HoneyChart: Automated Honeypot Management Over Kubernetes

George Kokolakis[1], Grigoris Ntousakis[2,3], Irodotos Karatsoris[2], Spiros Antonatos[3], Manos Athanatos[3,4], and Sotiris Ioannidis[2,4]

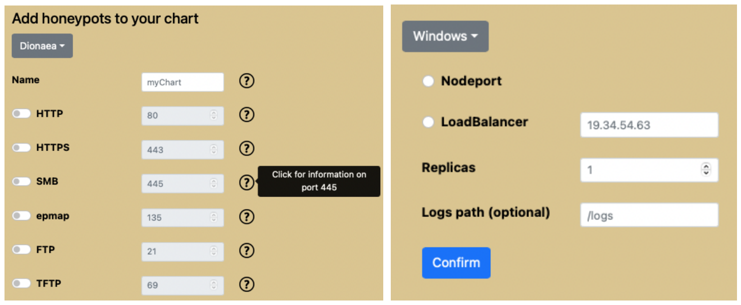
[1] Georgia Institute of Technology, USA
gkokolakis6@gatech.edu
[2] Technical University of Crete (TUC), Greece
{gntousakis, ikaratsoris, sioannidis}@tuc.gr
[3] Telecommunication Systems Research Institute (TSI), Greece
{gntousakis, santonatos, mathanatos}@tsi.gr
[4] The Foundation for Research and Technology – Hellas (FORTH), Greece
{athanat, sotiris}@ics.forth.gr

Abstract. Honeypots have been proven to be a useful tool in the arsenal of defense solutions against cyber-attacks. Over time, various honeypot solutions have been proposed to lure attackers that target both conventional networks and Industrial Control Systems. However, the current approaches do not make the deployment and usability of honeypots more attractive to defenders. In this paper we propose HoneyChart, a framework for honeypot deployment that leverages on Helm Charts for Kubernetes to create honeypot templates from existing virtualized environments and deploy the appropriate honeypots based on the desired services. HoneyChart allows the fast and automated deployment of containerized honeypots, allowing the defenders to focus on what really matters: the analysis of attacks, IoCs and imminent threats.

1 Introduction

A formal definition of a honeypot is a trap set to detect, deflect or in some manner counteract attempts at unauthorized use of information systems. Practically, honeypots are systems set up to lure attackers. Honeypots are non-production systems, which means machines that do not belong to any user or run publicly available services. Instead, in most cases, they passively wait for attackers to contact them. By default, all traffic destined to honeypots is malicious or unauthorized as it should not exist in the first place. Honeypots can also assume other forms, like files, database records, or e-mails.

The original goal of honeypots was to detect worms. As worms cannot afford to acquire the knowledge whether their next victim is a honeypot or not, they blindly try to attack the whole IP address space. Based on that behavior, honeypots are able to catch instances of worms by running decoy services, analyze them, and pass all the gathered information to signature generation systems and behavior analysis modules. However, as the attack landscape has changed and the existence


Add honeypots to your chart
Dionaea
Click for information on port 445
Windows
Confirm

Name	myChart
HTTP	80
HTTPS	443
SMB	445
epmap	135
FTP	21
TFTP	69

Nodeport	
LoadBalancer	19.34.54.63
Replicas	1
Logs path (optional)	/logs

(a) Adding honeypot to chart. (b) Using pre-configured interfaces.

Fig. 1: HoneyChart web interface.

of automated worms has become rare, the use of honeypots has also evolved. Honeypots are now used to detect worm-able attacks, like BlueKeep [11] or provide useful insights for protecting Industrial Control Systems (ICS) and their SCADA components.

While a number of honeypot solutions have been proposed, they also suffer from a major drawback when it comes to their deployability. They all come as standalone scripts or set of software components that need to be manually deployed and monitored. Most of the current honeypot developments has not caught up with the recent advances in virtualization and container orchestration, which are now become more prevalent for production workloads and even 5G core networks [13].

In this paper we try to close the gap by proposing HoneyChart, a framework for honeypot deployment that leverages Kubernetes to create honeypot templates from existing virtualized environments and deploy the appropriate honeypots based on the desired services. HoneyChart comes with a set of tools to map services running inside a Kubernetes cluster into their corresponding honeypot services and propose to the security administrators a meaningful template ready for deployment. In a next step, HoneyChart allows the fast and automated deployment of containerized honeypots. HoneyChart has been open-sourced and is available for download from GitHub: github.com/parasecurity/honeychart.

2 Background

In this section of the paper we are going to present background information about Honeypots (§2.1), Kubernetes (§2.2), and Helm Charts (§2.3).

2.1 Honeypots

The main classification criterion of a honeypot is its level of interactivity with the attackers. Honeypots can either do simple service emulation (low-interaction), more advanced emulation (medium-interaction) or run real services (high-interaction). Low-interaction honeypots emulate IT services that are usually requested by attackers. They do not provide an actual OS, only a limited subset of the functionality attackers would expect from a server [21]. Several low-interaction honeypots have been implemented, like honeyd [23], Honeytrap [6], LaBrea [9], and Dionaea [3]. A high-interaction honeypot is a complete system, which contains a fully functional OS and all the services that it could provide. High-interaction honeypots are used to capture the maximum amount of information concerning new and old ways of attacking. Minos [17] is a microarchitecture that implements Biba's low water-mark integrity policy [15] on individual words of data. Argos [22] is a containment environment for worms and manual system compromises. It is actually an extended version of the Qemu emulator that tracks whether data coming from the network is used as jump targets, function addresses or instructions by performing dynamic taint analysis [20]. Medium-interaction honeypots attempt to mix the benefits of both low and high-interaction honeypots. They are more advanced than low-interaction honeypots, but they are not as advanced as high-interaction honeypots. The Nepenthes platform [14] and Multipot [12] honeypots fall into this category.

Honeypots have also been used for protecting SCADA networks. Conpot [1] is a low-interaction server-side ICS honeypot and supports a variety of SmartGrid use cases. Conpot supports protocols such as Modbus TCP, HTTP, IEC104, FTP, TFTP, S7Comm, BACnet, and SNMP, and it, like other honeypots, can keep track of attacks. CryPLH [16] is a high-interaction honeypot that emulates a Siemens S7-300 PLC, with HTTP/ HTTPS, S7comm, and SNMP services running on a Linux host modified to accept connections on specific ports. SHaPe [18] (Scada HoneyPot) is a low-interaction honeypot that can be used on substation automation systems. S7commTrace [24] is a honeypot based on the S7 protocol. This protocol is running on PLCs of Siemens S7-300, 400, 1200 and 1500 series. HoneyPLC [19] is a high-interaction, malware-collecting honeypot for ICS. It simulates TCP/IP, HTTP, SNMP and S7comm protocols.

2.2 Kubernetes

Kubernetes [8], at its basic level, is a system for running and coordinating containerized applications across a cluster of machines. It is a platform designed to completely manage the life cycle of containerized applications and services using methods that provide predictability, scalability, and high availability. A Kubernetes system consists of a master node and any number of worker nodes. Kubernetes deploys applications to the cluster of worker nodes after the developer submits a list of applications to the master.

2.3 Helm Charts

Helm [5], is a package manager for Kubernetes. Helm is an open-source project that was originally developed by Deis Labs and is now maintained by Cloud Native Computing Foundation (CNCF). Helm was created with the intention of giving users a better way to manage all of the Kubernetes YAML (description) files that they create on Kubernetes projects. Deis Labs developed Helm Charts as a means of resolving the description file management problem. Each chart is a collection of one or more Kubernetes manifests while a chart can also have child and dependent charts. When we run the install command for the top-level chart, Helm installs the entire project's dependency tree. The Kubernetes project's goal is to manage your containers, but it cannot use template files. Helm gives us the ability to create template files and add to them variables and functions. These files are truly generic and can be used across large teams or organizations to deploy scalable applications where their parameters can be changed at any time.

3 Implementation

In this section of the paper we are going to present the implementation details of HoneyChart. To ease the deployment process of honeypots we have developed a tool for generating Helm Charts of containerized honeypots for deployment in a Kubernetes cluster. The charts we create can contain descriptions and deployment parameters for a single or multiple Honeypots. We support multiple honeypot types, including low-interaction and high-interaction honeypots. We create the honeypot charts via the HoneyChart web interface (Fig. 1. We developed the front-end of HoneyChart using `HTML`, `CSS`, and `JavaScript`. We developed the back-end of HoneyChart using `Node.js`.

We provide to the user of HoneyChart two main options. The user can either create custom honeypots (§3.3) or create honeypots based on pre-built templates, derived from real world scenarios, exposing specific interfaces (§3.2). The pre-built interfaces allow the user to select pre-built honeypot Helm Charts that can simulate different protocols, e.g. the PLC communication protocols or the Microsoft Windows protocols. With the custom honeypots creation option we allow the user to create custom Helm Charts configuring the supported honeypots on the platform, based on his requirements. All this functionality is supported by our web interface.

3.1 Custom Honeypots

On the Custom Honeypots page, the user fist chooses which honeypot is going to use. At the moment, there are three available honeypot images on HoneyChart. These are Dionaea [3], Conpot [1], and Cowrie [2]. After selecting the honeypot image, the user can then enable or disable the available protocols supported by the honeypot.

In addition to the selection of the honeypot image, we are able to provide to the user multiple other configuration options. For example, the user can select which Kubernetes `ServiceType` they prefer for their deployment. Kubernetes `ServiceType` is used when a developer wants to expose an application to an IP address that is outside of the cluster. The available choices of `ServiceType` are `Nodeport` or `LoadBalancer`. If the user selects `Nodeport`, they do not need to fill

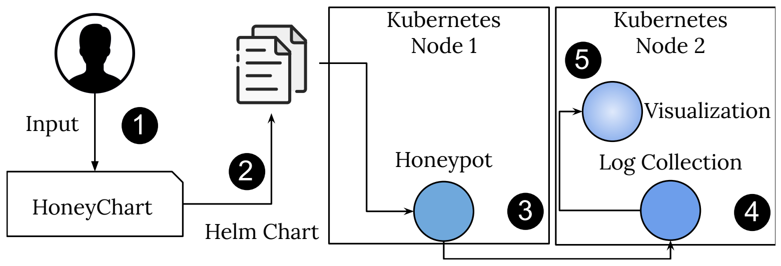


Fig. 2: Overview of HONEYCHART usage.

the IP address field. If they select the `LoadBalancer` option, then they must fill the IP address field using the `IPv4` format (`IBM`, `IPv4` and `IPv6` address formats).

Another configuration option available is the number of replicas and the file path for the honeypots' logs. In the replicas section the user can specify how many replicas of the Helm Chart they need. The allowed range of values is from one to 100. This information can be seen by hovering over the question mark next to the replicas text field. Logs' path allows the user to specify in which directory the honeypot logs will be extracted.

After filling the form and pressing the `Add` button the application performs a series of validations to make sure that the user's options are correct. If there is an error, the application will notify the user with a warning. Otherwise, the user has the option to add more honeypots to the chart, or finish the process by pressing the `Create` button. If they choose to add more honeypots to the chart, they just have to follow the previous steps again. Otherwise, the user downloads the charts to their computer and they can now deploy it on their Kubernetes cluster. With the options that HONEYCHART provides, the user can combine services from different honeypots to build certain profiles.

After the user clicks the `Create` button the Helm Chart generation begins. The data structure gets converted to a `JSON` string on the front-end and with a POST request the program sends the `JSON` string to the back-end.

When the server receives the JSON string, it validates the name of the Helm Chart for security purposes. Afterwards the program initializes the `generic_values`, `generic_deployment`, and `generic_service` data structures. The process continues with specifying the logs path for both the container and the worker node. Depending on the selection of honeypots made by the user, the program fills out values, services and deployment data to the corresponding data structures. Then we create three YAML files, `values.yaml`, `deployment.yaml`, and `service.yaml` using the three data structures.

Now that we have all the needed information, HONEYCHART creates a chart directory with all of the necessary files and directories. Then, HONEYCHART moves the `values.yaml` file to the chart's root directory and the `deployment.yaml` and the `service.yaml` to the templates directory. When the chart directory is ready the server creates a zip file from the directory and then the server sends the zip file to the client as a response.

3.2 Pre-built Interfaces

At the pre-built interfaces page the user can select specific setups from a drop-down menu. Then the user can choose the service type, set the number of replicas, and the honeypot log path. Every available interface is created in the form of a `JSON` file. The file contains the name of the chart, the names of the honeypots that will be used, a default logs path, the services, the container ports, the protocol type for every service that is going to be used and a string that contains a description of the honeypot's services. After the user has selected the pre-built interface, the service type, the number of replicas and the log path they must click confirm to move to the next step of the process.

Pre-built Interfaces page follows the same principles as the Custom Honeypots page. First, the program initializes the data structure. Afterwards, it checks the `ServiceType`. If the `LoadBalancer` is selected then it validates the IP address. The program then fills the `ServiceType` and the IP address, when applicable, to the data structure. Next, it validates the replica number and registers the number to the data structure. Finally, it fills the logs path in the data structure.

When the data structure is ready then the chart information gets displayed on the page. With the information panel the user can review their choices and then they can click `Create` to finish the chart creation. The chart creation process is the same as the custom build option.

3.3 Automated Port Mapping

Another tool that comes bundled with HoneyChart, is a template suggestion script that creates a Helm Chart based on services running in the Kubernetes cluster. This `python` script identifies the ports and protocols of each running service in the cluster and initiates a post request to HoneyChart containing that information. Subsequently, HoneyChart processes the request by mapping the corresponding ports and protocols that were identified with the corresponding services that the three honeypots offer and produces a Helm Chart which is sent back to the cluster.

4 Deployment

In this section of the paper we are going to present an in-depth example of a chart generation using HoneyChart and a deployment using the generated Helm Charts.

We are going to use HoneyChart to generate a Helm Chart for a `Conpot` honeypot deployment. `Conpot` honeypot is a low-interaction server-side `ICS/SCADA` honeypot. The `ServiceType` in our example is `LoadBalancer` and we are deploying a single replica of the honeypot.

After the creation of the `Conpot` honeypot chart the user can download it in a `zip` format and the user can deploy it on their Kubernetes cluster. The file is downloaded in `zip` format. After extraction, the chart can be deployed using the following installation command. In our example the name of the chart is `conpot-chart`.

```
helm install conpot-chart conpot-chart
```

After running the above command with `kubelet` on the worker node it will instruct Docker to download and run the `Conpot` container image from the Dockerhub registry. To check if the pod is running we run the following command.

```
kubectl get pods
```

When the pod is ready we can run `kubectl` get services to display additional information about the running pods, such as `ServiceType`, `Cluster-IP`, `External-IP`, exposed ports, and the amount of time the pod is running.

```
kubectl get services
```

Now that we know that the pod is running properly on the Kubernetes cluster, we can scan the external IP using the `nmap` scanning tool. The `nmap` tool is an open-source Linux command-line tool for scanning IP addresses and ports in a network.

After the honeypots deployment, we harvest their `JSON` logs using filebeat [4]. Filebeat send the logs to logstash [10], where an extra processing occurs in order to get stored in an efficient format to Elastic search. Finally, visualize the logs using kibana [7].

5 Conclusion

We use honeypots to detect, deflect or counteract unauthorized access to information systems. While a large number of honeypots have been proposed and created, the deployment process of honeypots has not been improved in recent years. In this work, we proposed HoneyChart, a framework for honeypot deployment that eases their deployment process. To ease the deployment process of honeypots we leverage the use of Helm Charts and Kubernetes. We create Helm Charts though a web-interface with a set of predefined or custom options. Then we use helm to deploy them on the desired Kubernetes cluster. We demonstrated the use of HoneyChart on an extensive example. HoneyChart source code is available to download from github.com/parasecurity/honeychart.

6 Acknowledgments

This work was partly supported by DARPA contract no. HR001120C0155. This work has also received funding from JCOP project (GA No INEA/CEF/ICT/A2020/2373266) and from European Union's Horizon 2020 program COLLABS (No 871518).

References

1. Conpot ICS/SCADA Honeypot. http://conpot.org/
2. Cowrie. https://github.com/cowrie/cowrie
3. Dionaea. https://github.com/DinoTools/dionaea
4. Filebeat. https://www.elastic.co/beats/filebeat
5. Helm. https://helm.sh//
6. Honeytrap. https://www.honeynet.org/projects/active/honeytrap//
7. Kibana. https://www.elastic.co/kibana/
8. Kubernetes. http://kubernetes.io
9. Labrea. http://labrea.sourceforge.net/labrea-info.html
10. Logstash. https://www.elastic.co/logstash/
11. Microsoft operating systems bluekeep vulnerability. https://www.cisa.gov/uscert/ncas/alerts/AA19-168A
12. Multipot. https://inc0x0.com/2019/05/multipot-web-application-honeypot-with-built-in-analysis-tools/
13. Why kubernetes over bare metal infrastructure is optimal for cloud native applications. https://www.ericsson.com/en/blog/2022/5/kubernetes-over-bare-metal-cloud-infrastructure-why-its-important-and-what-you-need-to-know
14. Baecher, P., Koetter, M., Holz, T., Dornseif, M., Freiling, F.: The nepenthes platform: An efficient approach to collect malware. pp. 165–184 (09 2006)
15. Biba, K.: Integrity considerations for secure computer systems. In: MITRE Technical Report TR-3153 (1977)
16. Buza, D., Juhász, F., Miru, G., Félegyházi, M., Holczer, T.: Cryplh: Protecting smart energy systems from targeted attacks with a plc honeypot. In: SmartGridSec (2014)
17. Crandall, J., Chong, F., Wu, S.: Minos: Architectural Support for Protecting Control Data. In: Transactions on Architecture and Code Optimization (TACO). Volume 3, Issue 4 (Dec 2006)
18. Koltys, K., Gajewski, R.R.: Shape: A honeypot for electric power substation. Journal of telecommunications and information technology (2015)
19. López-Morales, E., Rubio-Medrano, C., Doupé, A., Shoshitaishvili, Y., Wang, R., Bao, T., Ahn, G.J.: HoneyPLC: A Next-Generation Honeypot for Industrial Control Systems, p. 279–291. Association for Computing Machinery, New York, NY, USA (2020)
20. Newsome, J., Dong, D.: Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In: Proceedings of the 12th ISOC Symposium on Network and Distributed System Security (SNDSS). pp. 221–237 (February 2005)
21. Peter, E., Schiller, T.: A practical guide to honeypots. https://www.cse.wustl.edu/~jain/cse571-09/ftp/honey/index.html
22. Portokalidis, G., Slowinska, A., Bos, H.: Argos: an Emulator for Fingerprinting Zero-Day Attacks. In: Proceedings of ACM SIGOPS Eurosys 2006 (April 2006)
23. Provos, N.: A Virtual Honeypot Framework. In: Proceedings of the 13th USENIX Security Symposium. pp. 1–14 (August 2004)
24. Xiao, F., Chen, E., Xu, Q.: S7commtrace: A high interactive honeypot for industrial control system based on s7 protocol. In: Qing, S., Mitchell, C., Chen, L., Liu, D. (eds.) Information and Communications Security. pp. 412–423. Springer International Publishing, Cham (2018)