Honeyweb: a web-based high interaction client honeypot

Sainath Patil Assi.Prof.INFT.DEPT,
Nageshri B Karhade, Yogini K Kothekar
Vidyavardhini's College Of Engineering And Technology,
Mumbai University,
India
patil_sai@yahoo.co.in
nageshri@msn.com, kothekar.yogini@gmail.com

***Abstract*— Honeypots are closely monitored decoys that are employed in a network to study the trail of hackers and to alert network administrators of a possible intrusion. Using honeypots provides a cost-effective solution to increase the security posture of an organization. Even though it is not a panacea for security breaches, it is useful as a tool for network forensics and intrusion detection. Nowadays, they are also being extensively used by the research community to study issues in network security, such as Internet worms, spam control, DoS attacks, etc. In this paper, we advocate the use of honeypots as an effective educational tool to study issues in network security. We support this claim by demonstrating a set of projects that we have carried out in a Websites, which we have deployed specifically for running various web applications' under supervision . The design of our projects tackles the challenges in installing a honeypot in organizational website, thus determining various security compromises that are performed on it over the Internet by attackers/hackers. In addition to a classification of honeypots, we present a framework for designing projects for web application security courses.**

***IndexTerms*- Honeypot, honeypages, honeytokens**

I.INTRODUCTION

A honeypot is a security technology that provides organizations with a way to catch viruses, malware or attackers, as well as acting as an alarm system that can discover attempts to attack a network. Honeypot technology is defined as a 'security resource whose value lies in being probed, attacked or compromised' .There are two main types of honeypot: passive and active. The passive honeypot is a technology that passively waits for attacks in order to detect them, while the active honeypot, also called a client honeypot, interacts with a target web page to identify and determine its potential effect on the browser or operating system.

A. ADVANTAGES

- Honeypots are a valuable source of attack information, because they allow to observe the attack methodology from the active information gathering phase to the real intrusion and track covering. The possibility of analyzing the modus operandi of the attacker along with the discovery of the new attack tools are crucial means to be able to deploy the adequate protection safeguards. This attack visibility motivates the adequate investment in security prevention mechanisms, because it points out the real threat status of a network.

- Another benefit is that the data gathered by a honeypot is by default illegitimate minimizing the false positive rate of other security mechanisms, like for example an intrusion detection system. Comparing with intrusion detection systems, honeypots leave time for focusing in the real threats and the network administrators are not bothered with false attack event showers. Attacks against honeypots are system targeted, which provide the possibility of gathering detailed information even if the traffic is encrypted up to the endpoint. This enhances the possibility of catching insider threats due to the difficulty of identifying internal illegitimate behavior.

- The resources needed for running a honeypot are minimal, because it captures mostly attack traffic that is expected to be minimal when compared to normal traffic. It does not behave in promiscuous mode, because the traffic gathered is delivered directly to him, so it requires little network resources. Being a non-critical asset, the removal of a honeypot from the network usually does not influence the existing infrastructure and it can be analyzed, reinstalled and added later with no impact in availability of other systems.

- These deception mechanisms serve as bait to the attacker while the critical assets are further defended during the security incident response procedure against new forms of attack detected in the honeypot.

B.DISADVANTAGES

- Honeypots give a limited vision about attacks, because they analyze only the network segment in which they listen. This direct limited attack vision might impact conclusions leading to a false sense of situational awareness.

- The identification of the honeypot by the attacker using fingerprinting techniques will limit further attacks and actions against that lure system. If a virtual honeypot is used, the attacker will try to escape the honeypot and intrude the host system or hypervisor that provides the virtualization structure sandbox.

- The high interaction honeypot can be used to attack other systems and to distribute illegal information such as spam, so it requires higher maintenance and monitorization. On the other hand the low interaction honeypot is unable to capture malware and attacking tools as well as multiple phases.

C.LOW AND HIGH INTERACTION HONEYPOTS

The simplest form of a honeypot is a real vulnerable system that has been modified to include surveillance methods. Such a system is called a high-interaction honeypots because the attacker is able to fully interact with the honeypot just like a real system. This offers the best potential for analyzing all aspects of an attack, but also introduces risk that the attacker will use the capabilities of the system to attack others. A high-interaction honeypot must disguise itself as a real machine, hiding its surveillance methods to all users even if they have root privileges. This is usually done using very risky and resource intensive techniques like full system emulators or root kit-type software as in the GenIII honeynet. To monitor automated attacks as for example those performed by autonomously spreading malware, such effort is not always required. So called low-interaction honeypots over limited services to the attacker.For example by emulating only those parts of a service which are vulnerable. Low-interaction honeypots can typically be deployed with fewer resources because they are not fully offering the expected services and they also incur less risk. However, it is more likely that the attacker will cut short the attack before useful information can be learned either because the system does not support functionality needed for the attack or because the attacker suspects the system is a honeypot. A popular example of this kind of honeypots is honeyd , which is very easy to deploy (at least in comparison to a high-interaction honeypot).

II. RELATED WORKS

Threats to web applications have been previously investigated using intrusion detection systems and classical honeypots and a lot of information about attack techniques has been gained. Honeypots generated with our approach aim at collecting more such information using less resources.

Our approach is not the first tool in the field of web application honeypots.GHH (the Google Hack Honeypot) is a project within the Honeynet project which creates a vulnerable web application from scratch. The main focus of GHH (and of its descendent PHP.Hop) is to collect data about the search patterns attackers use to identify vulnerable applications. In this sense (and in contrast to our approach), GHH is a low-interaction honeypot and does not provide real web application functionality.

Another related area is that of so-called hitlist worms. A hitlist worm can use (among other information sources) search engines to collect large lists of vulnerable machines before spreading. Especially worms that target web applications are very dangerous and must be investigated before they spread . Typically, these search worms can only be observed by search engine operators or victims. Honeypots created by our framework can be used in this endeavor: Essentially, we become part of the hitlist and are thus able to learn more about.

III. CHALLENGES OF LOW INTERACTION CLIENT HONEYPOTS

Honeypots help security researchers to study the techniques and objectives of web-based malware. However, there are still some challenges to be overcome to allow their full benefits to be achieved:

- *IP tracking*: this technology is widely used in web-based tools to track the IP address of visitors. For example, according to Seifert the Mpack tool tracks visitors' IP addresses and only attacks the visitor with malicious scripts after a number of visits to the website. If a client honeypot tries to visit a malicious website running the Mpack tool with the IP tracking feature enabled, it will not detect any malicious behavior and may assume the site is clean, but if the client honeypot visited it a number of times, the IP tracking feature will trigger the malicious behavior. Therefore, the challenge is to solve this problem to prevent false negatives.
- *Geolocation-dependent*: this feature, provided by a number of malware tools, will cause the malware only to affect visitors from certain countries, while behaving normally with visitors from other countries.

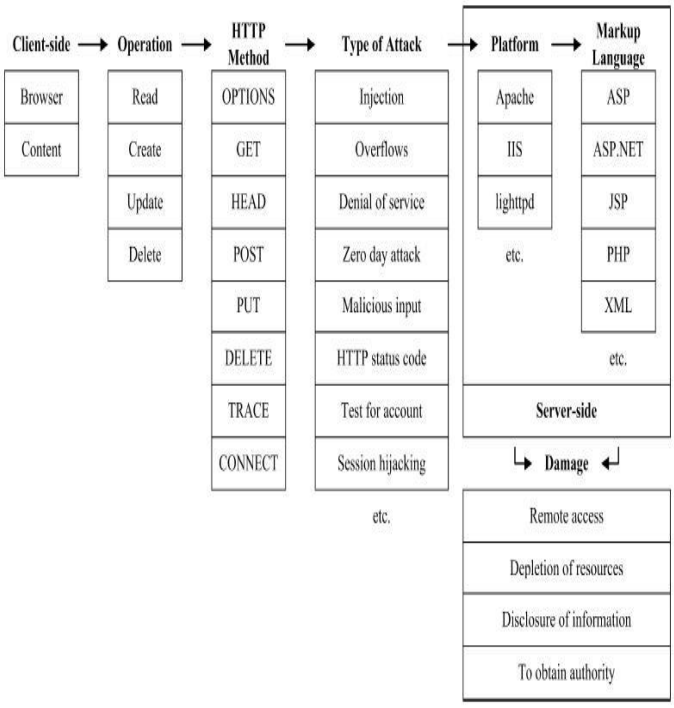
IV.WEB ATTACKS

Web attacks increase everyday as more and more enterprises deploy their business using web applications. These web applications are no longer based on static HTML or a single dynamic page provided by a CGI, but are composed of dynamic content that enables other forms of interaction and customization via programming languages such as PHP, Java or .NET and different web server engines.

- SQL Injection-Nowadays most of the websites have databases to support the backend data storage and SQL injection takes advantage of the communication between the website and the database to insert the attack. SQL injection consists on being able to run commands on a database by inserting interpreted SQL statements in the client input data. With this behavior the attacker is able to read the fields of the database directly if he knows its structure or fingerprint the database looking for existing tables forcing errors with debug information. If no useful output is provided by the website, the attacker can use response timing duration or true/false queries to infer database information. As databases typically run with high privileges when compared to the normal operating system user, the attacker might accomplish remote operating system command execution.

- Cross Site Scripting-Cross site scripting happens when the web server interprets malicious HTML or Javascript code supplied by an attacker as a parameter, including it as part of the page response to the browser of a legitimate user. This allows transferring user private information or redirecting the user to a malicious site taking advantage of the trust that the user deposits on the legitimate accessed site, although the site is not trustworthy due to this vulnerability. These sorts of attacks that affect the client's browser are called client-side attacks.

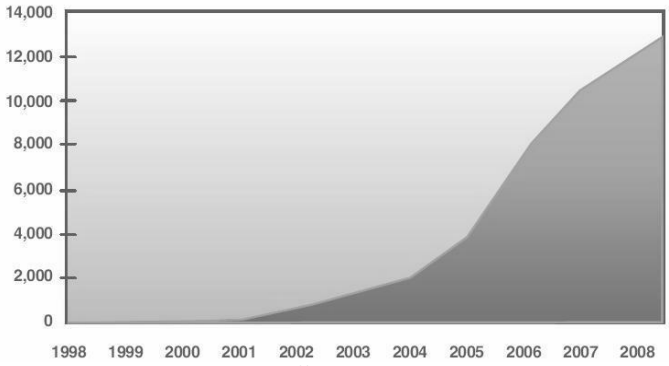
- Remote File Include-Remote file include allows attackers to execute malicious code residing on an external webserver on a vulnerable website. This causes the vulnerable website to implicit trust the malicious code interpreting it as an internal configuration or plugin and executing its actions.



Taxonomy of attacks

A.STATISTICS

One might wonder what makes web attacks so critical and differentiates them from other attacks. The following statistical information provides an insight regarding this subject by presenting real information about the security risk of web attacks.



Vulnerabilities affecting Web applications

Vulnerability Type	2007	2008	2009
Cross site Scripting	67%	67%	65%
Information Leakage	36%	45%	47%
Content Spoofing	26%	28%	30%
Insufficient Authorization	14%	18%	18%
SQL Injection	17%	16%	17%
Predictable Resource Location	22%	15%	14%
Session Fixation	0%	0%	11%
Cross Site Request Forgery	0%	10%	11%
Insufficient Authentication	16%	10%	10%
HTTP Response Splitting	0%	9%	9%
Abuse of Functionality	11%	8%	0%
Xpath Injection	3%	0%	0%
Directory Indexing	5%	0%	0%

Evolution of detected Web vulnerabilities

B. OBSERVED ATTACK TRAFFIC

Source	Number of Hits	Percentage
No parameter	1203	14.72%
HTTP GET	6874	84.07%
HTTP POST	99	1.21%
HTTP COOKIE	826	10.10%

Source of transferred data targeting the web-honeypot

In 14.72% of the hits, just the webpage itself was requested without transmitting further parameters to the application. When parameters were transmitted, the HTTP_GET method was used in 84.07% of the cases. Although only few requests were using the HTTP POST method, these hits contained a comparatively high number of attacks that were detected. Almost all these POST requests were malicious in nature.

Orthogonal to this, the request can also use cookie parameters in addition to GET or POST requests. This additional cookie data was transferred in about 10% of the requests. Interestingly, we were not able to observe any request which was trying to use HTTP cookies to perform an attack. Of course, this may change with a longer observation time or a different set of applications that is deployed.

Name of Attack	Number of Attacks	Percentage
SQL-Injection	41	58.57%
File-Inclusion	13	18.57%
Command-Injection	8	11.43%
Directory-Traversal	3	4.29%
Others	5	7.15%

Distribution of types for successful attacks

It displays the distribution of the different attack types. SQL injections build the vast majority of the attacks, together with remote file inclusions they represent nearly 80% of all attack types we observed on our honeypots. The main reason for the high percentage of SQL injections probably lies in the selection of modules we deployed.

C.SAMPLE ATTACKS

- Command Injection Example. We start with a brief example of a command injection .The following HTTP GET request was monitored.

name=Forums
highlight=%2527.$poster=%60id%60.%2527

The variable highlight is not filtered correctly. In the second line of the request we see that the attacker tries to inject and execute the command "id" into that variable. The command identifies the current user and denotes a typical test done by attackers in order to check a system for suitable command injection vulnerability.

- File Inclusion Example.The next example deals with a typical remote file inclusion and explains the tool attackers tried to download and use on our honeypots.

The attack itself consisted of just a single (sanitized) request:

/phpBB/includes/functions.php?phpbb_root_path=http://XXX/c99.txt

The attacker attempts to use a vulnerability in the phpbb_root_path variable in order to download the _le c99.txt from a remote server and include it in the web application. A copy of the file was automatically captured and stored in the database which allowed us to analyse its content. c99.txt is actually a PHP script which allows an attacker a web-based backdoor to a
compromised machine. Via this script, the attacker can for example create files, execute arbitrary commands, or list files and directories.

- Self-Propagation Example. As a third example, we show a more complex attack, involving different tools and file downloads. The attack started with a single HTTP GET query using the following (sanitized) request:

/phpBB/includes/functions.php?phpbb_root_path=\%20\%22powered\%20byhttp://XXX/j0.gif?\&add=bot

This denotes an attack against the PHP bulletin board: an attempted file inclusion attack targeting the variable phpbb_root_path in the file functions.php. The attack tried to include the file j0.gif from a remote location. Again, we retrieved a copy of this _le automatically via Honeyweb. This file turned out to be a PHP-based shell utility. The utility supports all basic operations like file listing, changing of permissions, command execution, and file upload. Moreover, it includes a mechanism for self-propagation. This mechanism

is activated once the shell is executed the first time. At this time, it tries to download and execute a second file, named spread.txt. The second file has only one purpose: it attempts to fetch and execute a copy of a third file called fast.txt. It uses fifteen different commands, download locations, and options to get a copy of fast.txt. Once the third file has been downloaded successfully, it is executed.
This file contains a so called IRC bot. An IRC bot is a program that connects to an Internet Relay Chat (IRC) server and typically allows to automate some of the IRC functions.

V. HONEYWEB FEATURES AND SCOPE

A. FEATURES

- Automatically scans for known attacks.
- Detects SLQ-Injections, (Remote) File-Inclusions, Cross-Site Scripting (XSS).
- Provides an overview mode which allows you to look and scan for new incidents quickly (semi-automatic mode).
- Supports detailed information about all data correlated with every access to the honeypot. This includes but is not limited to HTTP-GET, HTTP-POST and COOKIE data.
- Saves copies of malicious tools in a secured place for later analysis.
- Generates numerous statistics about all traffic recognized at the system.

B.SCOPE

- This project will be focusing on server side programming language specifically PHP which are mainly used for development of CMS – content management systems, e – commerce, social networking site, etc.
- The project acts as a Service Provider for Honeypot security to various websites.
- The project is proposed to trace the user activities on the client site – demo websites.
- Various attacks performed on the demo sites will be traced within the tool. Thus providing a competitive analytical & statistical data so as to find the loop holes of the demo sites.
- This project can be deployed at on the same server as in where the demo website is located or can also work with remote web server
- This project will be using PHP, JavaScript, & my SQL.
- With a user-friendly environment, it will diminish the problem of reading long and unstructured log files and efficiently captures what has happened inside the virtual honeypots.

VI. HONEYWEB

Honeyweb allows to transform arbitrary PHP applications into web-based high-interaction Honeypots. Furthermore a graphical user interface is provided which supports the process of monitoring the Honeypot and analysing the acquired data.

A typical use could be the transformation of PHPMyAdmin into a full functional Honeypot, which offers the complete functionality of the application to the users but performs comprehensive logging and monitoring in the background.

Honeyweb Architecure

Our basic approach is to start with an existing web application and convert it into a honeypot in an automated and generic way. This involves adding capabilities for logging important data about an attack and containing the existing application within a honeynet to protect others.
Most of the prevalent web applications are written in PHP. Thus, we chose to focus on PHP based web applications. Nevertheless, the ideas presented in this paper could also be applied to web applications written in other programming languages used on the web.
To automatically identify the data we will log, we begin by observing that all traffic coming to a web-based honeypot will use HTTP. This protocol provides two basic transmission methods:

- The GET method means that form data is encoded into the uniform resource locator (URL) by the web-browser. This method is commonly used when the form processing is

idempotent in such a way that no status changes will apply by performing the request. The maximum amount of data that can be transferred with a single GET request is limited and depends on the maximum size of a URL. For example, Microsoft Internet Explorer has a maximum URL length of 2,048 characters, minus the number of characters in the actual path.

- The POST method describes a procedure to transmit data that is meant to be used for non-idempotent queries. Every request that results in a status change is non-idempotent. In contrast to GET requests, form data appears within the body of a message when using POST requests. Typical examples for such non-idempotent requests using POST are _le uploads or sending emails. The amount of data transferable via POST is larger and theoretically unlimited. In practice, however, the maximum length depends on the settings of the web server.

In order to log the information an attacker enters into a web application, we need to track these two transmission methods. This can be achieved by monitoring four crucial arrays, which are provided by PHP within a global scope:

1. $_SERVER: This array contains the main server information such as headers, paths, and script locations. The entries within this array are created by the web server. There is no guarantee that every web server will provide all of them, servers may omit some, or provide others. However, a large number of these variables are part of the CGI 1.1 specification, thus it is reasonable to expect those.

2. $_GET: This array contains all data transferred to the server via HTTP GET requests. This type of requests typically includes data like session IDs or path information, referring to clicks of the user inside the web application.

3. $_POST: This array contains all data transferred to the server via HTTP POST request. These requests are used for similar purposes as the GET request. The differences between both types of requests were outlined above.

4. $ COOKIE: This array contains all data transferred to the server via HTTP cookies. Web applications typically use cookies in order to store data like configuration settings and session information, but also login information like username and password can often be found here.

These arrays contain all information that is needed to track every step of an attack against an arbitrary web application. If we thus monitor all these arrays and correlated the collected data, we are able to monitor the exact attack traffic for any PHP-based web application.

Different automated means are provided which facilitate a user who is monitoring and analyzing the acquired data. The combination between automatic data preparation by the tool and manual monitoring by a person ensures the highest detection rate for attacks and interesting incidents and the lowest rate of false positives. Furthermore, only a reasonably low amount of effort for the user is required for monitoring and analysis due to the high level of automatization. In terms of data analysis, the chosen design approach, which allows arbitrary web applications to be transformed, now results in several challenges. Thus, different problems and issues have to be considered in the design of Honeyweb.

The tool should display each access that was made to the honeypot web application. For example, this includes every single click within the application and every entry in a form. Thus the logging results in a vast amount of data. Nevertheless, all the information needs to be presented by the tool in such a way that the person who is monitoring the honeypot can quickly overview the information and extract the important data. Furthermore, the importance and impact of data depends on the web application it originates from. In one application a variable called username may be very important and at high risk of being attacked, whereas in another application, the same variable could be negligible.

As arbitrary web applications are involved, it is not possible to focus on some set of variables. All kind of variables with different names, contents, and lengths have to be taken into account.

The tool should identify known attacks automatically as good as possible. An example would be the automatic identification of all SQL injection attacks. But again, even for known attacks, this is not trivial since arbitrary web applications can be involved: whereas in one case an application may use parts of SQL statements to perform normal operations, SQL commands may refer to an attempted SQL injection attack in another case with a different variable or application. While it may be possible to automatically identify all SQL statements in the data, the decision if a specific statement actually denotes an attack or is part of the application's normal behavior cannot simply be made automatically.

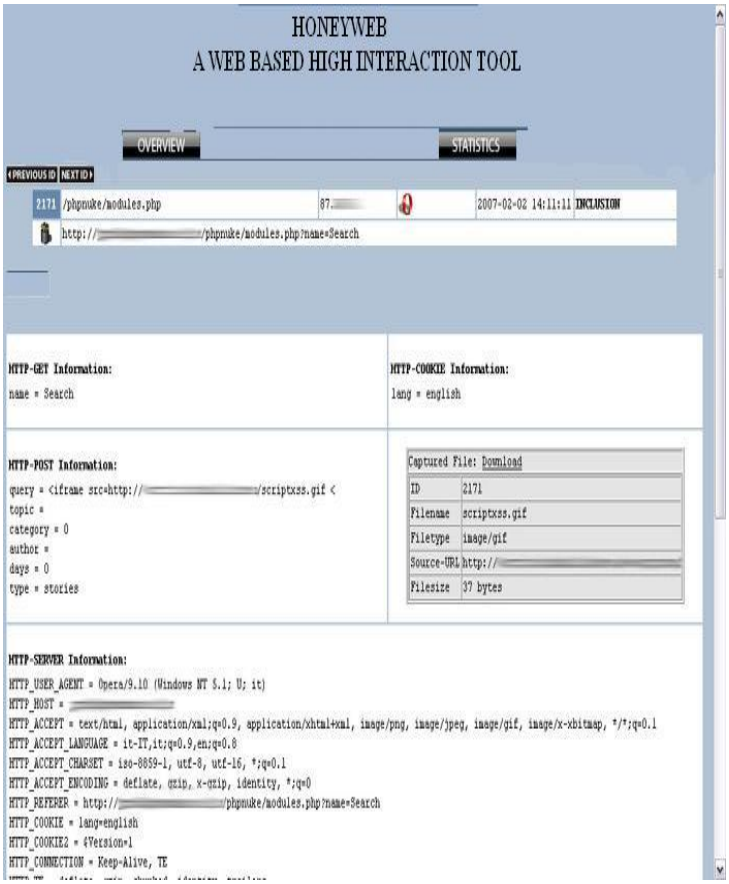
The tool should support the detection of new attacks as good as possible. Certainly not all new attacks can be detected automatically. Nevertheless the system could try to identify patterns, strings, or names that are more likely to represent an attack. This helps in identifying zero day attacks, i.e., attack vectors which are unknown at the time of attack.

As described before, the tool supports the person who is monitoring the honeypot and analyzing the collected data. The combination of automatic filtering by the system and human inspection is most likely to yield the highest accuracy in terms of information collection about attacks as well as the lowest rate of false positives. All issues identified above led to the following design decisions for Honeyweb. First, two main

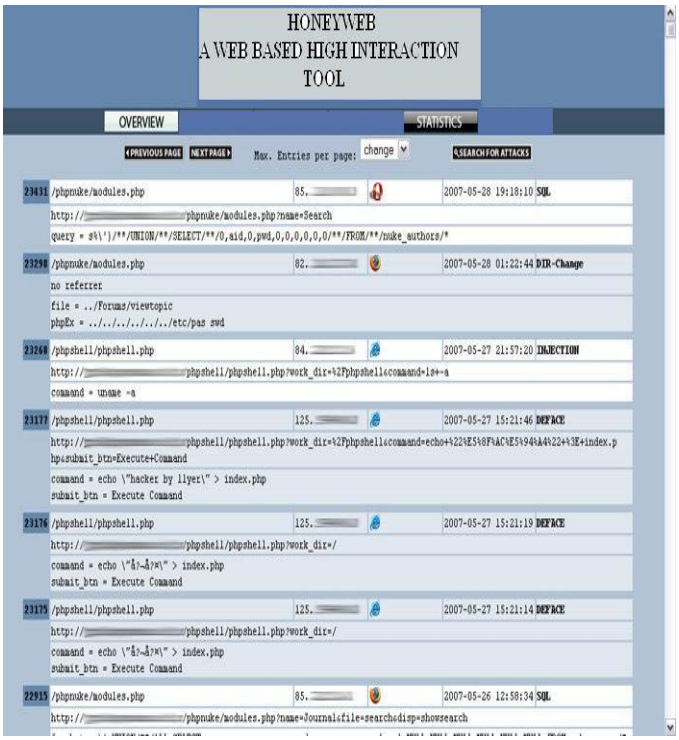
views are supported: On the one hand, there is the overview mode, which allows the user to get a quick general view about all the activity that was captured. On the other hand, the tool allows to switch into a detailed viewing mode, where all information about a single event is provided. The detailed view provides an overview of all data transferred to the web application via the four different arrays provided by PHP.

The tool automatically filters for attack patterns that can be derived from known attacks like SQL injection or file inclusion attacks. This is achieved via regular expression which we derived from an analysis of known attacks against web applications. For example, we search for patterns like INTO OUTFILE, script, or include which indicate possible attacks. Furthermore, we include generic attack patterns that identify common commands executed by an attacker after a compromise, e.g., the commands id or uname. The tool provides high extendibility because it supports the easy supplement of new patterns.

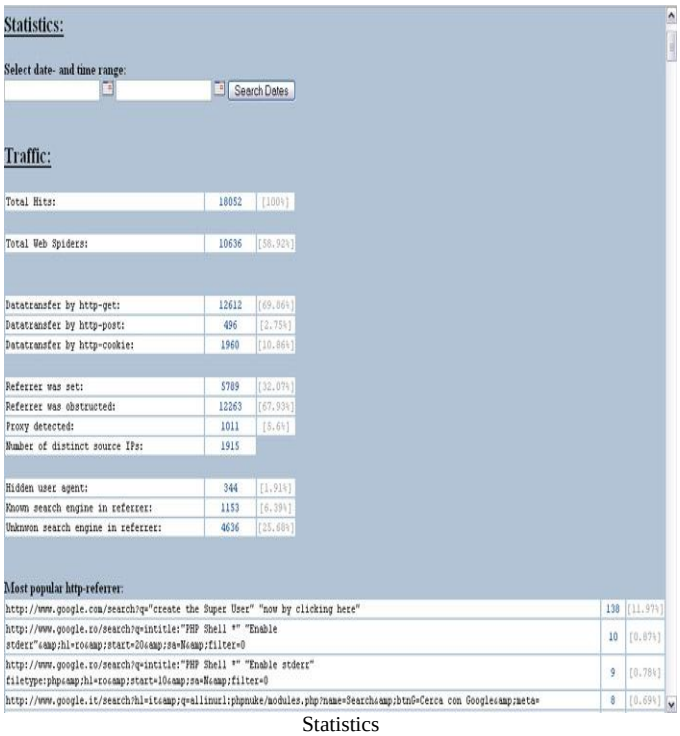
The overview mode helps the user to recognize attacks quickly and gain an impression about the traffic and the activities of a honeypot at a glance. When the user has spotted an interesting entry in the overview, he can access further information by switching to the detailed viewing mode. Honeyweb is also equipped with a search function in order to allow the user to quickly find the desired information and to facilitate the handling of large amounts of data. It can for example search for IP addresses, specific attacks, or date and time.



Detailed view



Overview



Statistics

VII.INCLUDED FEATURES

A.TRANSPARENT LINKING

In order to attract attackers to our honeypot, we need to catch their attention and interest. Since attacks on web-based applications commonly use search engines in order to find their victims, we want our honeypot to be listed by the indices of popular search engines. Once the honeypot is indexed, attackers that use search engines are drawn to the system, which results in more traffic being driven to the honeypot. Basically the trick is to become part of the hitlist.

The important question is how to add the honeypot to the index of a search engine. Nowadays, the search index is commonly constructed automatically with the help of so called web spiders. Web spiders are programs which crawl the World Wide Web in a methodical and automated manner with the intent of creating an index about the crawled contents. As search engines do not provide a method for directly modifying search results for such a research purpose, we need to use the behavior of the web spiders themselves in order to complement the search index with information about our honeypot. Specifically, we add links to our honeypot in existing, regularly crawled web pages.

There are two problems with this approach. First, the details about the exact behavior of the web spiders are usually kept secret, in order to avoid abuse or distortion. Neither the exact construction criteria for the ranking of the index are public, nor information about if and how the content of a web page gets rated. Some documents describe the basic principle of the algorithms, but the exact details are commonly not known. Secondly, we cannot place arbitrary links to our honeypot on a website. This is due to the fact that not only web spiders or attackers may follow the link, but probably also many benign users who are just visiting the webpage. By following the link, these users would cause false positives in our log files and incidentally also increase the chance that an attacker reveals the true purpose of this link for our honeypot.

In order to tackle these problems, we choose the following solution: a specially crafted link is required, which satisfies two requirements. First, it needs to be invisible to a benign Internet user surfing the web page. Second, it is still recognized by web spiders crawling the page. A link of this type is named transparent link since only web spiders can see it.

We have to keep in mind that transparent links represent an issue where web based honeypots strongly differ from traditional honeypots where every access is considered to be an illicit use of that resource. Instead, web-based honeypots need to be indexed by web-spiders in order to catch a reasonable amount of interest and to work properly as we explained above. Hence, in this point, web based honeypots pursue a different concept than other honeypots: we need to advertise the presence of the honeypot. Nevertheless, the main value for both types of honeypots lies in the unauthorized or illicit use of that resource.

B.HONEYPAGES

These are obscure web pages sprinkled in the web site. They have no legitimate purpose, nay they are not even linked from any valid page. Normal users would never reach these pages. However, we drop hints about these pages by embedding their url as comments or hidden fields in valid pages. While normal users would never see this, an attacker who analyzes the source code, or a vulnerability scanner that spiders the site would see these and follow the link. When the page is accessed, it points us to the intruder.

C.HONEYTOKENS

Honeytokens are fake records that are inserted in the database. These fake records are not expected to be used by normal users. If any of these honeytokens are used, they alert us of the database having been compromised. An example of honeytokens is fake username/passwords in the user database. These users do not exist in the real world, and hence are not expected to be logging in to the application. If the application sees these credentials being used, it immediately recognizes that the user database has been compromised.

File, web, or email servers can all have honeytokens embedded into them. Anything that has data can easily have additional bogus data added, bogus data that becomes our honeytoken. File servers could have bogus files, such as Word documents, .pdf files, or Excel spreadsheets. These files could have unique names, or unique tags embedded in the files. Intrusion Detection Systems can then have signatures customized to look for these honeytokens. If you see any honeytokens traversing your networks, you know you have employees (or someone on your internal network) accessing files they are not authorized to. For encrypted environments, kernel modules can be developed to monitor system files. If someone attempts to read() one of the files residing on the system, the kernel module can detect this unauthorized activity and generate an alert.

VIII.FUTURE WORK

Honeyweb was developed as a web-based high interaction client honeypot which integrate the web technology to client honeypot. Therefore, there are some future works which will be implementing in future versions of Honeyweb. The following points are the future for Honeyweb:

1. *Plug-in simulation*: Some malicious websites will exploit vulnerability within plug-ins installed on the web browser, such as Flash Player, RealPlayer and Adobe Reader. There are some exploits available for such plug-ins and therefore the attacker can check the visitor plug-in through some plug-in detectors and exploit the user if the valuable plug-in is present. An example of plug-in vulnerabilities is CVE-2009-0376 and CVE-2009-0375; these vulnerabilities affect RealPlayer version 11, allowing remote execution of an arbitrary code for an attacker to successfully exploit this vulnerability. However, if the attacker fails to exploit the previous vulnerability, then it will cause a denial-of-service condition.
2. *Intrusion detection system (IDS):* Honeyweb uses scan engines to scan the target server. In addition, the use of such IDS as Snort [23] will add an excellent feature to Honeyweb.
3. *Honeyweb Crawling*: The current version of Honeyweb supports two of the main search engines, Yahoo! and MSN, and it will be helpful if other search engines were added to Honeyweb to give the user the ability to search across different search engines. Search engines such as Google and Ask are examples that can be added to Honeyweb.
4. *Improve Honeyweb client*.

IX. CONCLUSION

In conclusion, we have presented the design and implementation of a generic toolkit for turning arbitrary PHP web applications into high interaction honeypots.
We have described a method for drawing attackers to our honeypot with transparent links.

In our case, we used this tool to deploy PHP applications with known vulnerabilities. This allowed us to study how often in what ways already known vulnerabilities are being exploited. We could instead have deployed the newest and most patched versions of these applications shifting the emphasis to monitoring for new exploits discovered in the latest software. Applying our logging
code is so simple and un-intrusive to apply that original application developers could consider use of this tool as a phase in testing their software.

One keys area of future work is to extend support to other programming languages popular for web development such as Javascript and Perl. Another is to make the limits on outgoing web traffic more dynamic. In order to protect other systems, we currently place relatively tight limits on the amount of outgoing web traffic an attacker can generate from our honeypot. However this can cut off the process of the attack before sufficient data is collected to completely analyze and understand it.
We would like the administrator of the honeypot to be able to write triggers to match attack patterns they have seen before and for which they want to allow incrementally more access in order to learn about the next stage of the attack vector.

X.REFERENCES

[1] L. Spitzner, (2002). Honeypots: Tracking Hackers. 1st edition. Addison-Wesley Professiona. ISBN-10: 0321108957.

[2] C. Seifert, (2007). Know Your Enemy: Behind the Scenes of Malicious Web Servers. [Online]. Available at: http://honeynet.org/book/export/html/181 [Accessed 1st Mar 2009].

[3] Laurent Oudot. PHP.Hop { PHP Honeypot Project. http://www.rstack.org/phphop/, February 2006.

[4] The Honeynet Project. Know Your Enemy: Learning About Security Threats. Addison-Wesley Longman, 2nd edition, May 2004.

[5] L. Spitzner. Honeypots: Tracking Hackers. Addison-Wesley Longman Publishing Co., Inc.,
Boston, MA, USA, 2002.

[6] Lance Spitzner. Honeytokens: The other honeypot. Securityfocus
http://www.securityfocus.com/infocus/1713, July 2003.

[7] Lance Spitzner. Honeypots: Catching the insider threat. In ACSAC '03: Proceedings of the
19th Annual Computer Security Applications Conference, page 170, Washington, DC, USA,
2003. IEEE Computer Society.

[8] Christian Kreibich and Jon Crowcroft. Honeycomb: creating intrusion detection signatures
using honeypots. SIGCOMM Comput. Commun. Rev., 34(1):51–56, January 2004.

[9] Jamie Riden and Laurent Oudot. Building a php honeypot. InfoSecWriters
http://www.infosecwriters.com, April 2006.

[10] Fabien Pouget and Marc Dacier. Honeypot-based forensics. In AusCERT Asia Pacific Information
technology Security Conference 2004, Brisbane, Australia, May 2004.