Deception in Dynamic Web Application Honeypots: Case of Glastopf

B. Mphago, O. Bagwasi, B. Phofuetsile, and H. Hlomani
College of Information Communication and Technology
Botswana International University of Science and Technology
Palapye, Botswana

Abstract— *Websites contain critical information to both the organization and the customers. With the cyber security threats currently on the rise, websites are getting easily compromised which prompts administrators to find ways to secure them from the black hat community. The use of honey pots as an alternative to other methods of securing web sites had brought with it advantages as well as disadvantages. Glastopf as one of the web application honeypots brings with it features that emulates a real server. It replies to the attack using the response that the attacker is expecting from his attempt to exploit the real server. However, Glastopf has its disadvantages too. Once deployed, Glastopf can be easily identified by the attackers due to the simplicity and static nature of its web-page templates. This paper seeks to improve the camouflage nature of the honeypot by proposing a new frame work which will produce dynamic web pages that will completely disguise the fake pages from the users.*

Keywords: honeypot security, deception, dynamic web pages, cyber attack

1. Introduction

Honeypots have emerged as great tools in tracking hackers and learning their attack methods. Traditional security tools such as firewalls, intrusion detection systems, and proxy servers, and as such, a tool that learns hacking mechanisms becomes evidently important to the security community. Many honeypots designs have been proposed and some of their configurations have been a challenge in many aspects [1]. The main intended purpose of a honeypot is to deceive an attacker by re-directing him/her to spoof hosts lines that will provide phoney information that appears to be informative or important after an analytic process was done on the attack signature. This suggest that the honeypot needs to be as dynamically deceptive as possible in all aspects in-order to achieve its goal of making attackers believe they have managed to gain access to real system, and most of honeypots are unable to achieve dynamicity to conceal the fact that they are honeypots, Glastopf being an example hence failing its primary task as a honeypot.

2. Background

As part of changing technologies, a honeypot can be involved in different aspects of security such as prevention, detection, and information gathering [2]. Lance Spitzner, defined a honeypot as a security resource whose value lies in being probed, attacked, or compromised [3], and for the purpose of this paper, we will adopt this definition mainly because it covers all the aspects that we believe a honeypot should be. A web application honeypot (WAH) in particular, is a basic web server with an attack surface [4]. This attack surface is the public HTML content which is indexed by search engines, and it contains links to files with known vulnerabilities.

Honeypots can be classified according to their purpose (research and production honeypots) and the level of interaction (low, medium, and high). A research honeypot is designed to gain information about black-hat community and does not add any direct value to an organization [3]. They are used to gather intelligence on the general threats organizations may face, allowing the organization to be better protected against those threats. A production honeypot is one used within an organizational environment to help protect the organization and mitigate the risk. Honeypots deployed in a production environment serve to alert administrators to the potential attacks in real time [5]. Low interaction honeypots are primarily production honeypots that are used to help protect a specific organization [3]. They only simulate services that cannot be exploited to gain total access to the honeypot. Medium interaction honeypots are slightly more sophisticated than low interaction honeypots but less sophisticated than high interaction honeypots [2], and like low interaction honeypots, they do not have an operating system installed, but simulated services are more complicated technically. High Interaction honeypots are actual systems with full-blown operating systems and applications. They are the extreme of honeypot technologies.

Currently, there are five major web application honeypots that are published and made available to the security community: HIHAT [6], DShield Web Honeypot Project [7], Google Hack Honeypot [8], PHPHoP [9], and Galstopf, being the most recent and the most sophisticated ever produced by The Honeynet Project. The first four honeypots above have one

thing in common: all of them use modified templates from real web applications to pretend that they are vulnerable and attractive to attackers. Thus, all of these honeypots are static, meaning you have to write new templates to support new vulnerabilities, and this can be time consuming and is a reactive process. However, the advantage to the template is that the honeypot looks very similar to a real victim and eventually will entice more manual and more complex attacks. The static limitation of this honeypots led to the development of another web application honeypot (Glastopf), which is a dynamic low-interaction web application honeypot capable of adapting to new and changing environments, thus making it a more reliable web application honeypot. The second reason that led to the development of Glastopf was the limited ability of the previously mentioned honeypots to deal with multistage attacks [9]. However, despite all the goods and praises about this honeypot, Glastopf still have some limitations in some quarters, the most notable one being deception by its webpage templates.

3. Motivation

In this paper, we investigate the Glastopf's deceptive qualities when queried through specially crafted requests. Our main focus is on the webpage templates it supplies when queried. The avenue for this is to theorize on how Glastopf web application honeypot can better deploy and avail its webpage templates and still stay deceptive enough from the black-hat community. We discovered that as of current, the webpages supplied by Glastopf are too basic for experienced hackers to see that they are not coming from a real system. This may be due to, as stated by Cohen [10], that the creators of Glastopf (The Honeynet Project) were not directed so much at deception to defeat the attacker in the tactical sense as at intelligence gathering for strategic advantage, but rather, unlike most historic honeypots, the creators are dedicated more at learning about the tool, the tactics, and motives of the black-hat community and sharing lessons learnt. This is despite the fact that a honeypot that can not hide itself entirely loose its value once detected. Its analogous to a trap without camouflage, attackers will simply avoid it, and as such it will not serve its purpose. In order to perform its function, a honeypot has to avoid being discovered. Responses given by a honeypot need to ensure they mimic that of the original system well so as not to raise suspicion, and this is normally the most difficult part when dealing with an attacker who has intimate, expert knowledge of a service or particular protocol[11].

4. Conceptual Framework

This section discusses the theoretical framework advanced by this paper. This is discussed by first exposing the limitations in the current set of Glastopf and further detailing a framework to address the identified weaknesses.

4.1 The Current Status

Glastopf is a low-interaction, dynamic web application honeypot capable of emulating thousands of vulnerabilities to gather data from attacks that target web applications [6]. It accomplishes its goal by deceiving the attackers that it is a hosting application with a list of vulnerable paths/ scripts, often referred to as dorks by its developers. These dorks are published in search engines and crawlers, which are then indexed and included in search results that attackers collect when they search for vulnerable paths in the web. So, when an attacker finds these published paths in search engines, he will, most probably, attempt to perform an attack on them, among other vulnerable paths that were not listed in the search engine. When these new vulnerable paths are detected, they will also be advertised and indexed in the search engines, thereby attracting new attackers looking for those paths.

Now, in the current situation, the webpage templates supplied by Glastopf when queried are static and too basic for experienced hackers to see that they are not legitimate web pages from a legitimate web-server. What happens now is when you install Glastopf you find templates in the data file within the honeypot, that can be customized to suit your needs. This means once customized, the template becomes static and does not necessarily provide everything the attackers wants to see.

4.2 The Proposed Framework

In this paper we therefore propose a framework which will provide a more secure way to camouflage webpage templates from the black hat community by introducing dynamic web pages as a replacement to the current static web pages. We employ a Content Management System, Wordpress to be precise, in designing these templates, which will give them a more realistic look that they are real web pages. Once the honeypot determines what the hacker wants, it emulates a response, either being the webpage itself or vulnerable scripts within the page, and send it back to the attacker. In our proposed solution, we build another module that tells the honeypot to auto-populate the templates with the information the hacker wants to see. Fig. 1 depicts the proposed model overview of how the dynamic web page can be integrated to the honeypot. This typically starts with an attacker sending a query to the honeypot (1). Whereupon the honeypot determines what kind of an attack this is and sends a request for a web page template to the template distribution module (2). The template distribution module then communicates with the template population module, giving it information on how to populate the required webpage template (3). The population module then interfaces with the inference engine and the knowledge base for the creation of the template (4). The inference engine makes use of the dummy database , UI Elements and Vulnerable code to create the desired template

(5). The reverse of these steps end up with a template that was requested by the honeypot being sent to the attacker.

An attacker normally sends a request to the honeypot with the belief that they are attacking a real system. The honeypot with its intelligent ability to process requests sends a request to the template population module. This is an expansion to the already existing mechanism of processing requests and returning the dummy webpages to the attacker. At the moment, the web pages generated by this honeypot are static, this is the main loophole that makes the honeypot less deceptive, and in the proposed frame work we seek to address this issue by building the template with a CMS and then build another module that directs the honeypot to populate the template with what the attacker is expecting to see.

The principle behind Glastopf is that a reply is sent to the attacker using the response the attacker is expecting to see from his attempt to exploit the web application. Our query processor which is made of two parts, the inference engine and the knowledge base lies as a middle framework between the template population module and the web page content. Once an attacker makes a request to the honeypot, the honeypot already knows the kind of response to give back to the attacker. Before giving the response to the attacker, the Query processor is responsible for auto populating the web template created using a CMS like word press. The inference engine acts as an expert system which contains rules responsible for mapping the page design commands with retrieval of page elements from the UI Elements store. The page elements which are prone to exploits are mixed with the dummy data and other elements are combined using rules defined in the inference engine. The knowledge base provides the facts that are known to the world and these facts are dynamically added to the page template as defined by the rules in the inference engine. After the Query processor successfully creates a dynamic web page, it is sent back to the honeypot containing elements which have been requested by the attacker.This process is depicted in Fig. 2.

5. Discussion

In this section we discuss and compare Glastopf with one of the closely related adaptive web application honeypots concepts, Heat Seeking Honeypots, and highlight on the limitations depicted by both of the honeypots. The actual reason for the comparison of the two honeypots is based on the fact that they are the two widely known adaptive and intelligent web application honeypots as of current.

Heat-seeking honeypots is not a honeypot per-se, but rather a framework in adaptive web application honeypots. These types of honeypots actively adapt to emulate the most popular exploits and construct pages that are similar to the ones targeted by attackers [9]. Heat-seeking honeypots have four components: first, they have a module to identify web pages that are currently targeted by attackers. Second, web pages are generated based on the queries the honeypot gets without manually setting up the actual software that is targeted. Third, these links are advertised through search engines and all the received attack traffic is logged. And finally, the honeypot logs are analyzed to identify attack patterns.

Fig. 3 summarizes the overall functionality of heat-seeking honeypots. First, attacker queries from the feed are searched using search engines, and the pages from the search results are collected. Then the pages are encapsulated and put on the heat-seeking honeypots, along with real software installed on virtual machines. The next stage would then be to advertise the pages in search engines and crawlers. In this case, when attackers issue similar queries to search engines, the honeypot pages are returned in the search results, and then the interactions with the attackers are logged for further analysis.

Just like heat-seeking honeypots, Glastopf is also an adaptive and intelligent web application honeypot; Both Glastopf and Heat-seeking honeypots have vulnerable scripts in their webpages which are advertised in the search engines for the attackers to attack. Unlike heat-seeking honeypots where vulnerable pages similar to the ones attackers are looking for being fetched from the web, in Glastopf, the webpage templates are not fetched from the web, but rather, they are pre-installed in the honeypot itself as static web page templates. These templates can be customized by Glastopf user according to their own specifications during installation, and be fed to the attackers when queried.

The authors believe that the addition of these dynamic auto populated web templates will fully give the honeypot what it is lacking at the moment, complete deception.

6. Conclusion

Honeypots provide a very good starting point of protecting web applications against malicious attackers. They accomplish this by hiding themselves from the attackers and respond as if they are a compromised system, and by so doing , they learn how an attacker is exploiting it and reports back to the security administrators who can make use of this information to secure their web applications. Glastopf is the only adaptive and intelligent web application as of current, which is believed to be the future of web application honeypots. However, Glastopf also lacks in the aspect of deception as it provides basic and static web pages which experienced hackers can easily identify. The improvement of this feature would be the production of dynamic web pages which will completely emulate real web pages from a real system, and as such the attacker has less chance noticing that they are interacting with a honeypot rather than a real system. The success of the implementation of the proposed framework will not only be a breakthrough to combat cyber security but will also serve as a platform for greater innovations in the security industry as a whole.

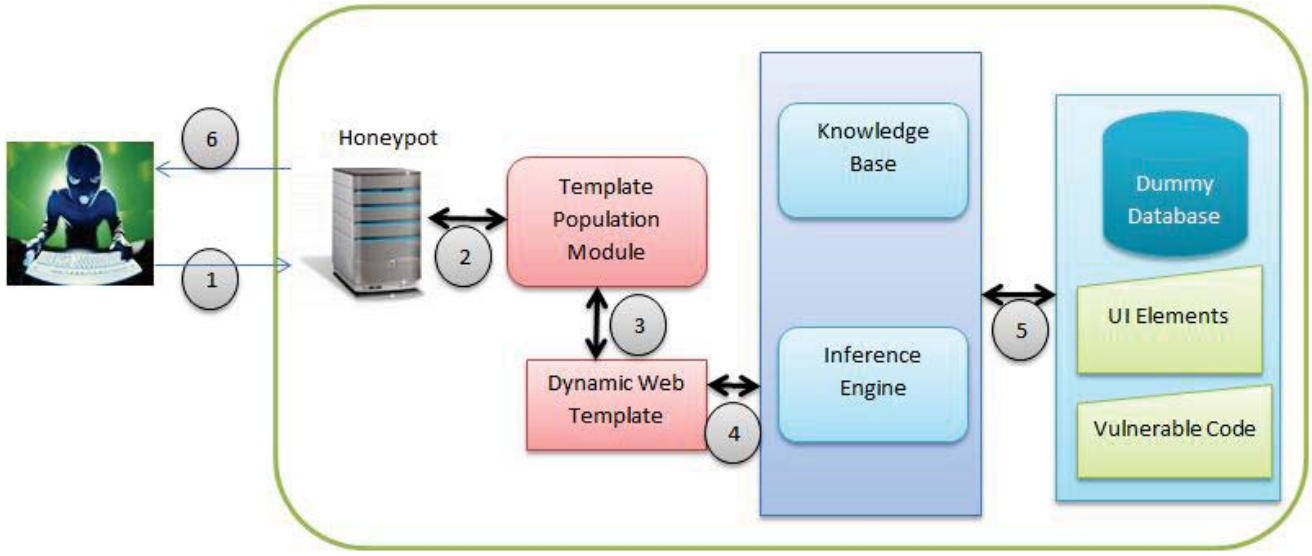

Fig. 1: Generic architecture for dynamic web page generation

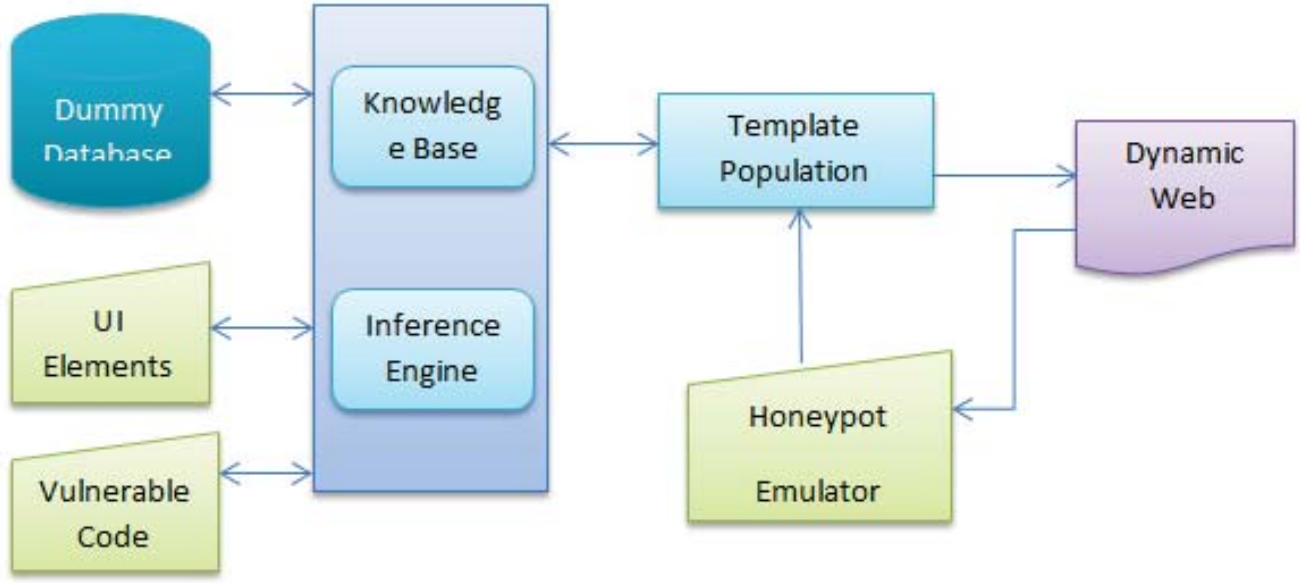

Fig. 2: An overview of the proposed model for dynamic web page integration into the honeypot

7. Future Works

In this paper, we discussed theoretical ideas on how Glastopf webpage templates can be improved and made dynamic in order for them to stay deceptive. We introduced Word-press to build these webpage templates and then discuss auto template population module for populating our templates. In our next paper, we will extend these ideas by employing empirical analysis and design of the proposed ideas. The idea is to create a python script that will direct the honeypot on what to populate the webpage template based on what the attacker expects to see. Then we would measure and analyze the response patterns of our content population module, hence compare it with that of a normal web server. This would basically give us an understanding that our proposed ideas have been a success.

8. Acknowledgment

Our research activities are sponsored by the Botswana International University of Science and Technology (BIUST) and hence we would like to extend our gratitude to the university for affording us the opportunity to contribute to the body of knowledge web and cyber security.

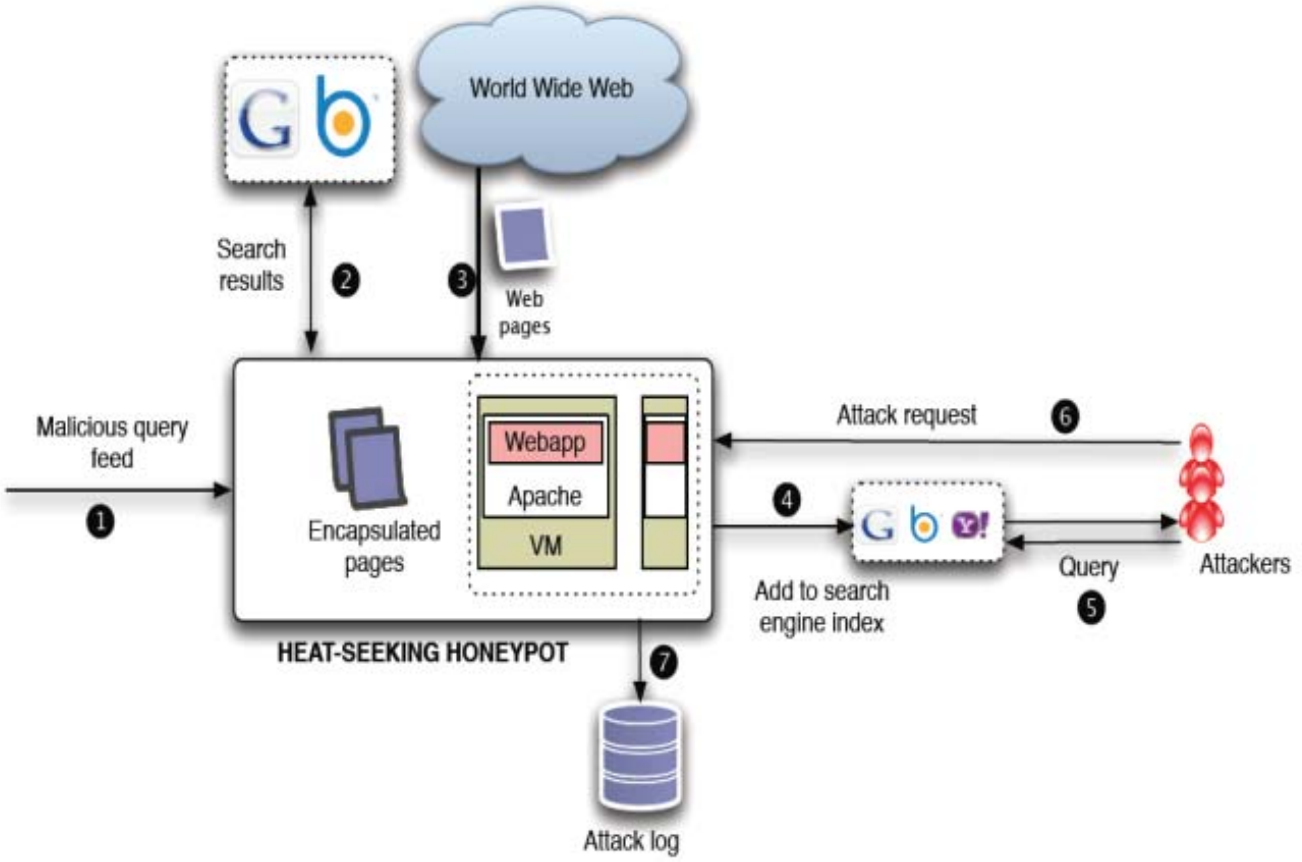

Fig. 3: Heat Seeking Honeypots Functionality

References

[1] G. Wagener, S. Radu, E. Thomas, and A. Dulaunoy, "Adaptive and self-configurable honeypots," in *12th IFIP/IEEE International Symposium on Integrated Network Management*, Luxembourg, 211, pp. 345–352.

[2] I. Mokube and M. Adams, "Honeypots: Concepts, approaches, and challenges," in *Proceedings of the 45th Annual Southeast Regional Conference*, ser. ACM-SE 45. New York, NY, USA: ACM, 2007, pp. 321–326.

[3] L. Spitzner, *Honeypots:Tracking Hackers*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[4] Honeypot Project. (2012) Cyber fast track: Web application honeypot. [Online]. Available: https://honeynet.org/files/CFT-WAH-FinalReport.pdf

[5] M. Gibbens and R. Vardhan. (2012) Honeypots. [Online]. Available: http://www.cs.arizona.edu/ collberg/Teaching/466-566/2013/Resources/presentations/2012/topic12-final/report.pdf

[6] HIHAT. (2007) High-interaction honeypot analysis tool. [Online]. Available: http://hihat.sourceforge.net/index.html

[7] DShield Web Honeypot Project. DShield Web Honeypot Project. [Online]. Available: https://sites.google.com/site/webhoneypotsite/

[8] R. McGeehan, G. Smith, B. Engert, K. Reedy, and K. Benes. GHH, The Google Hack Honeypot. [Online]. Available: http://ghh.sourceforge.net

[9] L. Rist, S. Vetsch, M. Koğin, and M. Mauer. (2010) Know your tools: A dynamic, low-interaction web application honeypot. [Online]. Available: https://www.honeynet.org/sites/default/files/files/KYT-Glastopf-Final_v1.pdf

[10] F. Cohen, "The use of deception techniques: Honeypots and decoys," *Handbook of Information Security*, vol. 3, pp. 646–655, 2006.

[11] S. Innes and C. Valli, "Honeypots: How do you know when you are inside one?" in *Australian Digital Forensics Conference*, 2006, p. 28.