] Suricata - open source ids/ips/nsm engine. https://suricata-ids.org/.
[50] Glastopf - web application honeypot. https://github.com/mushorg/glastopf.
[51] Elastichoney - a simple elasticsearch honeypot. https://github.com/jordan-wright/elastichoney.
[52] Amun honeypot. https://github.com/zeroq/amun.

[53] Snort nids. https://www.snort.org/.
[54] Dionaea - a malware honeypot. https://github.com/DinoTools/dionaea.
[55] Shockpot sinkhole. https://github.com/pwnlandia/shockpot/commit/59f037e58785ab5973e1797f09efbbbdefcda602.
[56] thc-hydra. https://github.com/vanhauser-thc/thc-hydra.
[57] Dejavu : an opensource deception framework. https://github.com/bhdresh/Dejavu.