# Getting Started with the Internet Storm Center Webhoneypot

*GIAC (GWAPT) Gold Certification*

Author: Mason Pokladnik, mason@schwanda.cc
Advisor: Manuel Humberto Santander Pelaez

Accepted: December 2nd, 2013

Abstract

The goal of this paper is to help people get started using the ISC/Dshield Webhoneypot client on a Linux or Windows system, and give them a guide to getting the tool up and running  successfully so that they can then develop an understanding of how it works and hopefully increase the number of systems submitting data for study.

## 1. Introduction

The DShield/Internet Storm Center (ISC) Webhoneypot is a new project from DShield—a distributed intrusion detection system—that extends its logging capabilities from layer 3 and 4 network traffic further up the OSI layers to help study application layer attacks. According to the ISC website, millions of network layer events are logged into the Dshield database every day (Center, N.D.-a) yet only a few thousand application layer events are being monitored by the database (Center, N.D.-b).

The bulk of this paper relates to getting users to the "black triangle" moment where they have successfully configured the prerequisites, installed the Webhoneypot client and are finally ready to start collecting data (Barnson, 2004). It will also discuss some background information about honeypots in general and some risks to consider before running one.

There is also a section on beginning to analyze the log files for useful security intelligence information once the Webhoneypot is functional.

At the end of the process, users will have an extensible tool to aid them in studying application level attacks against their networks. They can continue to customize the Webhoneypot client as they see fit creating new "vulnerable" applications that match their network.

### 1.1. What is a Honeypot/Webhoneypot?

A honeypot system is primarily designed to collect data about attacks against any layer of the computer network from the network devices to computing systems and the applications running on them. They are usually classified into two main types:

- Low Interaction Honeypots – these honeypots allow only a limited amount of interaction in response to the actions of an attacker. They may provide a predetermined response to an attack in an attempt to fool an attacker into thinking a real system is responding, but only for the purposes of collecting more information. Those responses are not generated by the actual network service or application an attacker is targeting. Due to the lower level of interactivity with an attacker, low-interaction honeypots pose a lower risk level to the organizations running them. This is not to say they are risk-free.

- High Interaction Honeypots – these honeypots allow an increased level of response to an attacker. They may intentionally provide a command shell in response to a remote exploit allowing the attacker to run commands interactively on the honeypot, as well as allowing them to download files revealing even more information on an attacker's tools and tactics. Often, one of the goals of an attacker will be to establish a way of getting back into the system in the future (Project, 2000). The danger lies in that once an attacker has established their foothold they may want to use your honeypot system to launch attacks on other systems creating a liability issue for those who run high interaction honeypots. These liability issues could arise from attacks launched on other networks or even from violations of the law regarding monitoring other's communications. You will want to become familiar with these issues before running any type of honeypot on your network (Radcliffe, 2007).

The ISC Webhoneypot is a low interaction honeypot specifically focused on attacks aimed at web based applications. Over the past decade, security specialist have watched as attacks migrated from attacks on network servers (SMTP, HTTP, DNS and many others) to client-side attacks against web browsers, email clients and then third-party applications (PDF readers, Flash, Java, etc.). Another recent shift has been attackers targeting web-based applications along with the database and other systems they are built on. These are the attacks the Webhoneypot client is designed to try to collect information about.

## 1.2. Why Would You Want to Run a Honeypot?

The two biggest reasons are to learn about what attacks your network is facing and to study attack techniques and tools. Most honeypots are systems that are never supposed to be accessed. These are not real business systems, they are decoys. Any traffic to a honeypot system is immediately suspect since there should not be any traffic to a honeypot system from normal users. While all traffic is suspicious, it is not all malicious. Certain types of normal traffic will be present as honeypots have to log data to other systems and be maintained, and a webhoneypot will want to be indexed by search engines so that attackers can search for vulnerable web applications instead of just scanning the entire IP address space. However, once you filter out this baseline of traffic, nearly everything else is interesting in some way.

Imagine a honeypot file server on the network with an obvious name such as R&D-server. Such a name is likely to attract attention from even curious internal users who will likely give up when not granted access to any files. However, if the honeypot were to log access attempts from multiple machines during the middle of the night—perhaps during normal work hours in Southeast Asia—that type of traffic profile should generate some interest from incident responders.

A webhoneypot can provide similar intelligence against application level attacks. Logs from the ISC Webhoneypot can be correlated against other legitimate application servers to see attackers scanning your address space in preparation for a later attack. The logs can also provide additional details that a normal server may not log. Using this information you may be able to identify the application under assault, and begin to analyze the attack to see if any of your legitimate servers are vulnerable.

This type of collection works best when the honeypot more closely resembles your legitimate servers, so the ISC Webhoneypot has been written as a PHP script that can run on several webservers and operating systems. It also has the ability for users to create custom templates that resemble the applications running on their network. As this is a low interaction honeypot, it is not going to fool an attacker manually interacting with the site using a web browser, but it will keep some automated tools busy for awhile.

## 1.3. Webhoneypot License

The ISC Webhoneypot PHP script is published under the GNU Public License (GPL) v2.0 license. If you are not familiar with the GPL, the highlights are that you are free to use and modify the code without restriction. However, if you distribute or sell a piece of software using the GPL license, you must include the source code to that software. You can find out more details by reading the license itself at http://www.gnu.org/licenses/old-licenses/gpl-2.0.html (Foundation, 1991).

The templates distributed with the Webhoneypot are typically from open source licensed applications as well. If you are in a position to place a honeypot on the network, it is assumed you are already familiar with your organizations' policies on open source software.

## 1.4. How Does it Work?

The Webhoneypot is a PHP script that accepts any web request to the website that it is configured to run on. Figure 1 shows how the work of processing the request is divided between the web server and the PHP script itself.

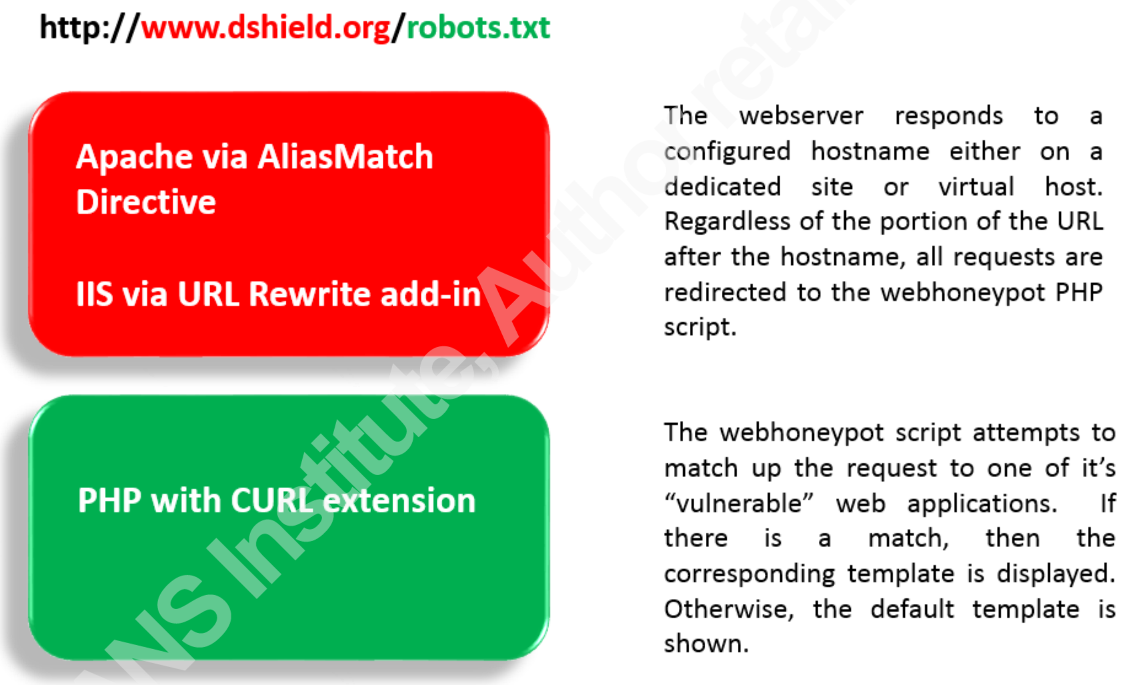

http://www.dshield.org/robots.txt
Apache via AliasMatch Directive
IIS via URL Rewrite add-in
PHP with CURL extension
The webserver responds to a configured hostname either on a dedicated site or virtual host. Regardless of the portion of the URL after the hostname, all requests are redirected to the webhoneypot PHP script.
The webhoneypot script attempts to match up the request to one of it's "vulnerable" web applications. If there is a match, then the corresponding template is displayed. Otherwise, the default template is shown.

**Figure 1**

The script uses a configuration file under templates/config.txt when attempting to match the request with a specific template. Using the example in Figure 1, the request for robots.txt would be matched via regular expression to the following line:

```
/(robots.txt$)/                                    104
```

The script would then look for a template in a subdirectory named 104 under the templates directory. Looking in that folder you will find a file called robots.txt that is returned to the client with the "interesting" directive that search engines please not index the directory /phpmyadmin_secret for the Webhoneypot webserver. This is, of course, an invitation for an attacker to please attempt to throw an exploit for PhpMyAdmin at the server if they (or their attack script) happens to have one available. There is no real instance of PhpMyAdmin on the server, but should they request it, the Webhoneypot will go through the process again of attempting to match the request to an appropriate template. If there is not a match, it will reply with the default template 1, a fake 404 error.

The templates provided cover a few known vulnerable applications, but the project will gladly take new ones supplied by users.

Finally, when the script has replied to the request with one of the available template sites, it then posts the original request and any HTTP headers to the Dshield database using PHP's cURL extension, and logs the request in the local log file as well.

## 2. Getting Started

### 2.1. Prerequisites

Before you can start running the Webhoneypot script, you will need to prepare an appropriate system to run it on. There are 2 required and 1 recommended prerequisites:

1. Mandatory – You will need an operating system running a webserver and PHP with the cURL extension installed. The most common operating system used is Linux running the Apache webserver, but Windows running IIS or Apache works as well. Don't let the common operating system suggestions limit your imagination. The Webhoneypot client could conceivably run on your home gateway device via DD-WRT or on nearly any other platform that supports PHP.
2. Mandatory - You will need a public IP address with port 80 available. A DNS hostname is not required, unless your address is dynamically assigned and will be changing. Many addresses for high-speed connections at home are dynamically assigned. If this is the case, you will need to register with and use a dynamic DNS service so that requests can be seeded for your host.

3. Optional – We highly recommend that anyone running the Webhoneypot client create an account on dshield.org. By default, the Webhoneypot will post data using a generic account, but by creating and using your own account you can review the data you are submitting via the ISC website without having to parse your own local logs. Plus, the ISC/Dshield website contains links to Webhoneypot clients that are registered under your ISC profile. This helps Google and other search engines to "find" your Webhoneypot and attackers as well.

### 2.1.1. Preparing a Linux or other UNIX-like Host to be a Webhoneypot Sensor

Installing Apache and PHP on Linux is a fairly well documented procedure. On many distributions you can just run a package utility like apt-get or rpm to download and install them. Please do take a few minutes to verify that your PHP installation is running with a production configuration by reviewing your php.ini file and comparing it with the file php.ini-production that should have been included with the distribution. Since the Webhoneypot is not actually running any of the vulnerable applications it is pretending to be, it can run under an extremely locked down configuration.

Verify your Apache install is working on your site or virtual host, and then move on to section 2.1.3 regarding verifying that PHP and the cURL extension are working.

### 2.1.2. Preparing a Windows Host to be a Webhoneypot Sensor

Traditionally, Windows has not been the first choice for running PHP, but it is a commonly held belief that there are far more Windows-only networks out there than there are those that run exclusively UNIX-like operating systems or a hybrid of the two. We recommend that you run your Webhoneypot sensor on whatever you are most comfortable administering so that you can keep it up to date with patches and not have your early warning sensor turn into a platform for others to attack into your—or other's—networks. As the process for getting Microsoft's IIS to run PHP is not as well documented we will go into more detail on this setup.

As the Webhoneypot requires that all requests be redirected to the Webhoneypot's index.php page to be processed, you will need a version of IIS that can perform this redirection. The method documented here uses the URL rewrite module v2.0 available from Microsoft (Yakushev, 2008). It is compatible with all versions of IIS since version 7—which shipped with Windows Server 2008 ®. Once you have downloaded and installed the rewrite module, verify that it shows up as a new option called URL Rewrite under IIS Manager.

You will also need to install PHP and configure IIS to use it. Download a recent version of PHP from [www.php.net](http://www.php.net). The 5.5 branch of the code is stable as of the time of this document, and you will want to download a 32-bit non-thread safe version of the software regardless of whether you are using a 32 or 64-bit version of Windows. This recommendation could change in the future. PHP for Windows is compiled against the Visual C++ runtime libraries. If you are using PHP 5.5, then you will need the appropriate version of the runtime as well. In this case, the x86 version (to match the x86 PHP binary) of the Visual C++ 11 runtime library (Microsoft, N.D.). Once installed, copy php.ini-production to php.ini.

PHP for Windows requires some changes to the php.ini file you copied earlier to prevent creating a security issue. Specifically, you will need to search for and change the following 4 values:

```
fastcgi.impersonate = 1

fastcgi.logging = 0

cgi.fix_pathinfo = 1

cgi.force_redirect = 0
```

Once PHP is installed, you will need to configure IIS to use PHP via the FastCGI module. Make sure the CGI module is actually installed under the application development section of the web server role. Detailed steps for how to do this are available at [http://www.php.net/manual/en/install.windows.iis7.php](http://www.php.net/manual/en/install.windows.iis7.php) under the header "Enabling FastCGI support in IIS." Keep that page open. It has several other useful sections. Then, you can either manually add the .php script handler following the instructions under the section "Using IIS Manager user interface to create a handler mapping for PHP" or you can download an open source tool called PHP Manager for IIS v1.2 which can make registering and switching between versions of PHP considerably easier (RuslanY, 2011). In this case, you will need the version of PHP Manager that matches the Windows/IIS bit level. For most of you, that will be 64-bit. You can confirm that by the presence of the directory c:\program files(x86)\ on the system. Once you have installed PHP Manager, close and reopen IIS Manager to find the new icon shown to the right. Click on the new PHP Manager icon and choose register new version of PHP. Step-by-step instructions for using PHP Manager to register the PHP script handler can be found at http://www.iishacks.com/2011/02/14/how-to-install-php-5-3-fastcgi-on-windows-2008-iis-7/. However, this link includes links to an older version of the PHP distribution and PHP Manager. Make sure you are getting the latest of each when you install.

PHP Manager

Since most of you will be running 64-bit windows with a 32-bit PHP binary, you will need to make sure the application pool that your site is running under supports running 32-bit applications. This step is described at http://www.iishacks.com/2011/02/14/how-to-install-php-5-3-fastcgi-on-windows-2008-iis-7/ as well. If this system is dedicated to the Webhoneypot, you can edit the DefaultAppPool's advanced settings and set the value "Enable 32-bit Applications" to True. If you are going to run the Webhoneypot on a host with multiple sites—dedicating one IP address to the Webhoneypot site—then you may want to consider creating a new application pool for the Webhoneypot site, and configuring it to enable 32-bit applications.

Finally, check the authentication settings for your site and verify that only anonymous authentication is enabled. You should edit the anonymous authentication settings and verify what user account is being used for the anonymous authentication. You will need this account name later when configuring permissions for log files.

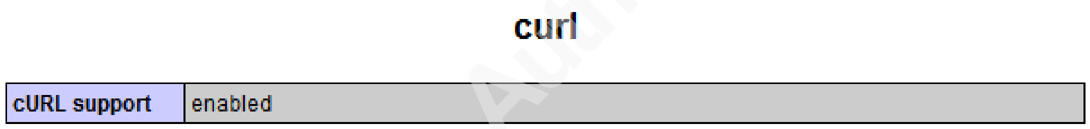
If all went well, you are ready to proceed to the next section to test your installation prerequisites.

### 2.1.3. Last Check – Use phpinfo to Verify Your Webserver, PHP, and CURL Status

On your webserver, create a file called index.php[1] in the document root of your site or virtual host. Place the following 3 lines in the file:

```
<?php
phpinfo();
?>
```

Save the file and then browse to the page on your webserver (e.g. http://FullyQuallifiedDomainName/index.php or http://ipaddress/index.php). If PHP is installed properly, you should see an information page listing the configuration status of your PHP install and installed modules. Do a search for "curl" on the page to verify the status of the module.



curl

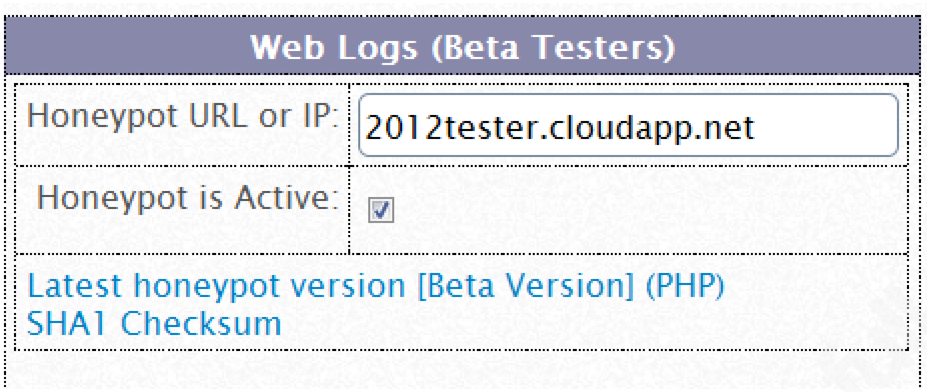
| | |
|---|---|
| cURL support | enabled |

If you see cURL support enabled, then you are ready to proceed to installing the Webhoneypot.

## 2.2. Downloading and Installing

The latest version of the ISC Webhoneypot can be downloaded from your profile page at https://isc.sans.edu. Login via the MyISC menu item, and then go to the My Information link. There you will find a field similar to Figure 2 below.

[1] We will be overwriting index.php later with the Webhoneypot client. Normally you would not want to leave this type of diagnostic information available for attackers to see.


Web Logs (Beta Testers)

| | |
|---|---|
| Honeypot URL or IP: | 2012tester.cloudapp.net |
| Honeypot is Active: | ✓ |

Latest honeypot version [Beta Version] (PHP)
SHA1 Checksum

**Figure 2**

Use this field to download the latest version of the Webhoneypot and enter either the DNS name or IP address that your Webhoneypot client is using. Once your system is operational, you will come back and check the box next to "Honeypot is Active" to mark it as an active sensor.

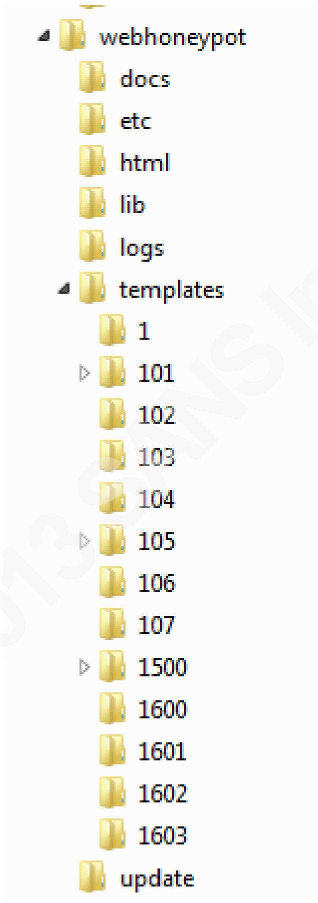


**Figure 3**

Extract the files to a temporary directory—not in your document root—and you should find a directory structure like the one in Figure 3. As this is a gzip'd tar file, you can use the command `tar –xzf webhoneypot.tgz` on a UNIX-like system or a tool like 7-zip on a Windows host to extract the files.

The first level of folders includes:

- DOCS – the documentation
- ETC – location of the local configuration file config.local. You will save any site specific configuration settings in this file.
- HTML – holds index.php which you will need to copy to your document root
- LIB – includes functions required by the Webhoneypot script
- LOGS – will hold the log files

Mason Pokladnik, mason@schwanda.cc

- TEMPLATES – holds all of the "applications" that the Webhoneypot will pretend to be. They are static HTML files the Webhoneypot script uses to respond to requests.
- UPDATE – holds scripts (update-templates.php and update-client.php) that can be scheduled to run to automatically check for a new version of the Webhoneypot script or templates and install them.

The location of these directories is expected to be at the same directory level as your document root for the website or virtual host. This is because the script calls relative paths to locations it needs to access (e.g. ../lib/common_functions.php and ../logs). Therefore, you will need to copy the first level directories—except for html—to the same level as your document root folder. By default, that would be html for apache or wwwroot for IIS. After that, copy just index.php into your document root folder overwriting the temporary file you created earlier which used the phpinfo command to verify that PHP was installed properly.

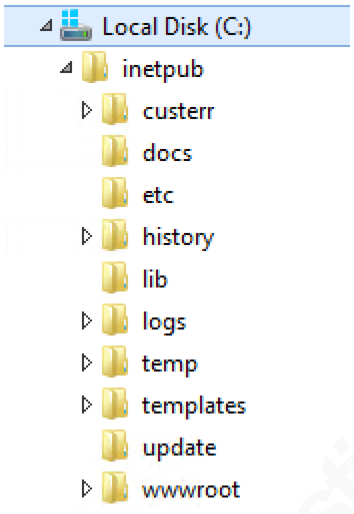


**Figure 4**

On a dedicated Windows Server 2012 R2 Webhoneypot client, the file structure looks like Figure 4 with the directories copied to the same level as wwwroot, and the index.php file copied into the wwwroot folder.

### 2.2.1. Setting Permissions for the Logs Folder

Typically, the account that your webserver process is running under will already have read access to the webserver directories. However, in order for the Webhoneypot to create a local log file, it will need access to write to the first level directory named logs. On a UNIX-like system, use the `chown` and `chmod` commands to give ownership of the logs directory and write permissions to the user account that the webserver is running under. Usually, that will be an account named apache. On a Windows system, you will need to find the user account that is being used to allow anonymous access to the website. Check this account under authentication settings, edit anonymous authentication.

Usually, the name of the account will start with IUSR. Once you have located the account, grant it write permissions to the logs folder.

### 2.2.2. Edit your Webserver Configuration to Redirect All Site Requests to the Webhoneypot

For Apache, you will need to edit the httpd.conf file for your site or virtual host. On Fedora, this is located at /etc/httpd/httpd.conf. On other distributions, it could be located elsewhere. Edit the configuration file and use the AliasMatch directive to have Apache redirect all requests to the absolute path for the Webhoneypot index.php file. On Fedora, the path to the index.php file under the default website is /var/www/html/index.php, so the AliasMatch command looks like:

```
AliasMatch ^/(.*) /var/www/html/index.php
```

A couple of notes regarding the placement of the AliasMatch directive in the file. First, make sure there are no other AliasMatch commands in the configuration file. You will either need to comment them out, or make sure that this is the first AliasMatch line in the configuration. As it matches any request, any other AliasMatch directives would be ignored. Second, if you are using a virtual host, make sure you make the configuration change in the appropriate section of the configuration. Under the docs folder in the Webhoneypot distribution, you will find an example virtual host configuration file used by honeypot.dshield.org.

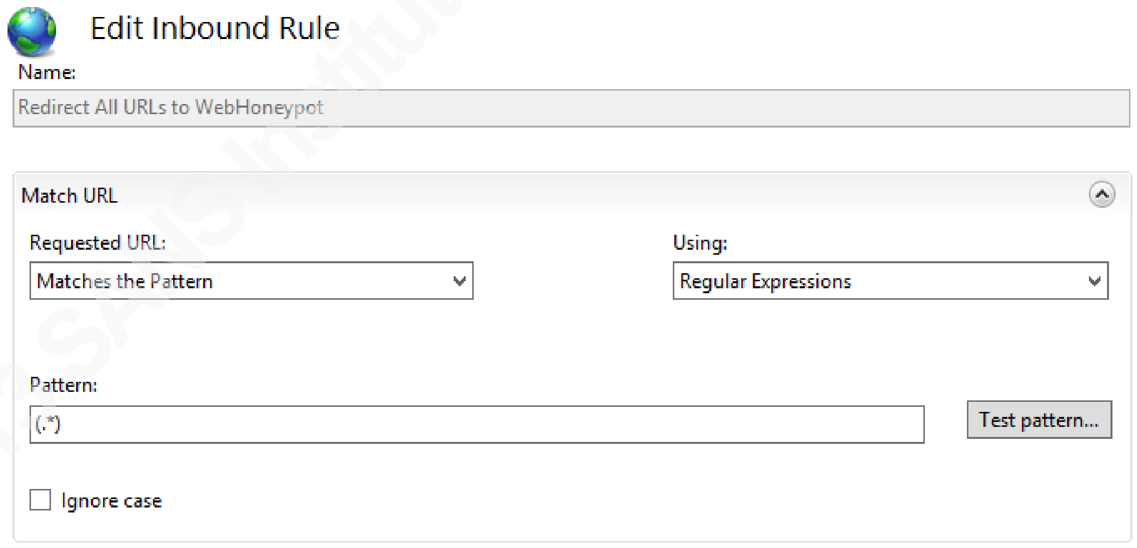
For Windows, you will need to configure the URL rewrite module. You can do this by either using the GUI in IIS manager or by manually placing a web.config file under your document root. For the latter, open a text editor, and copy in the following lines:

```
<configuration>
  <system.webServer>
    <rewrite>
      <rules>
        <!-- Rewrite URLs to redirect to 'index.php'. -->
        <rule name="Redirect All URLs to WebHoneypot"
enabled="true" stopProcessing="true">
```

```
          <match url="(.*)" ignoreCase="false" />
          <action type="Rewrite" url="/index.php" />
            <!-- <action type="Rewrite" url="/index.php"
appendQueryString="true" /> -->
        </rule>
      </rules>
    </rewrite>
  </system.webServer>
</configuration>
```

Save the file as web.config inside your document root—being careful not to save it as web.config.txt which will be ignored by IIS. Turning off the option to "hide file extensions for known file types" in Windows Explorer can make it easier to catch this mistake.

If you want to create the rule via the GUI, create a new inbound rule with the following settings:



Edit Inbound Rule

Name: Redirect All URLs to WebHoneypot

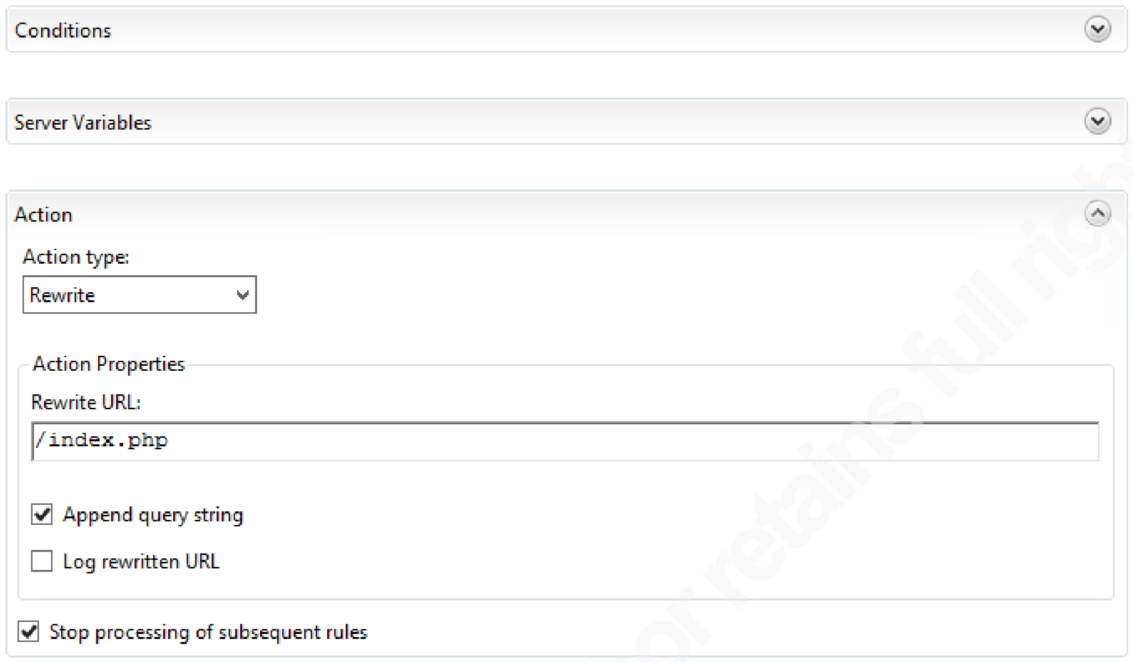
Match URL

Requested URL: Matches the Pattern

Using: Regular Expressions

Pattern: (.*)

☐ Ignore case


Conditions

Server Variables

Action

Action type:

Rewrite

Action Properties

Rewrite URL:

```
/index.php
```

☑ Append query string

☐ Log rewritten URL

☑ Stop processing of subsequent rules

That is a regular expression matching pattern of (.*) with an Action type of Rewrite and a Rewrite URL aimed at our Webhoneypot index.php file.

### 2.2.3. Configuring the Username to Submit Data to Dshield

The Webhoneypot comes preconfigured to submit data back to Dshield using a dummy account (demo@dshield.org). While this account will work to submit data back to the system, it means your submissions cannot be uniquely identified in the backend database. In which case, you cannot login to the ISC website and view the data you are submitting, but only see aggregated data.

To submit data under your own account, you will need to edit the file config.local under the etc directory of the Webhoneypot files you copied to your server earlier. In config.local you will see three default entries:

1. userid – this should be changed to the email address associated with your ISC/Dshield account
2. hashedpassword – a SHA1 hash of your login password
3. loglevel – the debug level for logging to the local logfile

As many of you do not have the SHA1 hash of your login password, the script has a special function to generate it for you. Once your Webhoneypot is operating, you can change the line in config.local from hashedpassword=hashvalue to password=cleartext password. Note the variable change from hashedpassword to password! If the variable is password, the script will compute the SHA1 hash of your password and save it in the local log file. You would then need to find the SHA1 hash in the local logfile and alter config.local once again with hashedpassword=SHA1 hash from log file. Do not put your plain text password in after hashedpassword.

### 2.2.4. Start Testing

You have completed configuring your Webhoneypot! Try to access a few URLs like http://yourhoneypot.dnsname/robots.txt or others that you create based on the pattern matching rules you find in templates\config.txt. Once you are satisfied that things are working properly, go back under your profile on the ISC website and check the box (Figure 2) indicating that your Honeypot client is active.

## 3. Putting it all Together

In this section, we will examine a full series of logs from a single request to the Webhoneypot client, and use them to follow the execution flow of the script. The example used is the first of several requests generated by the ZmEu scanner (Blizarazu, 2011). This scanner has been seen in the logs multiple times each month sourced from IP addresses from all around the world. Several of these IPs belong to one hosting provider in France. As ZmEu is looking for unpatched versions of PHPMyAdmin, we can speculate that there are vulnerable installations within their assigned IP address range.

The process starts when an HTTP request is received by the webserver—in this case IIS. The IIS rewrite module takes the request for http://2012tester.cloudapp.net/w00tw00t.at.blackhats.romanian.anti-sec:) and redirects it as a parameter to http://2012tester.cloudapp.net/index.php—our Webhoneypot client script.

As the Webhoneypot script starts, it verifies that it can write to the logs directory, and begins to log errors or progress.  It then initializes itself by loading configuration information from config.txt and config.local.  The information is loaded into an array called $aConfig in memory with default settings inside the script being overwritten by the configuration files as they are read in.

The example log we are looking at is from a successful run of the script, and should show what any Webhoneypot client will log—when operating correctly—at the default log level.  Every log line starts with the same four fields: Date, Time, Time zone and the IP address that made the request to the Webhoneypot client.  The first three log lines show debugging information from settings in config.local.  The log level is set to the highest level (9), and the username and password hash are logged.

```
2013-10-17 20:51:50 Central Standard Time 91.121.158.56
DEBUG: config parameter loglevel 9

2013-10-17 20:51:50 Central Standard Time 91.121.158.56
Configuration: user : notdemo@dshield.org

2013-10-17 20:51:50 Central Standard Time 91.121.158.56
Configuration: hashedpassword :

yoursha1hashedpasswordisloggedhere987654
```

The next two lines log URLs used for posting data back to Dshield and where to retrieve updates from config.txt.

```
2013-10-17 20:51:50 Central Standard Time 91.121.158.56
Configuration: posturl :

http://isc1.sans.org/weblogs/post.html

2013-10-17 20:51:50 Central Standard Time 91.121.158.56
Configuration: updateurl :

http://isc.sans.org/webhoneypot/update.html
```

The next three lines log the original request, and then a post-processing version of the request that attempts to remove the host and script name from the URL if they are present from a non-standard way of redirecting requests to the script.

```
2013-10-17 20:51:50 Central Standard Time 91.121.158.56
Debug: sUrl - before processing:
/w00tw00t.at.blackhats.romanian.anti-sec:)
2013-10-17 20:51:50 Central Standard Time 91.121.158.56
Debug: sUrl - after processing:
/w00tw00t.at.blackhats.romanian.anti-sec:)
2013-10-17 20:51:50 Central Standard Time 91.121.158.56
Debug: Request URI
/w00tw00t.at.blackhats.romanian.anti-sec:)
```

At this point the variable $sURL is now used as an input to a function that matches it against the regular expressions in config.txt. If there is a match, the matching template will be returned to the client. If not, the default template #1 will be returned. In this case, none of the regular expressions are matched, so the script chooses the default.

```
2013-10-17 20:51:50 Central Standard Time 91.121.158.56
Debug: Using template # 1
```

The script builds appropriate HTML headers and prepends them to the template file(s) which are then returned to the original requester.

```
2013-10-17 20:51:50 Central Standard Time 91.121.158.56
Debug: Looking for
C:/inetpub/templates/1/index.html
2013-10-17 20:51:50 Central Standard Time 91.121.158.56
Debug: using file
C:/inetpub/templates/1/index.html
```

```
2013-10-17 20:51:50 Central Standard Time 91.121.158.56
Info: no content type found for html

2013-10-17 20:51:51 Central Standard Time 91.121.158.56
INFO: sending body

2013-10-17 20:51:51 Central Standard Time 91.121.158.56
Delivered template 1
```

Finally, the script collects the original request, HTTP headers, IP address of the requester—and body of the request if there is one—to post back to the Dshield database.

```
2013-10-17 20:51:51 Central Standard Time 91.121.158.56
Collected request: 91.121.158.56

 GET /w00tw00t.at.blackhats.romanian.anti-sec:) HTTP/1.1

 X_ORIGINAL_URL: /w00tw00t.at.blackhats.romanian.anti-sec:)

 USER_AGENT: ZmEu

 HOST: 10.10.36.21

 ACCEPT_LANGUAGE: en-us

 ACCEPT_ENCODING: gzip, deflate

 ACCEPT: */*

 CONTENT_LENGTH: 0

 CONNECTION: Close

2013-10-17 20:51:51 Central Standard Time 91.121.158.56
Debug: Done collecting request.

Size=268 Bytes
```

The information is submitted to Dshield and either returns success (1) or failure.

```
2013-10-17 20:51:51 Central Standard Time 91.121.158.56
Debug: Post result = 1
```

## 4. Examining Your Logs for Security Intelligence

As the IPv4 address space does not take nearly as long to scan as it used to[2], it won't take long before your Webhoneypot client starts to attract some attention. Before I could even finish setting up my Windows based system it was already receiving requests for a file called azenv.php which turns out to be proxy judge script. It returns back the headers sent by the web browser so that you can look for additional headers, such as X-Forwarded-For, which would allow the user to evaluate both if they were being proxied, and, if so, how anonymous the connection might be.

You will find the local logs broken out by month in the logs folder located at the same level as your document root folder for the Webhoneypot. These log files include all sorts of useful information on how the client is functioning, useful debug information about the HTTP requests the Webhoneypot is receiving, whether they were posted successfully back to the database and error information when something goes wrong.

Using `grep`—or the powershell command `select-string`—you can search for lines in the log files with the text string "sUrl". These lines contain the raw and processed requests the Webhoneypot client has received and allow you to start researching potential attacks. For example, one of lines in my logs was a request for /invoker/JMXInvokerServlet. Just from the name it appears to be Java related. By correlating on the requesting IP address, the logs showed earlier requests for / (the default document) along with /web-console/Invoker. Using the inurl: directive to search Google for these directory names turned up hits related to the JBoss application server. A plugin for the w3af web scanner also shows the same JBoss link (Riancho, 2012).

While it is impossible to determine an attackers exact intentions and motivations, we now have an example of what one is searching for. A likely reason to be searching for unauthenticated access to JBoss directories would be to perform some sort of injection attack against the application server. This is just the sort of thing we are hoping to learn about with the project. One potential attack on an open application server directory like this could be to load the attackers own Java code and have the application

[2] See http://scans.io for an example of IPv4 wide scanning on a regular basis.

server run it as described at http://breenmachine.blogspot.com/2013/09/jboss-jmxinvokerservlet-exploit.html

The Web Application Hacker's Handbook describes this as attacking an application's architecture. They are exploiting a configuration issue—in this case unauthenticated access to the directory /invoker/JMXServletInvoker—to abuse the trust relationship between the different tiers of an application (Stuttard & Pinto, 2008).

The next step is to map this information on to your network. Do you run any JBoss application servers? Are you sure? If so, then you would want to include a scan to try and find vulnerable directories like this using w3af or another tool found during your research and regular auditing. Then, pick another scan out of the logs and repeat the process as you have time.

## 5. Conclusion

Given that tools like Havij make simple SQL injection attacks possible with little to no knowledge on how to manually carry out the attack (as seen in http://www.youtube.com/watch?v=Fp47G4MQFvA), it is imperative that network defenders understand application level attacks and have methods to monitor them. At the end of the process described in the paper, you should have a functional Webhoneypot client that provides you with insights into application attacks directed at your network and helps feed data into a the DShield distributed sensor network to help identify broader attack trends. After correlating with log files from other systems and some research, you should have improved your ability to detect and prevent attacks as you now know these systems are being actively scanned.

As with any type of honeypot, you will want to monitor it on a regular basis and patch the underlying PHP/webserver/OS stack as needed to prevent it from becoming a resource for attackers instead. We encourage you to download the Webhoneypot, try it out and see if you find it useful. If you come up with something new, and want to contribute it back to the project, contact details are available in the distribution under Docs/Install.

If you find the Webhoneypot client to be a useful tool, please also look at the 404 project https://isc.sans.edu/404project/. The 404 project is a script that you can load on to a customized 404 error "page not found" HTML page that will allow you to submit data from requests to your production websites. A word of warning: This tool has a much higher potential for exposing sensitive information, and you should consider carefully the risks of the data being submitted before utilizing.

## 6. References

Barnson, J. (2004). The Black Triangle. Retrieved Sept 21, 2013, from http://rampantgames.com/blog/2004/10/black-triangle.html

Blizarazu. (2011). ZmEu attacks: Some basic forensic. Retrieved November, 09 2013, from http://ensourced.wordpress.com/2011/02/25/zmeu-attacks-some-basic-forensic/

Center, I. S. (N.D.-a). About the Internet Storm Center. Retrieved September 9, 2013, from https://isc.sans.edu/about.html

Center, I. S. (N.D.-b). Webhoneypot: Report Volume. Retrieved September 9, 2013, from https://isc.sans.edu/webhoneypot/volume.html

Foundation, F. S. (1991). GNU General Public License, version 2. Retrieved Sept. 18, 2013, from http://www.gnu.org/licenses/old-licenses/gpl-2.0.html

Microsoft. (N.D.). Visual C++ Redistributable for Visual Studio 2012 Update 3. Retrieved October 11, 2013, from http://www.microsoft.com/en-us/download/details.aspx?id=30679

Project, H. (2000). Know Your Enemy Pt. III. *Know Your Enemey.* Retrieved October 30, 2013, from http://old.honeynet.org/papers/enemy3/

Radcliffe, J. (2007). Cyberlaw 101: A primer on US laws related to honeypot deployments. Retrieved October 23, 2013, from http://www.sans.org/reading-room/whitepapers/legal/cyberlaw-101-primer-laws-related-honeypot-deployments-1746

Riancho, A. (2012). w3af / plugins / infrastructure / find_jboss.py. Retrieved October 27th, 2013, from https://github.com/andresriancho/w3af/blob/master/plugins/infrastructure/find_jboss.py

RuslanY. (2011). PHP Manager 1.2 for IIS 7. Retrieved October 11, 2013, from http://phpmanager.codeplex.com/releases/view/69115

Stuttard, D., & Pinto, M. (2008). *The Web Application Hacker's Handbook*: Wiley Publishing, Inc.

Yakushev, R. (2008). Using the URL Rewrite Module. Retrieved October 11, 2013, from http://www.iis.net/learn/extensions/url-rewrite-module/using-the-url-rewrite-module