Deception Honeypots: Deep Intelligence

Nil Ortiz Rabella

Resum—En un món on Internet és una eina fonamental pel desenvolupament de les empreses, que volen créixer i establir-se en el mercat econòmic global, la seguretat dels seus sistemes informàtics es converteix en una necessitat. La constant evolució de les tecnologies, promou un ambient en el qual els mètodes que es fan servir per atacar els sistemes informàtics, evolucionen encara més ràpid que les pròpies tecnologies, crean un estat on és pràcticament impossible garantir la integritat i la seguretat completa dels sistemes. La majoria dels mètodes actuals de seguretat, tenen com a objectiu la prevenció o detecció. Per aquest motiu aquest treball implementa els honeypots d'alta interacció, amb els quals podem implementar un factor proactiu en la nostre seguretat, atraient als atacants a un espai controlat, per aprendre els seus mètodes i fer servir aquesta informació per protegir els sistemes reals. En aquest article, es proposa el desenvolupament d'un honeypot d'alta interacció i la seva implementació, en una xarxa similar al entorn de producció d'una empresa per enganyar possibles atacants.

Paraules clau—Honeypot, Alta interacció, Cyber-seguretat, Deception, Intel·ligència d'amenaçes, Amenaça dirigida

Abstract—In a world where Internet is a key element for the development of any company, that wants to rise and establish in the economic global market, the security of the computer systems used in the company's becomes an imperious need. The constant evolution of technology, provides an environment where the methods used to attack the computer systems evolve even faster than the technologies itself, creating a state where it is practically impossible to assure the integrity and complete security of the systems. Most actual security methods and policies, act only as a prevention or detection solution. Therefore in this paper we implement high interaction honeypots, which allow a new proactive factor in our security, to attract the attackers into a controlled environment, where we can learn their methods and use that information to protect the real systems. In this paper we will propose the development of a high interaction honeypot, and its implementation in a network, which we could find in a real bussines environment.

Index Terms—Honeypot, High interaction, Cyber-security, Deception, Threat intelligence, Targeted threat

1 Introduction

NOWADAYS Internet has taken part not only of our personal life, but it is a core piece in the scheme of any company that wants to succeed in a global market, and for every successful product, there is someone trying to get advantage illegally from it.

Cyber-crime is an ongoing and critical issue that raises every day, evolving even faster than security, and no longer can we think of a computer system in terms of completely protected, with preventive and detection policies as our only security, it is just a matter of time since our systems gets compromised, and maybe by the time we find the breach it will be too late to prevent losses.

In the last years, a new approach to security has been developed, proactive security, instead of trying to keep the attackers from breaching in the system, create a controlled environment where we can track all the actions done by any attacker to learn their methods and techniques to improve the security of our real systems at the same pace as the attacks evolve.

The honeypots are tools designed to detect, deflect and counteract unauthorized use of a computer system, they can be classified in two groups, low interaction honeypots and high interaction honeypots, the core difference between the two groups is that the low interaction honeypots don't provide real services, they only simulate them, therefore the information that can be obtained through this type of honeypot has quite low depth or value, where high interaction honeypots have real services with whom the attackers can fully interact, and provide information about each and action done by the attacker that can be used to prepare our security against it on the real systems.

In order to enhance the security of a company, we can place a high interaction honeypot in the same network as the real production systems that will attract potential attackers and waste their time using deception techniques.

- *Contact E-mail: nil.ortiz@e-campus.uab.cat*
- *Specialization: Information Technologies.*
- *Tutored by:*
  Ramon Vicens (Blueliv)
  Guillermo Navarro Arribas (DEIC)
- *Curs 2016/17*

There are four styles of deception, which combined create the so called Deception Stack. The deception stack consists of a set of tools and responses that operate at different layers the attacker may interact with, the network, endpoint, application and data layers. In order for the deception techniques to be believable, they have to be deployed across the whole Deception Stack, it is important to notice that further the attack carries on; the harder it becomes to maintain the deception. [1]

A graphic of the deception stack can be found on the section A1 of the annex.

1.1 Objectives

This work is done in cooperation with Blueliv, a targeted cyber-threat intelligence provider for enterprises, aiming to provide new sources of threat intelligence and counter-intelligence data.

The main objective of this project is to develop a high interaction honeypot, which can be deployed in to a fully functional business network to retrieve intelligence about the attacks targeted to the network company.

The specific goals to accomplish the main objective are the following:

1. Find or develop a series of tools to monitor the honeypots at both user and kernel level.
2. Harden the system to keep the attackers from fully compromising the system.
3. Implement a logging system that can produce personalized outputs from the system calls, the user's activities and the network traffic.
4. Install and modify a series of services and applications to be exploited.
5. Test all services and functionalities.
6. Develop deception techniques to retrieve deeper intelligence on the attacks.
7. Develop installation scripts to automate the deployment process.

While objectives two, three and five are the most important for the project itself, spending more time on the objectives one and seven will provide a more solid solution. Objective six could be split for each layer of the deception stack optionaly, but since we will be focusing on the data layer, there is no need.

2 Methodology

2.1 Research

In order to gain a full view of the state of the art in the field of honeypots, both high and low interaction, an extensive research phase took place at the start of the project. The goal of the research phase was to understand the architechture and functionality of honeypots that could become the first stone of the project.

This phase concluded with the results that there was no available build-in technology suitable to our case, hence we would need to elaborate our own customize solution.

2.2 Requirements

In order to ease the design phase, a set of basic requirements for the high interaction honeypot that would be fundamental to the succeed status of the project were established.

2.3 Design

After having set the initial requirements of the project, the design phase started, trying to find tools that fitted our needs to register all activity done in the system and towards it, considering the requirements, a specific architecture for the system was planned and series of services and applications were selected to be tested.

2.4 Development

Once the design of the system was complete and all requirements and functionalities were established, they were developed combining open-source tools with tailored scripts for each situation.

2.5 Test

After the system architechture was developed and functional, we begin to test to functionalities of the services and our capability to register the activity done by a user, this test were conducted on a standalone honeypot, first simulating attacks from our own platform and then setting the honeypot open for real attackers to interact with.

2.6 Deployment

Once all functionalities of the honeypot were tested, a base image of the system was developed and deployed to a series of honeypots to create a proof of concept, each system was then loaded with tailored scripts to deploy the desired services.

A deception story was planified to interconnect the honeypots and mislead the attackers from on to another in order to acquire intelligence on their methods.

3 State of the art

Even if the project was aimed towards high interaction honeypots, to understand better its functionality, low

interaction honeypots were also part of the research process and considered to be part of the project initial state.

3.1 Honeypots

Honeypots are security resources that simulate and pretend to be services of a computing system, or the whole system, with vulnerabilities or weak points in order to be probed, attacked, compromised, used or accessed in any unauthorized way. [2]

Their objective is to gather all the possible information from these attacks or unauthorized accesses in order to study the methodology used from the attackers and prevent them in the future. Since they only simulate the environment and responses of these services, no sensitive information is compromised from the attacks.

3.1 Low interaction honeypots

Low interaction honeypots simulates services, networks or other aspects of a real computing system, usually they simulate a limited set of services and network protocols (mostly TCP and IP) that allow a limited interaction with the attacker in order for him to think at it is a real system.

The main advantages of low interaction honeypots it is that they are easy to deploy and maintain, even without deep knowledge in the field; with a basic configuration anyone can deploy a low interaction honeypot in their network. Since every service its simulated, this kind of honeypots can't compromise the real system behind it, because they only offer a limited set of interactions with the attacker, if he were able to surpass the honeypot using instructions out of its reach, he would discover that it is a honeypot instead of a real system but still couldn't do anything in the system behind it.

The fact that the interactions with the attacker are limited develops the main problem with these type of honeypots, they can only retrieve a limited amount of information about the attacks based on what interactions are designed to handle.

Considering everything stated before, it is clear that this kind of honeypots aren't useful to gather intelligence against targeted attacks, but their main use is to gather statistics on attack types and patterns. They can also be used as a NIDS (Network Intrusion Detection System) given the fact that the honeypot isn't a production system; any access attempts can be considered as an attempt to breach in the network.

3.2 High interaction honeypots

High interaction honeypots are real systems, with a real operating system, real services and real protocols. To disguise the honeypot in a network it is advised that the honeypot has the same operating system and the same services that the other systems in the network have.

Since it is a real system, the interactions with the attacker are also real and this allow us to gather all possible information about the attack, every command or file used, every I/O action, every bit of data transmitted through the network is recorded and encapsulated in a session for post forensic analysis. We can also modify the system in any way we need to make it look like the production systems, enabling the same services, creating similar file systems, using outdated former credentials, and since it is placed in the same network than the production systems but isn't actually one of them, any access will be categorized as an attack, eliminating the false positive alerts and providing a view of focused and targeted attacks against a network.

Given that the system is real, the operating system and the services may have vulnerabilities out of our knowledge that can be exploited and used against us, therefore the possibility of having a compromised honeypot becomes real, and so it has to be prepared and protected from any unwanted activity, it is the designer responsability to make sure that the attacker can't use the system for anything that isn't supposed to do, which leads to a high cost design, deploy and maintenance.

3.3 Honeynet

Honeynets [3] extend the concept of a single honeypot to a highly controlled network of honeypots. A honeynet is a specialized network architecture configured in a way to achieve data control, data capture and data collection. This architecture creates a highly controlled network, in which one can control and monitor all kind of systems and online activity. Honeypots are then placed within this network. A basic honeynet is composed by honeypots placed behind a transparent gateway – the honeywall. Acting as a transparent gateway the honeywall is undetectable by the attackers and serves its purpose of logging all network activity going in or out of the honeypots.

3.4 BeeSwarm

BeeSwarm [4] is a built-in solution for Linux to deploy a honeynet using low-interaction honeypots, its architechture its similar to the honeynet minus the honeywall, but it keeps the three main focus points of data control, data capture and data collection.

The minimum agents that compose BeeSwarm are a system with a BeeSwarm server who is responsible for deploying the rest of the agents and receiving the data gathered on them, it also provides a web interface to visualize the data retrieved from the honeypots and classify the attacks in two types, as credential reuse if the attacker manged to log in or brute force if the attacker didn't log in the service.

The other agents are the honeypot sensor and the honeypot drone. The sensor is the one with the simulated services, BeeSwarm provides simulation of 9 services and protocols (FTP, HTTP, HTTPS, POP3, POP3S, SMTP, SSH, Telnet, VNC) through Python scripts. The honeypot drone has the same base as the sensor, but doesn't simulate any services, it only generates traffic against the

honeypot sensors through dummy tests connections feeding them a set of pre-stablished credentials.

Theoretically an attacker could intercept the traffic between the drones and the sensors and then reuse the stolen credentials to attack the sensors, but usually the drone's traffic is used only to generate activity in the sensors, in order to make the sensors look like real systems with real activity to deceive anyone who is sniffing the network.

3.5 Sebek

Sebek [5] is a data capture tool. As with all data capture tools, the goal is to capture data that will allow us to accurately recreate the events on a honeypot.

Sebek has two components, the client and the server. The client captures data of a honeypot and exports it to the network where is collected by the server. The server collects the data from one of two possible sources: the first is a live packet capture from the network, the second is a packet capture archive stored as a tcpdump formatted file. Once the data is collected it is either uploaded into a relational data base or the keystroke logs are immediately extracted. The communications used by Sebek are UDP based and as such are connection-less and unreliable.

The client resides entirely in kernel space on the honeypot and, in the case of the Linux version, is implemented as a Loadable kernel Module (LKM). The client can record all data that a user accessed via the read() system call. This data is then exported to the server over the net in a manner that is difficult to detect from the honeypot running Sebek. The server then gathers the data from all of the reporting honeypots. Since there is a standard platform independent log format the server can collect from any honeypot independent of Operating System type.

Since Sebek register the activity of the honeypot accessing to the system calls of the operating system at kernel space, even if the attacker were to download and use trojaned binaries or shared libraries to bypass regular loggin methods, every action would still be recorded by Sebek, the only way to surpass it would be a hook on the system calls table at kernel space, which should take too much time for the attacker to do without being kicked from the system. It is a perfect tool to deploy in a high interaction honeypot.

3.6 Loadable kernel modules (Linux)

The most basic method of adding code or capabilities in the Linux kernel is through adding source files in the kernel source tree and recompile it, actually the configuration of the kernel consists mainly of choosing a set of files to include in the kernel. But there is a method to add code to the Linux kernel while its still running without recompiling it.

Loadable kernel modules (LKM) [6] are chunks of code that can be loaded in the kernel on the go, they usually provide functionalities for device drivers, file system drivers and system calls, but are not limited to those.

3.7 Linux security modules

The Linux Security Module (LSM) [7] framework provides a mechanism for various security checks to be hooked by new kernel extensions. The name "module" is a bit of a misleading since these extensions are not actually loadable kernel modules. Instead, they are selectable at build-time and can be overridden at boot-time.

The primary users of the LSM interface are Mandatory Access Control (MAC) extensions which provide a comprehensive security policy. LSM include SELinux, Smack, Tomoyo, and AppArmor. In addition to the larger MAC extensions, other extensions can be built using the LSM to provide specific changes to system operation when these tweaks are not available in the core functionality of Linux itself.

4 REQUIREMENTS

After understanding the nature of the high interaction honeypots we set a base of requirements for the design of the honeypot.

1. Operating system:
    a. Has to be customizable.
    b. Has to be easy to deploy.
2. System security:
    a. We must remain in control of the honeypot before, during and after the attacker breaches in even if he gets root permissions.
    b. The tools used to monitor the honeypot must be concealed from the attacker.
    c. Failed attempts to breach in the system must also be logged.
3. Services:
    a. They have to be customizable
    b. Provide a method to retrieve new or modified files.
4. Monitoring
    a. We have to be able to register all activity at both kernel and user level.
    b. Any changes done by the service in the file system must be logged.
    c. All actions done by the attacker must be logged.
    d. We must be able to retrieve any new or modified files by the attacker.
    e. The actions of a specific attacker must be registered in a specific session with a specific identifier.
5. Deploy:
    a. Customizable within the needs of the client.
    b. Easy and automate deployment.

5 DESIGN

In order to comply with the requirements, the following solution became the design of the high interaction honeypot. The names of certain tools or software cannot be revealed due to a confidentiality agreement with Blueliv.

For the operating system we chose Linux Ubuntu 4.7 since it is an open source solution, which provides extensive documentation to ease the process of customization and can easily be installed on any machine using virtual images.

In order to achieve the security requirements we considered using LSM, but were discarded because: LSM only involve Mandatory Access Control (MAC) through file system permission, LSM is compiled and enabled in the kernel, which means that the symbols are exported and every hook is accessible using root-kits, allowing the attacker to be create back-doors and be completely undetected, since LSM are included in the Linux kernel since release 2.6, advanced attackers would be aware of its presence.

Instead, we modified the Linux kernel using an already build solution as a base, that allowed us to track all system calls and a software that works upon the custom kernel like the LSM but using Role Base Access Control (RBAC) policies and allows control over all Linux capabilities, the RBAC functionality allow to create a new level of permission above the usual root user, that acts like a super-root, invisible to any attacker and only available with the appropiate credentials, using this features we can easily hide any process, parts of the file system and pretty much anything we need from the attacker.

To keep a register of the connection attempts, port scans and other sniffing network actions, we used an Intrusion detection System (IDS) software to examine the packets before and after they enter in the system, the IDS had been pre-loaded with some traffic rules created by the community and some rules developed by us.

On the initial phase of the design, we included the following services:

1. FTP
2. Samba
3. HTTP
4. SSH
5. VNC
6. OpenVPN

Later we decided to focus only on the FTP, Samba, HTTP and SSH services due to the lack of use and possible information that was retrievable of the remaining.

To enhance security, the FTP and Samba services were configured to prevent the users from accessing any other part of the file system that was not dedicated to the service, limitating the FTP users to their home path and the Samba users to a unique share.

HTTP was implemented using Apache2, enabling specific modules to enhance the security of Apache2 and retrieve more information from the HTTP requests; we also added the possibility to equip Apache2 with WordPress and phpMyAdmin depending on the needs of the client.

SSh was configured so only a list of specific users could gain access to the system using the standard username password log in option.

To achive the monitoring requirements a few options were considered, sebek seemed like a good option, but it needs a kernel 2.4 for Linux, which not only is highly outdated but also would conflict with our custom kernel previously mentioned, and does not have active support or a community behind it.

The next explored option was to make us of the logging facilities of Linux itself and the services along side with the tracking of system calls implemented in the custom kernel, which then would be parsed and sent to a centralized server to process the information. This solution was deeply tested using a server with Logstash, an open source, server-side data processing pipeline that ingests data from a multitude of sources simultaneously, transforms it, and then sends it to the storage application, to process the information, ElasticSearch, a distributed, RESTful search and analytics engine that stores data indexed in volumes, to store and sort it and Kibana, an open source data visualization plugin for Elasticsearch, to display the final information, but was discarded because it was very difficult to encapsulate the actions of a user in a session format across all services, we would need some sort of backup software to retrieve the new or modified files which added a new layer of complexity due to the possibility of I/O atomic operations.

Finally, we decided to use a software that reads the system calls of the kernel and saves them in a file that can later be processed using LUA scripts, the developers of the software provides a wide variety of scripts but we chose to develop our own scripts to extract the information that better suited our needs.

To improve visibility from the network traffic and the actions developed outside of the system, we used the IDS not only to emit alerts but also capture the network traffic in a pcap format.

To fulfil deployment requirements we created a set of bash scripts that install all the Linux packages, tools and services that we need in a modular fashion, therefore when a customer needs to deploy a honeypot we can execute a series of scripts on the machine to install the base of the honeypot and the functionalities selected by the customer without having to remove or leave unused

modules.

6 TEST

To test all capabilities and services of the honeypot, we deployed a standalone virtual machine with the previously mentioned set up, including the FTP, Samba, HTTP (Apache2) and SSH services. The system was being monitored with the kernel syslog and the services own loggin facilities which were sent to a centralized server using Filebeat. On that server an instance of Logstash received and parsed the logs using grok patterns and then the logs were stored on a ElasticSearch, this server also had an instance of Kibana to visualize the logs in a comprehensive way.

In order to test the logging mechanism's the FTP and Samba services were configured so they could be accessed anonymously. The Apache2 logging facilities were modified to acquire the request headers, the request body, the response headers and the response body. To monitor the activity on the SSH service we kept track of the attempts of connection using the auth.log provided by the operating system and once a intruder managed to get in the system, we monitored he's activity tracing the system calls using the logging facilities provided by the custom kernel and implementing a script to record the session commands when ever a shell is created in the Linux system, two users were created, one with a very easy to guess password and another one with a more complex but quite accessible password, the root password was set at the highest difficulty to test the capabilities of the attackers.

The first month of activity generated on the standalone test virtual machine provided us a lot of intelligence on how the attacks were being executed. The following table shows the statistics of the attacks during the first month.

TABLE 1
ATTACKS STATISTICS

Service	FTP	Samba	SSH	Apache2
Attacks	1.110 (12%)	462 (5%)	5924 (64%)	1758 (19%)
Successful	966 (87%)	328 (71%)	1421 (24%)	X
Files uploaded	113	48	906	0
Different IP's	535	171	3859	768

As we can see on the table above, the SSH service received the most attacks, but only 24% of them were successful, that is because most of the attacks were from bots that just go through dictionaries trying different sets of users and passwords, it is worth mentioning that only seven attackers were able to access with the root user, but even then they weren't able to gain full access of the virtual machine hence the security policies from the RBAc software and the custom kernel proved to work so far. During the SSH sessions the attackers usually started looking what permissions do they had, then trying to create a new user and download some root-kit or command and control software to include the machine in a bot network.

The FTP and Samba services were the ones with the highest success rate, that is because when first connecting to the service, you would receive a banner advertising the virtual machine as an anonymous server, still some attackers didn't manage to get it, probably bots without the option to access using empty credentials.

The sessions on the FTP service followed two defined patterns, the first one could be labeled as mass deploy since they uploaded files on every available directory, usually bit coin miners, and the second patter could be labeled as reconnaissance since they try to move laterally through the file system to find files with valuable data in the /etc, /opt and /home directories, but failed since the FTP user only had visibility of the directory specified for its purpose.

The successful Samba sessions followed two patterns that looked alike the FTP sessions, the first was like the FTP mass deploy, were the attackers would upload files, again mostly bit coin miners, on every available directory, and in the second one the attackers would first check if the operating system was windows or Linux, then look for all the available shares, printers and other devices, and then try to reconnect on the new found devices.

For the HTTP service, Apache2 was configured to allow any request but always return either the default index.html file or a 404 response, therefore if we define a session as a number of requests done by the same IP during a limited short time, we can separate the attacks in two sections, the first one would be PUT request were the attacker would try to input code using binary or hexadecimal characters to run commands on the virtual machine, which failed since one of the modules of Apache2 prevented any code from the requests to be executed. The second one would be a series of GET requests of different files to identify if there was any application on the Apache2 service, mostly phpMyAdmin and, less but still worth considering, WordPress. The GET request not only looked for applications but also bot-nets, in this case the most searched was muieblackcat, a web scanner bot that attempts to exploit PHP vulnerabilities or missconfigurations. Since the GET requests never found any of the files, we have no information about what the attacker may do next.

The logs generated from the attacks contained a lot of useless or repeated information and it was difficult to keep track of a single attacker through the different services of the virtual machine, hence we decided to implement a different and unified logging system for all services using a software that reads all system calls on a kernel level, then tracking the process of the service we

could easily record all activity on a single file that later, when the session had finished, would be analyzed using LUA scripts developed to extract different information from each session file, using this method also would facilitate display the output.

7 Deployment

Considering that the test on the standalone machine was successful we proceeded to deploy 8 virtual machines with different services to create a honeypot network as a proof of concept.

The following table shows the services installed on each honeypot deployed in the honeypot network.

TABLE 2
HONEYPOT NETWORK

	FTP	Samba	SSH	Apache2
Alpha			X	
Bravo			X	X
Charlie			X	X
Delta	X			
Echo	X			
Foxtrot		X		
Golf		X		
Hotel	X	X	X	X

All honeypots have the same base configuration with the custom kernel, the RBAC policies, the system call reader software and the IDS software.

We pictured a story to get accross the honeypots and test how the attackers may move through a real production network were the goal was to gain access to the Hotel honeypot.

We created a user with the most used credentials on the honeypots with the FTP and Samba services and placed a series of documents with users, passwords and IP's of the Alpha, Bravo and Charlie honeypots to gain access through the SSH service.

Since we received a lot of GET requests on the standalone honeypot concerning phpMyAdmin and WordPress, we placed an instance of phpMyAdmin on the Bravo honeypot and an instance of WordPress on the Charlie honeypot, both systems had a MySQL data base with credentials to access the same machine respectively using the SSH service and the Alpha honeypot.

In the Alpha honeypot we created a series of users bound with each of the other honeypots that lead to Alpha, to keep track of where the attacker had gotten the credentials to access it and keep track of its actions. Inside this honeypot there were files with credentials to access each service of the Hotel honeypot.

Most of the attacks the honeypots received didn't followed the story we set up and got stuck in the first honeypot they got access to, without developing any kind of lateral movement, but we will focus on the ones that did follow the story line.

From the FTP honeypots there wasn't any attack that managed to use the credentials found in the file system to gain access to the Alpha honeypot which leads to believe that most of the activity generated on there honeypots it is caused by bots and not actual human attackers, but since they downloaded the files, that could happen later on.

From the Samba honeypots one attacker managed to access the Alpha honeypot and download a set of scripts to transform the honeypot in to a proxy SMTP server to forward spam mail, using the IDS we were able to read and save all the e-mails sent, and keep a list of source systems that most likely are also compromised machines.

On the Bravo honeypot an attacker managed to gain access to the same machine using the SSH credentials stored on the MySQL data base and also transformed the honeypot in to a proxy SMTP server to forward spam mail, using also the IDE we were able to retrieve the same information.

On the Charlie Honeypot several attackers managed to use the credentials stored on the MySQL data base and gain access to the Alpha honeypot, where five of them managed to escalate privileges through a vulnerability named Dirty COW [8] of the type race condition, and find the credentials to gain access to the Hotel honeypot, these attackers certainly would have to be humans.

So far from the five attackers that managed to gain the credentials to access the Hotel honeypot, only four of them have access it. All attackers used the SSH service to access the honeypot. One of the attackers left the honeypot shortly after breaking in; therefore we can assume that he realized it was a honeypot and not a real machine. Two of the attackers spend almost an hour in the system, looking for potential valuable data and trying to install different services and root-kits that failed due to the custom kernel and the protections set by the RBAC policies. The last attacker downloaded a command and control package to use it as part of a bot-net which failed for the same reason as the previous two mentioned, then proceeded to download various bit coin miners and left.

8 Results

During the two weeks that the eight virtual machines corresponding to the honeypot network were online and fully functioning, they gathered intelligence of hundreds of attacks each machine, both statistical data, and targeted intelligence against a specific machine.

On the main entry points of the honeypot network

composed by the four honeypots with FTP and Samba services we have mostly statistical data about the attempts to break in, the credentials used and the files uploaded to the services, since the access to these machines wasn't locked down.

On the Bravo and Charlie honeypots we have both types of information, statistical data consisting of the attempts to break in the SSH service and the requests that the Apache2 service received, and intelligence on targeted attacks using the credentials stored in the MySQL data base located respectively in each machine that can be used to gain access through the SSH service.

On the Alpha and Hotel honeypots we also have both types of information, statistical data consisting of the attempts to break in the SSH service, and intelligence on targeted attacks using the credentials stored in the other honeypots.

This architecture of the network allow us very easily to classify the more interesting information concerning the targeted attacks separated from the less meaning statistical data, since the virtual machines are not real production servers, any access to anyone of the machines will be an attack without false positives.

9 Conclusions

In the requirements section, we stated five different aspects of the design that would need to be achieved to call the project a success. Through the design phase we were able to accomplish each requirement modifying the kernel base of Linux to make it more secure and create a second layer of protection between the kernel and user level with the RBAC policies, due to the new layer of permissions we never lost control of the virtual machines even when the attackers escalated its privileges to root level.

Using an IDS we were able to log all the network activity to further inspect any action when it was needed, but this feature remains mostly as a second layer of back-up information since all the attacks were analyzed using the system calls of the operating system.

Currently, the honeypot network deployed as a proof of concept is still online and keeps gathering intelligence from the attacks received, the information is centralized in a platform that allow to visualize the attacks in a friendly user view that allows the client to keep track of the actions performed in each honeypot.

The data gathered so far by the network of honeypots already provided valuable information about behaviors, trends of attacks and, in general, intelligence about cyber-threats at anonymous scale.

The solution implemented in the project seems viable to be adapted to the needs of the clients, with a few changes in the service configurations and RBAC policies, since the honeypots should resemble the actual production systems.

From a personal point of view, this project has served to learn about honeypot technology and to expand the previous knowledge about information security and networks. It also helped to take contact with many technologies, such as ElasticSearch, MySQL, IDS software, Linux kernel; to experience all the steps of a real development project; to understand the importance of a thorough research and design stage and how positively it affects to the development of the project itself; and to be able to face unexpected problems and find suitable solutions in a limited time frame.

10 Future work

Since the High interaction honeypots only differ from a real system by its use, the only limitation they have it is the time spent developing it.

There are countless other services and applications that could be implemented to have ready to deploy, but it would be better to first ask what the clients need in order to not generate useless content.

Right now the data acquired from the network sniffing by the IDS isn't being actively used, with more time we could develop a platform that analyzed that data and provided a correlation with the attacks as well as new information that may not reach the system.

The story created in the honeypot network relies in having a human attacker go through the available directories and fetching the documents that we exposed for him with the credentials, considering that a almost all attacks are originated by bots, we could develop a story that a bot may follow, creating virtual private networks between the honeypots, generating TLS/SSL certificates to access the SSH service of other machines among other methods.

Last but not least, there could be a more in deep analysis of the uploaded files and the data gathered by the honeypots.

Acknowledgements

I would like to thank both, Ramon Vicens from Blueliv and Guillermo Navarro from UAB, for their mentoring and help; to Ramon especially for making possible this project and giving me the opportunity to work in it.

I would also like to thank Ramsés Pascual, Àngel Puigventós and Asif A. Khan Ferdous for their priceless help, as well as other co-workers from Blueliv.

Finally, huge thanks to the open source community which has been really helpful through forums, mail lists and IRC chats, providing tips and clarifying a lot of doubts.

This project was supported by Blueliv.

REFERENCES

[1] M. Dornseif, T. Holz1, S. Müller, "Honeypots and Limitations of Deception", 07 June 2010.
[2] T. Grudziecki, P. Jacewicz, Ł. Juszczyk, P. Kijewski and P. Pawliński, "Proactive Detection of Security Incidents II – Honeypots", 2012.
[3] "Conode a tu enemigo: Honeynets", HIS Honeynet Project. Aviable: HTTP://his.sourceforge.net/ honeynet/papers/honeynet/, 25 July 2003.
[4] "BeeSwarm: Active IDE made easy", Johnny Vestergaard, the Honeynet Project, Aviable: HTTPs://github.com/honeynet /beeswarm.
[5] "Know Your Enemy: Sebek", The honeynet Project, Aviable: HTTP://old.honeynet.org/papers/sebek.pdf, 17 November 2003.
[6] "Linux Loadable Kernel Module HOWTO", The Linux Documentation Project, Aviable: HTTP://www.tldp.org/HOWTO/Module-HOWTO/x73.html, 24 September 2006 .
[7] "LSM Design: Mediate access to Kernel Objects," Usenix, the Advenced Computing Systems Association, [Online].Available: HTTPs://www.usenix.org/legacy /event/sec02/full_papers/wright/wright_html/node3.html.
[8] DirtyCow, Aviable: HTTPs://dirtycow.ninja/ 19 October 2016.
[9] M. T. Qassrawi, Z. Hongli, "Deception Methodology in Virtual Honeypots".
[10] F. H. Abbasi, R. J. Harris, "Experiences with a Generation III Virtual Honeynet", 13 May 2010.
[11] S. Forrest, S. Hofmeyr, A. Somayaji "the Evolution of System-call Monitoring", December 2008.
[12] Honeynet Project & Research Alliance, "Know Your Enemy: Honeywall CDROM Roo", 17 August 2005.

Annex

A1. DECEPTION STACK

.