HoneyCloud: Elastic Honeypots
On-attack Provisioning of High-interaction Honeypots

Patrice Clemente[1], Jean-Francois Lalande[1] and Jonathan Rouzaud-Cornabas[2]
[1]*ENSI de Bourges, LIFO, 88 Bd Lahitolle, 18020 Bourges, France*
[2]*Laboratoire d'Informatique du Parallélisme, INRIA, ENS Lyon, 9 Rue du Vercors, 69007 Lyon, France*

Keywords: Honeypot, Cloud Computing, Security.

Abstract: This paper presents HoneyCloud: a large-scale high-interaction honeypots architecture based on a cloud infrastructure. The paper shows how to setup and deploy on-demand virtualized honeypot hosts on a private cloud. Each attacker is elastically assigned to a new virtual honeypot instance. HoneyCloud offers a high scalability. With a small number of public IP addresses, HoneyCloud can multiplex thousands of attackers. The attacker can perform malicious activities on the honeypot and launch new attacks from the compromised host. The HoneyCloud architecture is designed to collect operating system logs about attacks, from various IDS, tools and sensors. Each virtual honeypot instance includes network and especially system sensors that gather more useful information than traditional network oriented honeypots. The paper shows how are collected the activities of attackers into the cloud storage mechanism for further forensics. HoneyCloud also addresses efficient attacker's session storage, long term session management, isolation between attackers and fidelity of hosts.

1 INTRODUCTION

Honeypots are hosts that welcome remote attackers. Honeypots enable to collect valuable data about these attackers, their motivations and to test countermeasures against attacks. Two major issues with honeypots are their robustness and their scalability. As the attacker tries to violate the security of the host, he can damage the host or the contained data. Moreover, welcoming attackers uses a large amount of resources and needs frequent re-installations.

This paper proposes a new type of honeypot: HoneyCloud which is a honeyfarm, i.e., an architecture designed to provide multiple honeypots, able to deploy virtual honeypots on-demand in a cloud. This proposal solves the two previous issues: HoneyCloud is highly scalable and the robustness of the host might be relaxed as a new honeypot virtual machine (VM) is automatically provisioned for each new attacker. Our goal is to show that the proposed solution enables to setup high-interaction honeypots that help to study attacks at OS level. Our solution increases the fidelity of both the honeypot host (from the attacker point of view) and the fidelity of the collected data. Moreover, our solution also preserves the scalability of the farm of honeypots.

The paper is organized as follows: the next section exposes the motivations for this work regarding the current state of the art. The proposed HoneyCloud infrastructure is described in Section 3. Section 4 describes the virtualized hosts that welcome the attackers. Section 5 concludes the paper and gives some perspectives.

2 MOTIVATION

In this work, we use honeypots in order to collect attack logs. "*A honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource*" (Spitzner, 2003). Honeypots are often classified by the level of interaction they provide to attackers:

1. Passive network probes: such tools can be deployed widely and welcome/analyze millions of IP addresses, e.g., Network Telescopes (Moore et al., 2004).
2. Low-interaction honeypots: easy to setup and administrate, but simply emulate services. Attackers have the illusion of using normal computers.
3. High-interaction honeypots: the most difficult to setup, deploy and maintain, but provide the best fidelity for attackers.

Clemente P., Lalande J. and Rouzaud-Cornabas J..
HoneyCloud: Elastic Honeypots - On-attack Provisioning of High-interaction Honeypots.
DOI: 10.5220/0004129604340439
In *Proceedings of the International Conference on Security and Cryptography* (SECRYPT-2012), pages 434-439
ISBN: 978-989-8565-24-2

We mainly focus on OS level aspects of attacks for several reasons. Except for DoS/DDoS attacks and massive network scans, most attacks aim at exploiting an OS vulnerability to finally steal or corrupt information, or to install distributed botnets onto the OS. Thus, attacks should be studied at OS level to be able to have a good description of them. Following these objectives, in previous work, we correlate various OS level logs to rebuild the full sessions of attacks (Bousquet et al., 2011) that were collected using a previous honeypots architecture (Briffaut et al., 2012). The architecture presented in this paper is new. It aims at being fully scalable, easier to use and for forensics while providing almost the same fidelity as the previous one.

As explained in the following section, current honeypot solutions have to find a tradeoff between fidelity and scalability. This issue, and the fine description at OS level of the attacks are what we mainly want to address in this work.

2.1 Fidelity

Low-interaction honeypots, e.g., *Honeyd* (Provos, 2004), DarkPots (Shimoda et al., 2010) and farms of such honeypots, e.g., SGNET (Leita and Dacier, 2008)) provide scalability but do not gather any information about system events related to attacks. At best, they collect network logs and malwares. HoneyCloud collects all the attack steps from a system point of view. Precisely, it collects keystrokes, HIDS events, syslog/SELinux audit logs and other host sensor logs. This provides the highest fidelity of attack data.

In addition, HoneyCloud also aims at providing the highest fidelity for attackers, by using high interaction honeypots. High interaction honeypots are full operating systems that interact with attackers. They offer the possibility to monitor the consequences of an attack at network and OS levels.

On physical high-interaction honeypot hosts, when dealing with real system sessions, i.e., from login to logout, the issue is that all the system events of different attackers are mixed in the same logs of the host. That complicates the analysis and understanding of the logs and needs technics like PID tree reconstruction to isolate events for each session (Briffaut et al., 2012). With HoneyCloud, each attacker gets his own virtualized honeypot, avoiding the mix of the precious logs.

2.2 Scalability

In a classical honeyfarm, a pool of public IPs is redirected to a pool of hosts that will welcome the attackers. In case of a very high number of attackers at the same moment, this architecture may not be sufficient to welcome correctly the attackers, as performances may dramatically decrease.

When there is a real need of provisioning new resources, the use of a cloud and elasticity seems the most appropriate. Using clouds leads to a better consolidation of resource usage and helps to implement green IT: resources are assigned on-demand when they are needed and are powered-off or put in hibernation otherwise. Thus our HoneyCloud architecture aims at providing both scalability and fidelity, but also global efficiency: save of computing resources, money, and the planet!

In (Balamurugan and Poornima, 2011), the authors present a Honeypot-as-a-Service high-level architecture, without any implementation evidence. On the opposite of their vision, we do not think that honeypots can help to trap attackers and thus protect legal computer sharing the same network. In particular, filtering of attackers is very difficult for *0-day* attacks.

2.2.1 On-attack Provisioning of Honeypots

Honeylab (Chin et al., 2009), a honeyfarm approach, provides high-interaction honeypots but does not provide any on-demand service allowing to provision new virtual honeypots when new attackers arrive. Collapsar (Jiang and Xu, 2004), a VM based honeyfarm architecture for network attack collection, shares the same limitations: the honeypots are statically deployed once and for all. HoneyCloud clearly overcomes this limitation.

In (Vrable et al., 2005), the authors introduce the Potemkin architecture, which is the closest one to our proposal. Potemkin uses virtualization and thus has a better scalability. However, Potemkin is intrinsically limited to 65k honeypots, i.e., 65k IP addresses. Moreover, the authors give only results for a representative 10 minutes period. Our HoneyCloud proposal is not essentially limited to a number of IP addresses. Indeed, as a cloud computing architecture, each public IP of the HoneyCloud can handle a possibly infinite number or attack sessions. Each attacker receives on-the-fly its own virtual honeypot. More than that, since our HoneyCloud architecture is based on the Amazon EC2 API, it could potentially extends its honeyfarms physical servers to other (private) clouds compatible with EC2.

2.2.2 Attack Sessions

Potemkin starts a VM per public IP and does not consider the IP of the attacker. Thus, Potemkin is not

able to explicitly isolate attackers between each others. Potemkin does not really consider attackers' sessions and thus does not deal with them (no session storing, so session analysis). Potemkin only maintains active IPs for a set of services and OS to deliver. Potemkin keeps the running state for each VM: its load and liveness, and stands ready to shut down the VM as long as it is not used any more.

As a single VM may welcome multiple attackers in Potemkin, forensics may be quite complicated in some cases, as explained before with non-virtualized (physical) honeypots. Of course, some works, like Nepenthes (Baecher et al., 2006) store VMs' snapshots for future forensics, but those snapshots do not distinguish attackers sessions, and most of the time, those snapshots seem to be only used for manual investigations ((Jiang and Xu, 2004), pp.1173-1174). Potemkin, Nepenthes and many others share the same drawback: they do not keep all precise events related to attacks. The use of HoneyCloud provides a set of clear system logs. HoneyCloud only accepts connections on the `ssh` port 22 and thus attack sessions are lighter to store. It is also easier to restore later such sessions back to honeypots if the corresponding VMs have been deleted due to a too long *idle* time. That voluntary restriction to `ssh` based attacks solve many problems of resource consumption. For example, existing honeyfarms of virtualized honeypots have to deal with network scans: shall the architecture instantiate a new honeypot VM for each IP scanned? Our vision intrinsically tamper this issue. More than that, it is now common that attackers use encryption "enabled backdoors, like trojaned `sshd` deamons" (Jiang and Xu, 2004). So, even if we use some classical network sensors, such as *snort*, we can not only rely on network logs. This is is the reason why we focus on OS events and host logging systems.

2.2.3 Instantiation, Storage and Recycling

HoneyCloud introduces the ability to easily and dynamically setup a new environment for each attacker. The challenge is to setup an architecture that is persistent for the attacker as it may come back latter and should not notice any change between the two attack moments. But the technical solution must also ensure a lightweight storage mechanism. An attack can sometimes take several days to end up and may finish with sending back some reports, e.g., scan results. Some attacks need each step to be validated remotely (Bousquet et al., 2011). It is thus not relevant to prematurely recycle the virtual honeypot and its resources. HoneyCloud stores all attackers sessions in order to revive those session if the attacker comes back later, even if the VM itself has been deleted. Sessions are stored in a abstract and compressed representation but not the related VM.

2.2.4 Network Traffic Management

While Potemkin uses hidden external routers to route 65k IP addresses to the Potemkin gateway, which in turn redirects the flow to physical servers, HoneyCloud can handle virtually much more connections at the same time with only a few range of IP addresses. Potemkin and Collapsar have the most sophisticated outgoing traffic redirection policies. However, they consider redirecting outgoing traffic within the honeyfarm, whereas other solutions, e.g., Honeylab, only allows to see the DNS servers. In some cases, redirecting all the traffic inside can maintain some illusion, e.g., for worms. But in many cases – manual or scripted attacks, or malwares that need to communicate outside (using encrypted connections) – those policies are not relevant.

Our policy is simple, HoneyCloud only limits the outgoing rate but not the destination's nature. Contrary to Collapsar (Jiang and Xu, 2004), HoneyCloud does not limit the attackers: HoneyCloud does not avoid internal network and system scans.

2.3 Robustness, Detection, Protection

In the state of the art, many researchers using virtualized honeypots try to obfuscate the virtualization technologies. But we think this is not needed any more: nowadays, almost servers tend to be virtualized. So, virtualization is not always an evidence of being captured by a honeypot. It can even mean the opposite: only big companies and Cloud Service Providers have funds big enough to build cloud architectures. However, some of the sensors used in HoneyCloud, like *snort*, are well-known and can easily be detected. Even if invisibility is not our main goal at the moment, fidelity is needed in order to convince attacker that he entered in a regular network. To achieve authenticity, honeypots share some network and OS configuration with normal servers: DNS configuration, mail servers of the domain.

No direct protection for the virtual honeypots is provided. Again, providing a vulnerable machine is the aim, not its opposite. However, one can assume that, by re-cloning the reference VM after a long time of inactivity, some protection is provided for the honeypots.

When an attacker gains root priviledges on a classical honeypot, he may corrupt or delete his own data, history and logs, and sometimes also others' ones. HoneyCloud also manages this issue. Indeed, isolation is provided between users, guaranteeing the pro-

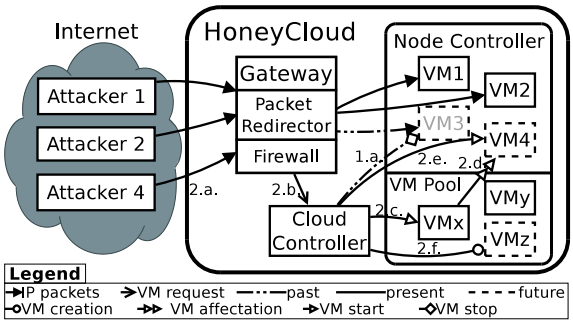

Figure 1: Overall architecture of HoneyCloud.

tection between the VMs, and thus protection of the attacks' logs.

At last, while using public IPs among the legal IPs locally owned, the honeypot server is physically not connected to the normal local network, which prevents the use of HoneyCloud against any form of stepping-stone for further attack on the legitimate network.

3 ARCHITECTURE

The architecture proposed here uses a private open-source cloud: Eucalyptus (Nurmi et al., 2009). Eucalyptus exposes the EC2 API. This allows our HoneyCloud to work on any other clouds that provide EC2, without any modification.

The remaining of this section describes how the proposal uses the cloud and what new services have been added to implement the HoneyCloud. First, the gateway facility is introduced. It acts as a network flow redirector for incoming packets of attackers. Then, the configuration and security settings of the running VMs are presented. Then, the implemented data persistence service is described for the data uploaded and/or modified by attackers, when connected. Finally, the sensors used on the HoneyCloud and their setup are presented.

3.1 Provisioning Architecture

The Figure 1 presents the architecture and describes the general scenario of provisioning virtual honeypots. Attackers (on the left) connect to the HoneyCloud by the gateway. Attackers #1 and #2 already have a honeypot assigned, i.e., resp. VM 1 and VM 2. Their packets are redirected to their corresponding VMs. Attacker #3 has left a long time ago. Thus, its honeypot (VM 3) is deleted by the Cloud Controller (arrow 1.a). Later arrives attacker #4 (2.a). The gateway requests the Cloud Controller for a new VM (2.b),

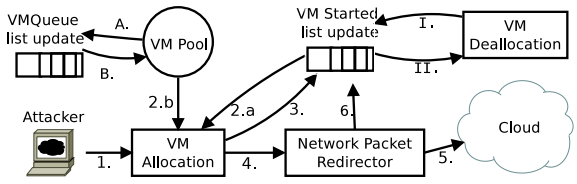

Figure 2: HoneyCloud Gateway Flowchart.

which in turn assigns the VM*x* (2.c) from the VMs pool to the attacker #4 (2.d), and starts the VM now labelled as VM 4 (2.e). The attacker #4 can now interact with its VM. At the same time, the Cloud Controller creates VMz in the pool, ready for further assignment (2.f).

3.2 Gateway

The purpose of the gateway is to act as an entry-point for attackers on the HoneyCloud. The gateway is connected to a range of IP addresses (that are continuous or discontinuous ones). It redirects the attackers to a VM that is started on-demand for each attacker by the Cloud Controller. The gateway also shutdowns the VMs once they are no more in use.

The gateway uses two separate lists: `VMQueue` that contains the list of VMs that are started but not allocated to attackers and `VMStarted` that contains the list of VMs that are allocated to the pair ⟨attacker IP, public IP⟩. It means that, for each attacker and for each targeted public IP, a VM is provisioned. Furthermore, two static variables are needed to setup the whole gateway: `POOL_SIZE` defines the number of started VMs that are not allocated yet (2 by default) and `DURATION` defines the amount of time before a VM is stopped when it does not received new packets (60 minutes by default).

As shown on Figure 2, the life cycle of the gateway can be separated into three main parts, each part using one or more *python* threads. The first part (edges 1 to 6) implements the routing algorithm of attacker's incoming packets. The main steps are described below:

1 A (new) attacker sends a packet to one of the public IP addresses allocated for the gateway.

2.a The VM allocation thread checks if a VM is already allocated to the pair ⟨attacker IP, public IP⟩. If it is the case, the process jumps to step 4.

2.b A VM is pulled from the `VMQueue` list.

3 The `VMStarted` list is updated with the newly assigned VM to the pair ⟨attacker IP, public IP⟩.

4 The packet sent by the attacker is forwarded to the Network Packet Redirector.

5 The packet is sent to the allocated VM on the cloud.

6 The `VMStarted` list is updated for the VM with the date of the last packet.

The second part (edges *A* and *B*) of the gateway checks the size of `VMQueue` list (edge *A*). If the queue size is smaller than `POOL_SIZE`, new instances are started to refill the queue.

The third part (edges *I* and *II*) of the gateway checks the date at when the last packet has been received for each VM. If `LAST_PACKET` + `DURATION` $<$ *CurrentDate*, the corresponding VM is stopped.

4 HONEYPOT VMS

Each incoming attacker is welcomed in a VM through an `ssh` tunnel on port 22. The goal is to provide a remote shell to the attacker into the VM. From this shell, the attacker will be able to perform its malicious activities, mainly:

- to inspect the system: available resources, vulnerabilities, data;
- to use the host as a stepping stone host for further attacks to other computers;
- to install malicious software as botnets, worms, keyloggers, scanners, etc;
- to exploit vulnerabilities to become root of the system.

In our experiments, the goal was not to avoid the attacker to become root or to install a malware: if the virtualized host is compromised or totally unusable, it does not impact the cloud infrastructure nor the other attackers. On the contrary, HoneyCloud is setup to help the attacker to enter the honeypot while hiding the cloud infrastructure and auditing what is performed. These three points are described in the next sections.

4.1 Welcome Operations

A modified `open-ssh` server is integrated into the VMs. The service allows remote password connection of attackers and automatically accepts randomly the tried login/password when bruteforcing the service. It allows for example to accept 10% of the tried login/-password pairs. When the service accepts a challenge from, for example *bob*, it automatically adds *bob* to the system and creates its home directory. Thus, the attacker obtains a remote shell located at `/home/bob`.

4.2 Data Persistence

One of the main difficulties in honeypot solutions close to ours is the honeypot storage. How to store efficiently honeypots that have been used by attackers, but are currently not in use? The objectives are double: to keep the honeypot available for further attack steps of the attacker, and to preserve valuable attack information for future forensics. To solve theses issues, we choose to make a self abstract version of a lightweight snapshot of the VMs. This vision has the advantage of requiring almost no space for the storage. Concretely, the deployed VMs in HoneyCloud have a special `init.d` script that save the home directory of the attacker when the VM is powered-off. To do that, HoneyCloud uses the *Walrus* storage mechanism of Eucalyptus[1] which offers an access to a centralized storage zone. The record is based on the attacker's login, which is used to name the *bucket* stored in Walrus. If the attacker *bob* comes back latter, a new VM is assigned to him. Nevertheless, just before creating his home directory, the `ssh` service checks into the Walrus storage service if the same login has already been seen. If so, the corresponding bucket is recovered and *bob*'s home directory is restored. This way, it offers homedir persistence to the attacker.

Of course, there are some limitations to this mechanism. A deep and careful analysis of the system could reveal to the attacker that the VM of the second connection is not the same as the first one. For example, the uptime of the host may not be consistent. We may very soon improve this mechanism by also checking the source IP of the attacker when he tries to use an existing login. Another limitation is that restoring only *homedirs* can not bring back system modifications made by malwares for example, nor malware themselves. We have to face all those limitations in the next phase of this work in order to reflect almost perfectly the VM the attacker left before coming back.

4.3 Sensors

With the proposed architecture, it becomes possible to install multiple sensors into the virtualized hosts. In our experiment, we deployed the classical network and system sensors that may report malicious events in the logs of the virtualized host. The network and system sensors are:

- **P0f**, for passive network packet analysis. It can identify the operating system of the attacker's host that is connected to the audited host.

[1]Walrus is a storage service included with Eucalyptus that is interface compatible with Amazon's S3 http://open.eucalyptus.com.

- **Snort**, for real time traffic analysis of the packet that are exchanged between the two hosts.
- **syslog/SELinux**, that is activated in auditing mode. It enables to monitor all the forbidden interactions that are controled by the standard "targeted" policy.
- **Osiris**, that monitors any change in the system's files and kernel modules.

All the sensor's logs are saved before killing the VM for off-line investigation. Furthermore, important system files like the user's `bash_history`, `/dev/shm`, `/tmp` are saved too.

5 CONCLUDING REMARKS

This paper introduces HoneyCloud, a new honeypot infrastructure based on cloud computing technics that enables to deploy a large-scale high interaction honeyfarm. This new type of honeyfarm provides a virtualized honeypot host per attacker. HoneyCloud introduces persistence facilities in order to restore the homedirectory of the attacker in case of multiple venues. The architecture lets the attacker exploit any vulnerability of the honeypot. He may become root and install malicious software. This is a real advantage as HoneyCloud stores all network and system logs related to attacker's session, enabling to finely study the attacks.

The architecture of HoneyCloud is very scalable, as it is based on a cloud and can multiplex a few public IP to thousands of attackers. Further works will focus on deploying HoneyCloud on a larger infrastructure as the one used here in order to collect attacks logs during a long period.

REFERENCES

Baecher, P., Koetter, M., Holz, T., Dornseif, M., and Freiling, F. (2006). The Nepenthes platform: An efficient approach to collect malware. In *9th international symposium on Recent Advances in Intrusion Detection (RAID)*, pages 165–184, Hamburg, Germany. Springer.

Balamurugan, M. and Poornima, B. S. C. (2011). Article: Honeypot as a service in cloud. *IJCA Proceedings on International Conference on Web Services Computing (ICWSC)*, ICWSC(1):39–43. Published by Foundation of Computer Science, New York, USA.

Bousquet, A., Clemente, P., and Lalande, J.-F. (2011). SYNEMA: visual monitoring of network and system security sensors. In *International Conference on Security and Cryptography*, pages 375–378, Séville, Espagne.

Briffaut, J., Clemente, P., Lalande, J.-F., and Rouzaud-Cornabas, J. (2012). Honeypot forensics for system and network SIEM design. In *Advances in Security Information Management: Perceptions and Outcomes*, pages –. Nova Science Publishers.

Chin, W. Y., Markatos, E. P., Antonatos, S., and Ioannidis, S. (2009). HoneyLab: Large-scale honeypot deployment and resource sharing. In *NSS'09: Proceedings of the 2009 Third International Conference on Network and System Security*, pages 381–388, Gold Coast, Queensland, Australia. IEEE Computer Society.

Jiang, X. and Xu, D. (2004). Collapsar: a VM-based architecture for network attack detention center. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 2–2, Boston, MA, USA. USENIX Association.

Leita, C. and Dacier, M. (2008). SGNET: A worldwide deployable framework to support the analysis of malware threat models. In *EDCC-7 '08: Proceedings of the 2008 Seventh European Dependable Computing Conference*, pages 99–109, Kaunas, Lituania. IEEE Computer Society.

Moore, D., Shannon, C., Voelker, G., and Savage, S. (2004). Network telescopes: Technical report. *CAIDA, April*.

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. (2009). The eucalyptus open-source cloud-computing system. In *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131, Shangai, China. IEEE Computer Society.

Provos, N. (2004). A virtual honeypot framework. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, Boston, MA, USA. USENIX Association.

Shimoda, A., Mori, T., and Goto, S. (2010). Sensor in the dark: Building untraceable large-scale honeypots using virtualization technologies. In *2010 10th Annual International Symposium on Applications and the Internet*, pages 22–30, Seoul, Korea. IEEE Society.

Spitzner, L. (2003). *Honeypots: tracking hackers*. Addison-Wesley Professional.

Vrable, M., Ma, J., Chen, J., Moore, D., Vandekieft, E., Snoeren, A. C., Voelker, G. M., and Savage, S. (2005). Scalability, fidelity, and containment in the Potemkin virtual honeyfarm. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 148–162, Brighton, United Kingdom. ACM.