# Using honeypots and ELK to analyze cyber security trends

*Honeypots are very effective tools for monitoring global hacking attempts. Combined with the powerful analysis and visualization tools in the ELK stack (Elasticsearch, Logstash and Kibana) they form a great solution for analyzing cyber security trends.*



| | |
|---|---|
| Author | Gijs Rijnders |
| E-mail | gijs.rijnders@surfnet.nl |
| Date | 26-6-2015 |

# Abstract

I am currently a bachelor student Cyber Security at Fontys University of Applied Sciences in Eindhoven. As we speak I am doing my graduate internship at SURFnet in Utrecht. I am doing research about centralizing security logs from different services and providing insight in them using SIEM and big data analysis. As a side project, I am doing research about how honeypots can be used to gain intelligence about security treats and malware. You should definitely read this document if you are interested in how honeypots can help you gather information about possible threats for your infrastructure.

This document is about that side project. A possible context in which honeypots can be used is described. After the context, the setup which was used is described. The core of the document describes how the honeypot and analysis components can be configured on another system and how they interact. The appendixes contain a few scripts that are needed for the configuration of honeypot components.

# Index

# 1. Introduction

Cyber security is currently a hot topic. Computers are attacked on a daily basis. Many of the victims may not even be aware of that. More and more companies, governments and institutions are taking a lot of measures to ensure that their business and data are safe. However, it is impossible to be a 100 percent secure. Minimizing the risk is the best you can do. That brings us to possible preventive measures. You can monitor your own network and applications to be able to detect anomalies in an early stage, but that way you will only be able to detect threats that are already targeted at your organization.

Honeypots can be used to learn about threats before they are targeted at your organization. When a new threat is detected prematurely, a time window is created to research it and implement a countermeasure in vulnerable systems. Therefore honeypots can provide added value to the cyber security of an organization. This document describes the configuration of a low-interaction honeypot system with scalable data visualization capabilities.

# 2. Context

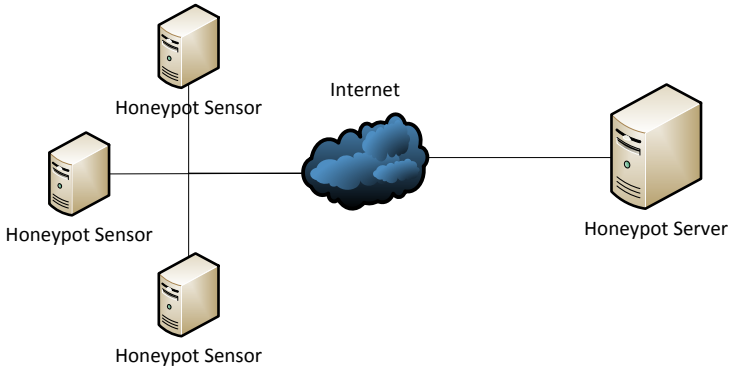
A few years ago SURFnet developed a honeypot solution using Kippo and Dionaea. This solution was called SURFids. It was based on a central log collection server that was keeps track of a set of sensors. Sensors may be placed anywhere on the internet. The sensors are honeypots that gather logs. These logs were collected on the central log collection server.

Sensors were placed in different networks. Providing insight in which networks draw more attacks and why was the goal of the system. The SURFids system is not running inside SURFnet anymore because it was no longer beneficial. Scalability and visualization are two features that were missing in the original setup.

As time goes by, new technologies and software enters the market. So did the ELK-stack. A powerful log collection and visualization tool that was highly scalable. There were new possibilities to create an effective honeypot setup. With slight modifications of the existing honeypots, it was possible to chain them to the ELK-stack. With previously created goals achieved, it was time to start thinking about new goals. For example, it should be possible create a fully automated process for capturing, classifying and analyzing malware. With the ELK-stack it is also possible to analyze trends from attacks that were executed against the honeypots.

Below is a graphical representation of the current setup. The amount of sensors is variable. The advantage of the new setup is that the sensors are loosely coupled. There are multiple ways to chain them to the central log collection server, each with their own pros and cons. Some methods may have better performance where others provide better security.

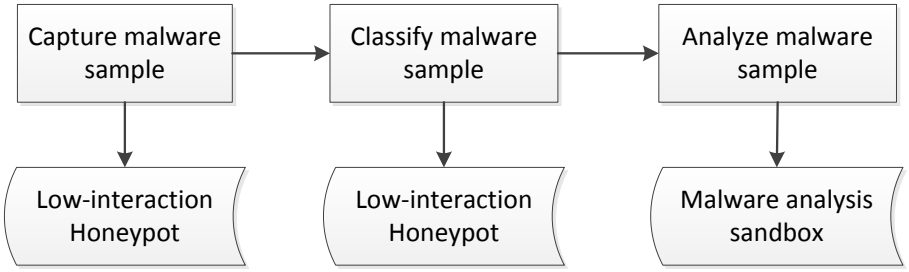
Figure 1: Graphical network map of honeypot setup



## 2.1. Malware analysis

The goal of the new honeypot setup is providing a highly scalable solution for analyzing trends at cyber security level. This involves automated scans, but also the analysis of malware samples. The process of gaining information about malware has three phases.

Figure 2: The process of analyzing malware samples




The goal of the three-phase process is providing the necessary information to be able create mitigation for a malware attack. Once malware is classified, it is known to other institutions as well. Signatures can be provided to update anti-virus software. Other than that, knowing what system objects are used and changed by a malware sample greatly increases the capability of network operators to take measures against the particular malware sample. This document does not cover the installation of sandboxing software.

# 3. Setup

The used honeypot setup is running on a virtual 64-bit Debian Linux system. It has two processors and 4 GB of memory. It is recommended to provide enough storage to the virtual machine. Especially the /var/ and /opt/ directories are extensively being saturated with data when the system is used in production. At least 100 GB of disk space is recommended.

The main honeypot used is Dionaea. Dionaea is a low-interaction honeypot solution that was designed to capture malware samples and classify them. Dionaea is meant to be a Nepenthes successor, which is a similar, older honeypot solution. Dionaea should be installed using the manual, which is located on the Dionaea website [1]. Another honeypot used is Artillery. Artillery is a relatively new honeypot that can be used to detect port scans and connections to specific ports. It can also be used to monitor files and folders for changes. The beauty of Artillery is that it is very easy to configure and the type of logging is very compact. There are short installation instructions for Artillery on the Artillery website. It needs to be installed using those instructions [2]. The third honeypot is Kippo. Kippo is a SSH honeypot that logs every command that is used by an attacker and saves downloaded malware. It needs to be pulled from git [3]. Malware samples are being caught and classified by Dionaea. However, analyzing that malware is also a very important step. To analyze caught malware automatically, sandboxing software can be used.

Honeypots generate logs. These logs need to be saved and aggregated in order to be able to analyze them. The ELK stack is a solution for big data analysis on logs. ELK resolves to: Elasticsearch, Logstash and Kibana, and is developed by a Dutch company called Elasticsearch. It is an open source solution that makes visualization of honeypot logs an easy task. Elasticsearch is a real-time search engine that uses a lot of memory. 4 GB should be enough to have Elasticsearch run properly. However, the more memory that is made available to it, the better it will perform. Take into account that the Elasticsearch documentation states not to use more than 32 GB of memory.

To enable Dionaea to submit malware samples to the Virustotal web service for classification, you need an API key. To get an API key, create an account on the Virustotal website. Once you made an account, the API key can be retrieved by clicking 'My API Key'.

p0f is also part of the setup. p0f is a tool that attempts fingerprinting of TCP connections. Based on a local signature database it tries to identify the operating system of the system on the other side of the connection is running. The installation guide for p0f is included.

Table 1: Used software in the research setup

| Software | Version |
|---|---|
| Debian Wheezy | 3.16.0-0.bpo.4-amd64 (kernel) |
| Dionaea | 0.1.0 |
| Artillery | 1.0 |
| Kippo | Latest GIT (using Twisted daemon 13.2) |
| p0f | 2.0.8 |
| Elasticsearch | 1.6.0 |
| Logstash | 1.5.1 |
| Kibana | 3.1.2 |

[1] *The Dionaea website can be found in the bibliography.*
[2] *The Artillery website can be found in the bibliography.*
[3] *The Kippo website can be found in the bibliography.*

# 4. Configuration

This is the core chapter of this document. It describes how the honeypots should be configured and if necessary, customized. It requires the Dionaea and Artillery honeypots to be installed. It also requires Elasticsearch, Logstash and Kibana to be installed. Before getting started, it is recommended to make sure a few security measures are taken.

Our testing environment was not protected by a firewall. This is also not recommended in a honeypot setup. SSH is a service that is vulnerable to brute force attacks. Therefore we configured a simple brute force mitigation tool called Denyhosts. On Debian based Linux distributions this tool can be installed using the following command.

*apt-get install denyhosts*

By default, Elasticsearch's REST API is accessible at port 9200. There is no authentication required, so leaving it open is a serious security risk. In the used setup, everything is located on the same server. This allows us to bind Elasticsearch to localhost. This way the service will still be accessible for Logstash and Kibana, but it will not be available to the public anymore. Look for a line named: *network.host* in the Elasticsearch configuration (/etc/Elasticsearch/Elasticsearch.yml) and change it to the following. Make sure the line is uncommented.

*network.host: localhost*

It is also recommended to disable the possibility to execute scripts via the Elasticsearch REST API. Allowing scripts to be executed creates the risk of code to be executed on the server remotely. Disabling scripts can be done by appending the following line to the Elasticsearch configuration.

*script.disable_dynamic: true*

If you run an Elasticsearch cluster and don't want to disable public to the REST API access using the built-in Elasticsearch configuration option, iptables can be used to restrict access. For example, to restrict access to the subnet in which the Elasticsearch nodes are present, execute the following iptables commands. The IP address and subnet should be changed according to your situation.

*iptables –A INPUT –p tcp --dport 9200 –s 192.42.130.0/22 –j ACCEPT*
*iptables –A INPUT –p tcp --dport 9200 –j DROP*

The best solution is to run Elasticsearch clusters behind a firewall though. This can eliminate a lot of security issues using a centralized solution.

## 4.1. Dionaea

By default, Dionaea will be installed into '/opt/dionaea/'. The structure of the folder is as following.

Table 2: Filesystem structure of Dionaea installation

| Directory/File | Description |
|---|---|
| bin | Contains the main Dionaea binaries, including some specialized python binaries. |
| etc | Contains configuration files for Dionaea and its libraries. |
| include | Contains header files for Dionaea and its libraries. |
| lib | Contains binary files for libraries that Dionaea uses. Most of these are manually compiled during the installation of Dionaea. |
| sbin | Contains some secondary binaries for Dionaea. |
| share | Contains documentation for Dionaea and its libraries. |

| Directory/File | Description |
|---|---|
| var | Contains log files and information used by Dionaea during runtime. Examples: log files, captured malware, data streams and the fake webserver root. |

### 4.1.1. Configuration

Dionaea is hard to configure properly. It needs to be built from source and has a lot of features. Other than that, we are going to make a few modifications to Dionaea in order for it to fit into the ELK-stack setup.

Open up the configuration file in the installation directory when the installation is done. This file is located in /opt/dionaea/etc/dionaea/dionaea.conf by default. There are two sections that we need to modify.

- Logging;
- Modules.

Dionaea is extremely verbose when generating logs using its default settings. We don't need all levels of logging so we are going to change these settings. The Dionaea configuration is a C-style configuration file with blocks. Removing blocks from the configuration is an easy job. Place comment block indicators around the default logging entry and entries for 'logstash' and 'virustotal'. The configuration should look like the configuration displayed below.

```
logging = {
	/*default = {
		// file not starting with / is taken relative to LOCALESTATEDIR$
		file = "log/dionaea.log"
		levels = "all"
		domains = "*"
	}*/

	errors = {
		file = "log/dionaea-errors.log"
		levels = "warning,error"
		domains = "*"
	}

	logstash = {
		file = "log/surfids.log"
		levels = "all"
		domains = "surf*"
	}
	virustotal = {
		file = "log/virustotal.log"
		levels = "all"
		domains = "virustotal*"
	}
}
```

Dionaea is based on modules. Every feature that Dionaea uses is a module. The SURFids module configuration does not need to be changed. We need to edit the configurations of the Virustotal module and the ihandlers module. The Virustotal module will submit malware samples to the Virustotal web service. The classifications for the malware sample are returned to be logged to a database, or in our case, a log file. The ihandlers module provides logging functionality for every module that needs to. Therefore we need to enable the Virustotal and SURFids modules to log their data using an ihandler. Look for the 'virustotal' block in the modules configuration section. Add your Virustotal API key to the configuration [1]. The database file does not need to be changed. The configuration should look like this.

```
virustotal = {
                apikey = "<API key>"
                file = "var/dionaea/vtcache.sqlite"
}
```

Look for the 'ihandlers' block and change it so it looks like this. Note the removed comment before the Virustotal and SURFids modules.

```
ihandlers = {
    handlers = ["ftpdownload","tftpdownload","emuprofile","cmdshell","store","uniquedownload",
//  "logsql",
    "virustotal",
//  "mwserv",
//  "submit_http",
//  "logxmpp",
//  "nfq",
//  "p0f",
    "surfids",
//  "fail2ban"
     ]
}
```

Changing the Dionaea configuration file was the first step that is needed. The next step is applying a few modifications to the Dionaea scripts. The scripts are written in Python and located at /opt/dionaea/lib/dionaea/python/dionaea/ by default.

The SURFids and Virustotal modules have to be modified. The modifications throughout these files are bigger. Therefore the completed replacement files are appended to this document. The contents of the following files have to be changed. The correct appendixes are listed as well.

Table 3: Dionaea modification appendixes

| File | Appendix |
|---|---|
| surfids.py | SURFids module for Dionaea |
| virustotal.py | Virustotal module for Dionaea |

Once the configuration is finished, Dionaea can be started. We will create an init script to start Dionaea. This can be done with the following commands.

*cd /etc/init.d*
*touch dionaea*

Copy the contents of the appendix: 'Init script for Dionaea' and place it inside the newly created script. Start the script using the following command.

*service dionaea start*

[1] *The Virustotal API key is a prerequisite. Check the 'Setup' section for more information.*

### 4.1.2. Diagnosing problems

A lot of steps of the Dionaea installation or configuration can cause problems. In case something goes wrong when installing Dionaea, the output of the 'configure' and 'make' commands can be very useful to help you solve the problem.

A known build issue is brought by pulling libnl3 from its git repository. A few changes have been made in newer versions of libnl3. These changes make the Dionaea build process fail. Version

3.2.7-4 of libnl3 (-dev), installed using the Debian package manager, has been tested. This version of the library works as expected.

If something goes wrong during the configuration of Dionaea, the debug logging that Dionaea generates can be very useful to help you solve the problem. To run Dionaea in the foreground with debugging enabled, run the following command.

*/opt/dionaea/bin/dionaea -l all -L '*'*

A common issue is that one or more modules were not successfully located during the build process of Dionaea. This may originate from the include paths that are not correct when calling the configure script. If the debug output from Dionaea shows purple entries like the following picture, this is likely the issue. Note: the NFQ module is not required! This is just an example.

Figure 3: Example output of failed module when running Dionaea with debugging enabled

```
[16032015 13:24:10] dionaea dionaea.c:668: OpenSSL 1.0.1e 11 Feb 2013
[16032015 13:24:10] dionaea dionaea.c:681: udns version 0.0.9
[16032015 13:24:10] modules modules.c:109: modules_load node 0xb973a0
[16032015 13:24:10] modules modules.c:120: loading module curl (/opt/dionaea/lib/dionaea/curl.so)
[16032015 13:24:10] curl module.c:700: module.c:700 module_init dionaea 0xb8d4c0
[16032015 13:24:10] modules modules.c:120: loading module emu (/opt/dionaea/lib/dionaea/emu.so)
[16032015 13:24:10] emu module.c:82: module.c:82 module_init dionaea 0xb8d4c0
[16032015 13:24:10] modules modules.c:120: loading module pcap (/opt/dionaea/lib/dionaea/pcap.so)
[16032015 13:24:10] pcap pcap.c:397: pcap.c:397 module_init dionaea 0xb8d4c0
[16032015 13:24:10] modules modules.c:120: loading module nfq (/opt/dionaea/lib/dionaea/nfq.so)
[16032015 13:24:10] modules modules.c:56: could not load /opt/dionaea/lib/dionaea/nfq.so /opt/dionaea/
lib/dionaea/nfq.so: cannot open shared object file: No such file or directory
[16032015 13:24:10] modules modules.c:125: could not load module nfq (Success)
[16032015 13:24:10] modules modules.c:120: loading module python (/opt/dionaea/lib/dionaea/python.so)
[16032015 13:24:10] python module.c:1048: module.c:1048 module_init dionaea 0xb8d4c0
[16032015 13:24:10] modules modules.c:120: loading module nl (/opt/dionaea/lib/dionaea/nl.so)
[16032015 13:24:10] nl module.c:449: module.c:449 module_init dionaea 0xb8d4c0
```

When the Python modules of Dionaea are modified, it is possible that one of them contain an error. When an exception is thrown from Python, it is caught when running Dionaea with debugging enabled. The following picture gives an example of how a caught exception can be identified.

Figure 4: Example output of an exception thrown from a Python module

```
[17032015 13:33:18] incident incident.c:167:    uuid: (string) 4b324fc8-1670-01d
3-1278-5a47bf6ee188
[17032015 13:33:18] incident incident.c:167:    con: (ptr) 0x7f40a4075c90
[17032015 13:33:18] incident incident.c:167:    opnum: (int) 31
[17032015 13:33:18] python module.c:778: traceable_ihandler_cb incident 0x7f40a4
035280 ctx 0x2153368
[17032015 13:33:18] python module.c:1001: NameError at NameError("global name 'C
onnectionError' is not defined",)
[17032015 13:33:18] python module.c:1026: /opt/dionaea/lib/dionaea/python/dionae
a/surfids.py:46 in handle_incident
[17032015 13:33:18] python module.c:1027:        except ConnectionError as e:
[17032015 13:33:18] python module.c:1026: thon/binding.pyx:1181 in dionaea.core.
c_python_ihandler_cb (binding.c:12276)
[17032015 13:33:18] python module.c:1027:        None
[17032015 13:33:18] python module.c:778: traceable_ihandler_cb incident 0x7f40a4
035280 ctx 0x2153758
```

## 4.2. Artillery

Artillery is installed to '/var/artillery/' by default. Artillery should be set to start on system boot up. The structure of the folder is as following.

Table 4: Filesystem structure of Artillery installation

| Directory/File | Description |
|---|---|
| database | Contains data that is used by artillery during runtime. This information is not useful for the setup we have. |
| logs | Contains log files that Artillery generates during runtime. |
| readme | Contains text files with information about the software and its license. |
| src | Contains source files of Artillery. |
| artillery.py | The startup script for the Artillery honeypot. |
| banlist.txt | Saves information about banned systems. This information is pulled from a central thread intelligence server every 24 hours by default. |
| config | The Artillery configuration file. |
| remove_ban.py | A script that can be used to remove a ban from the ban list. |
| restart_server.py | A script that can be used to restart the Artillery service once started. |
| setup.py | A script that can be used to invoke the Artillery setup. Once installed, this script should not be necessary anymore. |

### 4.2.1. Configuration

The default Artillery configuration is very easy and straightforward. That is the biggest advantage of using this honeypot. A few things need to be changed.

The monitor is used to let Artillery detect changes in certain directories. Turn it off by changing the value of the monitor configuration line to the following.

*MONITOR="OFF"*

Artillery checks whether the local SSH service runs at port 22 by default. We don't want it to do that. Turn it off by changing the configuration line to the following.

*SSH_DEFAULT_PORT_CHECK="OFF"*

In our setup, we are mixing Dionaea with Artillery. Because services are opened on ports by both honeypots, we have to create a distinction between which honeypot uses which ports. Dionaea gets the highest priority for its ports because it has to capture malware. You are free to add any valid port to the Artillery configuration. It will spawn a service on that port and log connections to it. Just make sure that ports do not overlap other local services. To remove the overlap with Dionaea and local services, a few ports need to be removed from the default configuration. Below is the configuration line that was used inside the original testing environment. In that line, the overlaps are removed and some additional ports are added.

*PORTS="23,53,69,110,137,138,139,143,161,162,369,389,465,546,547,569,587,990,993,1099,1241,1337,1723,1813,1863,4031,4662,4899,5190,5432,5666,5667,5737,5800,5900,6346,6347,8080,8088,9999,10000,44443"*

Artillery also checks whether for brute force attempts on the local SSH service by default. This configuration may be kept if it is useful for your setup. For the original testing environment, the Denyhosts was used to provide this security measure. Alter the configuration line as following to turn off the SSH brute force check.

*SSH_BRUTE_MONITOR="OFF"*

'Anti-DOS' is a feature of Artillery that is enabled by default. It throttles connections that rise above the specified limit. We don't want this feature turned on. Change the anti-DOS configuration line as following.

*ANTI_DOS="OFF"*

Last but not least, we need to configure where Artillery saves its logs. We want Artillery to write its logs to the file 'alerts.log' inside the /var/artillery/logs/ directory. The logs are formatted as Syslog. Change the log configuration line according to the following.

*SYSLOG_TYPE="file"*

If you run an Elasticsearch cluster, logging to a remote Syslog server may be a better solution. Below are the lines that need to be changed in order to enable remote logging to Syslog.

*SYSLOG_TYPE="REMOTE"*
*SYSLOG_REMOTE_HOST="<remote_ip>"*
*SYSLOG_REMOTE_PORT="514"*

The configuration of Artillery is done. The service can be started with the following command.

*service artillery start*

### 4.2.2. Diagnosing problems

The configuration of Artillery is pretty straightforward and should not cause serious problems. However, some things may go wrong. If you configured Artillery to log to the alerts.log file, that is the place to start looking.

If a port that Artillery is configured to listen on is already occupied, the log file will contain an error. This error looks like the following line.

*[!] 2015-03-12 15:25:40: Artillery was unable to bind to port: 22. This could be to an active port in use.*

Artillery will ignore the failed port and continue running. However, it is better to resolve the issue. This can be done by finding out what service is running on that port or simply removing the port from the Artillery configuration.

## 4.3. Kippo

Kippo is a side honeypot. It runs next to Artillery and Dionaea as a fake SSH service. It doesn't necessarily need to be installed. It just needs to be downloaded. The best part is: we can use it with its default configuration. We just need to create an init script for it to start when the machine boots up. Create an init script using the following command (Kippo is located in /home/gijs/Kippo in our setup).

*cd /etc/init.d*
*touch kippo*

Add the following contents to the init script. Edit the path to the Kippo installation if necessary.

```
#! /bin/sh
# /etc/init.d/kippo

# The following part always gets executed.
echo "Kippo operations script..."

# The following part carries out specific functions depending on arguments.
case "$1" in
  start)
    echo "Starting Kippo SSH Honeypot."
    /home/gijs/kippo/start.sh
    echo "Kippo SSH Honeypot started."
    ;;
  stop)
    echo "Stopping Kippo SSH Honeypot."
    /home/gijs/kippo/stop.sh
    echo "Kippo SSH Honeypot stopped."
    ;;
  *)
    echo "Usage: /etc/init.d/kippo {start|stop}"
    exit 1
    ;;
esac

exit 0
```

## 4.4. p0f

Dionaea provides a module for p0f. Still, we are not using it. We don't use it because p0f cannot be forked into the background without providing it an output file. The logs that the Dionaea module would receive are the same as the logs that p0f generates when running in a standalone manner. This does not have any disadvantage for the data that is logged.

The installation and configuration of p0f is easy and straightforward. p0f will attempt to fingerprint connections that are made to the honeypot. It does this by matching known signatures to SYN packets that are received by the honeypot. For our setup we will be using p0f v2. Install it with the following command.

*apt-get install p0f*

After installing p0f we would like it to start when the honeypot server boots up. We need an init script to do so. Execute the following commands to create the script.

*cd /etc/init.d*
*touch p0f*
*chmod +x p0f*
*update-rc.d p0f defaults*

The script is created, but it does not contain anything yet. The init script is a larger piece of text. Copy it from the appendix: 'Init script for p0f' and place it in the newly created script. Once you've done this you should be able to start p0f using the following command.

*service p0f start*

## 4.5. Logstash

Once the honeypots are configured, we need to configure Logstash to use their logging as Elasticsearch input. We are going to do this using configuration files. There are three types of configuration files.

- Input;
- Filter;
- Output.

The input configuration files tell Logstash where it should look for logging. The filter configuration files tell Logstash how it should interpret and manipulate the logs that it finds. Finally, the output configuration files tell Logstash where it should push its processed logs to. The configuration files must be located in '/etc/logstash/conf.d/'. They can be numbered using the filename, where they will be loaded sequentially. The following table describes which configuration files we need to create. The contents of these files are located below the table.

**Note:** You also need to install the logstash-contrib package in order to have certain community plugins available, if the version of Logstash you installed is lower than 1.5.0. This package is available from the Elasticsearch website. Using plugins from the contributions package without having it installed will result in Logstash throwing errors in its log file.

Table 5: Logstash configuration files

| Configuration File | Description |
|---|---|
| 10-dionaea-input.conf | Defines the input parameters for logging generated by Dionaea. |
| 11-artillery-input.conf | Defines the input parameters for logging generated by Artillery. |
| 12-kippo-input.conf | Defines the input parameters for logging generated by Kippo. |
| 13-p0f-input.conf | Defines the input parameters for logging generated by p0f. |
| 20-dionaea-filter.conf | Tells Logstash how it should parse logs generated by Dionaea. |
| 21-artillery-filter.conf | Tells Logstash how it should parse logs generated by Artillery. |
| 22-kippo-filter.conf | Tells Logstash how it should parse logs generated by Kippo. |
| 23-p0f-filter.conf | Tells Logstash how it should parse logs generated by p0f. |
| 30-output.conf | Tells Logstash where it should push its parsed logs to. |

10-dionaea-input.conf:

```
input
{
        file
        {
                path => [ "/opt/dionaea/var/log/surfids.log", "/opt/dionaea/var/log/virustotal.log" ]
                type => "dionaea"
        }
}
```

11-artillery-input.conf:

```
input
{
	file
	{
		path => "/var/artillery/logs/alerts.log"
		type => "artillery"
	}
}
```

12-kippo-input.conf:

```
input
{
	file
	{
		path => "/home/gijs/kippo/log/kippo.log"
		type => "kippo"
	}
}
```

13-p0f-input.conf:

```
input
{
	file
	{
		path => [ "/var/log/p0f/p0f.log" ]
		type => "p0f"
	}
}
```

20-dionaea-filter.conf:

```
filter
{
	if [type] == "dionaea"
	{
		grok
		{
			match => { "message" => [ "%{MONTH-
DAY:day}%{MONTHNUM:month}%{YEAR:year}%{SPACE}%{TIME:timestamp}%{DATA}%{WORD:domain}%{SPACE}%{GR
EEDYDATA:file}:%{POSINT:line_number}-
%{WORD:level}%{DATA}%{IP:src_ip}:%{POSINT:src_port}%{DATA}%{IP:dst_ip}:%{POSINT:dst_port}%{DATA}%{WOR
D:protocol}%{SPACE}%{GREEDYDATA:message}",
                                        "%{MONTH-
DAY:day}%{MONTHNUM:month}%{YEAR:year}%{SPACE}%{TIME:timestamp}%{DATA}%{WORD:domain}%{SPACE}%{GR
EEDYDATA:file}:%{POSINT:line_number}-
%{WORD:level}:%{SPACE}Scanner%{SPACE}%{WORD:scanner}%{SPACE}reported%{SPACE}%{GREEDYDATA:classificatio
n} for%{SPACE}%{GREEDYDATA:filehash}" ] }
			add_field => [ "received_at", "%{@timestamp}" ]
		}
		geoip
		{
			source => "src_ip"
			target => "geoip_source"
			add_field => [ "[geoip][coordinates]", "%{[geoip][longitude]}" ]
			add_field => [ "[geoip][coordinates]", "%{[geoip][latitude]}"  ]
		}
		mutate
		{
			convert => [ "[geoip][coordinates]", "float" ]
		}
	}
}
```

21-artillery-filter.conf:

```
filter
{
```

```
	if [type] == "artillery"
	{
		grok
		{
			match => { "message" =>
"%{TIMESTAMP_ISO8601:timestamp}%{GREEDYDATA}%{IP:src_ip}%{GREEDYDATA}%{POSINT:dst_port}" }
		}
		geoip
		{
			source => "src_ip"
			target => "geoip_source"
			add_field => [ "[geoip][coordinates]", "%{[geoip][longitude]}" ]
			add_field => [ "[geoip][coordinates]", "%{[geoip][latitude]}"  ]
		}
		mutate
		{
			convert => [ "[geoip][coordinates]", "float" ]
		}
	}
}
```

22-kippo-filter.conf:

```
filter
{
	if [type] == "kippo"
	{
		grok
		{
			match => { "message" =>
"%{TIMESTAMP_ISO8601:timestamp}%{GREEDYDATA}%{IP:src_ip}\]%{SPACE}%{GREEDYDATA:kippo_message}" }
		}
		geoip
		{
			source => "src_ip"
			target => "geoip_source"
			add_field => [ "[geoip][coordinates]", "%{[geoip][longitude]}" ]
			add_field => [ "[geoip][coordinates]", "%{[geoip][latitude]}"  ]
		}
		mutate
		{
			convert => [ "[geoip][coordinates]", "float" ]
		}

	}
	if "_grokparsefailure" in [tags]
	{
		drop { }
	}
}
```

23-p0f-filter.conf:

```
filter
{
	if [type] == "p0f"
	{
		grok
		{
			match => { "message" => "%{SYS-
LOGTIMESTAMP:timestamp}%{SPACE}%{YEAR:year}%{DATA}%{IP:src_ip}:%{POSINT:src_port}%{SPACE}-
%{SPACE}%{GREEDYDATA:fingerprint}->%{SPACE}%{IP:dst_ip}:%{POSINT:dst_port}" }
		}
		if "_grokparsefailure" in [tags]
		{
			drop { }
		}
		if [src_ip] == "<enter ip of your honeypot server here>"
		{
			drop { }
		}
	}
}
```

30-output.conf:

```
output
{
      elasticsearch
      {
            host => "localhost"
      }
}
```

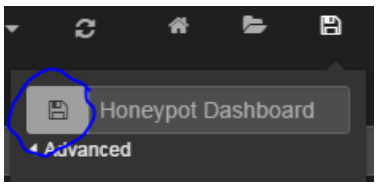
## 4.6. Kibana

We are using Kibana to visualize the data we get from the honeypots. A dashboard needs to be made in order to display the data. The following steps were taken to get the dashboard as displayed in the final figure of this chapter.

**Note:** Data from the honeypots is needed in order to create the dashboard. If there is no data collected yet, some of the required fields may not yet be created.

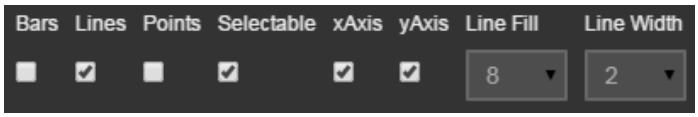
Open up the prebuilt 'Logstash Dashboard' and click the 'Save' icon on the top right corner of the page. Enter a name, for example: 'Honeypot Dashboard' and hit the newly appeared button to save it.

Figure 5: Saving a new Kibana dashboard



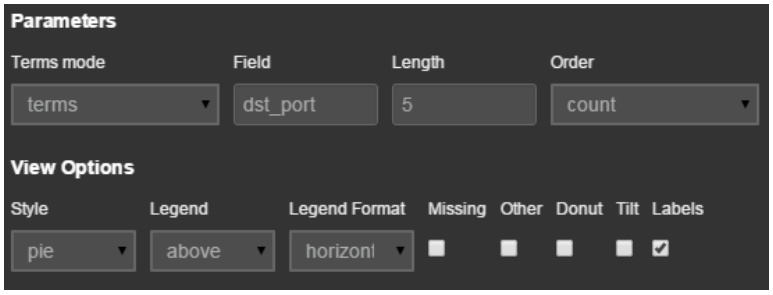

Click the cog-shaped icon in the top right corner of the 'Events over time' panel. Change the 'Span' property to 4. This reduces the width of the panel to a third of the screen width. Click the 'Style' tab on top. Uncheck the 'Bars' checkbox and check the 'Lines' checkbox. Set the line fill to 8 and the line width to two. Then click 'Save' to apply the changes to the panel.

Figure 6: Altering the events over time panel




Inside the top row of the dashboard, click the green button that says 'Add Panel'. Select 'terms' as panel type and enter 'Trending Ports' as title. Set the field to 'dst_port', the length to 5 and the style to 'Pie'. Uncheck 'Other' and 'Missing'. Click 'Save' to add the panel.

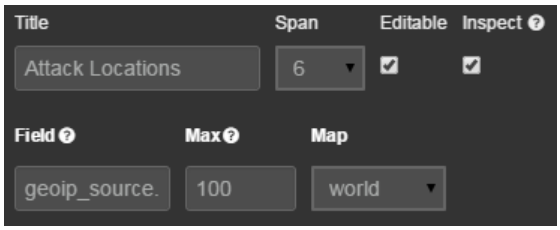
Figure 7: Adding a trends panel to the Kibana dashboard




Add another terms panel inside the same row. Use the same settings as in the previous panel except the field. Leave the field at its default value: '_type'.

Remove the table panel inside the second row by clicking the cross-shaped icon in the top right corner of the panel. Add a new panel to the second row. Select 'map' as type, name it 'Attack Locations' and select a span of 6. Select 'geoip_source.country_code2' as field and save the panel.

Figure 8: Adding a map panel to the Kibana dashboard




Add another terms panel inside the second row. Set the title to 'Captured Malware' and select a span of 6. Set the field to 'classification.raw'. Set the style to 'table' and uncheck the 'Missing' and 'Other' options. Finally, save the panel.

Figure 9: Adding the captured malware panel to the Kibana dashboard

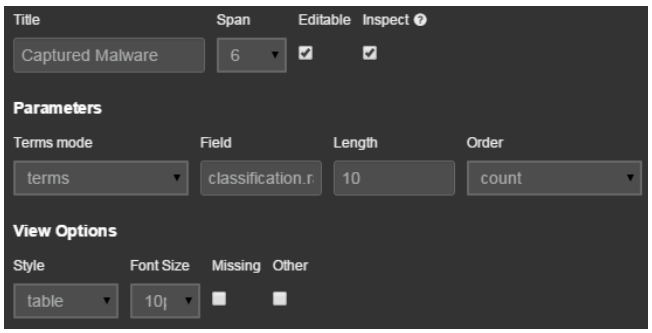
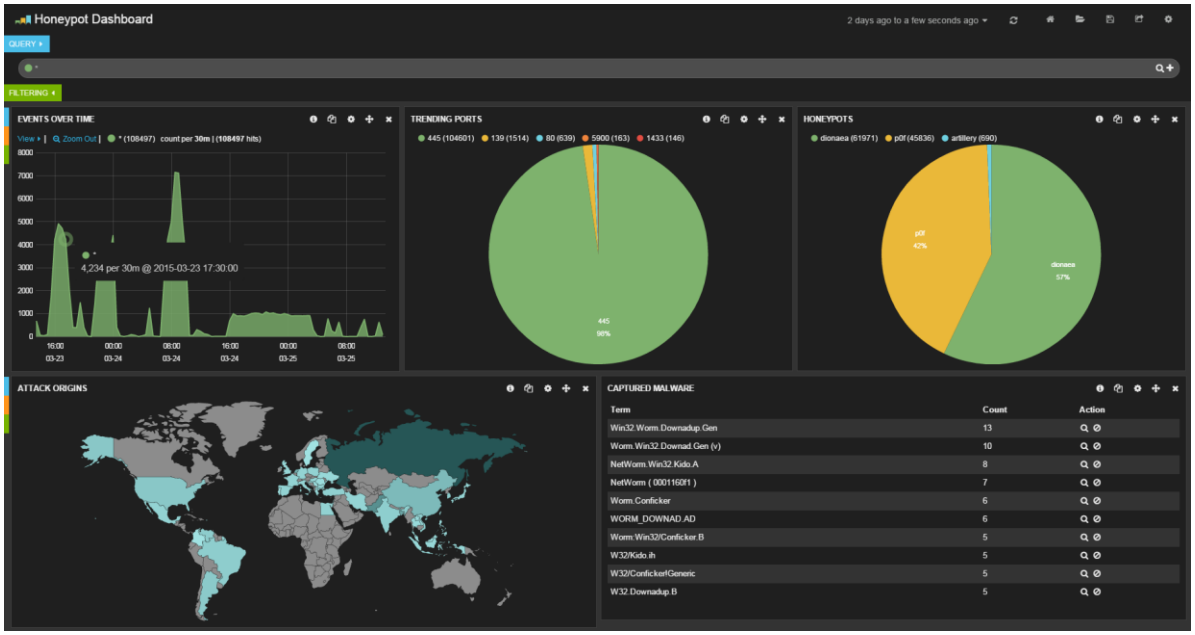



Figure 10: The resulting Kibana dashboard




# Bibliography

desaster. (2014). Opgeroepen op March 17, 2015, van desaster/kippo · GitHub: https://github.com/desaster/kippo

Elasticsearch. (2015). Opgeroepen op March 12, 2015, van Elastic · Revealing Insights from Data (Formerly Elasticsearch): https://www.elastic.co/

Koetter, M. (2009). Opgeroepen op March 12, 2015, van Dionaea - Catches bugs: http://dionaea.carnivore.it/

trustedsec. (2014). Opgeroepen op March 12, 2015, van trustedsec/artillery · GitHub: https://github.com/trustedsec/artillery

Zalewski, M. (2014). Opgeroepen op March 16, 2015, van p0f v3: http://lcamtuf.coredump.cx/p0f3/

# Appendix: Init script for Dionaea

```
#!/bin/sh -e
### BEGIN INIT INFO
# Provides:          dionaea
# Required-Start:    $syslog
# Required-Stop:     $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Start dionaea at boot time
# Description:       Dionaea intention is to trap malware exploiting vulnerabilities exposed by services offerd to a network, the ultimate goal is gaining a copy of the malware.
### END INIT INFO

PIDFILE=/var/run/dionaea.pid
DAEMON=/opt/dionaea/bin/dionaea
DESC="Dionaea"
ROOTDIR=/opt/dionaea/
test -x $DAEMON || exit 0
test -d $ROOTDIR || exit 0

case $1 in
start)
  echo -n "Starting $DESC: "
  if [ -e $PIDFILE ]; then
    echo "already running, please stop first"
    exit 1
  fi
  cd $ROOTDIR
  STATUS="OK"
  $DAEMON -D -l all -L '*' -p $PIDFILE > /dev/null || STATUS="FAILED"
  echo "$STATUS"
  ;;
stop)
  echo -n "Stopping $DESC: "
  if [ -e $PIDFILE ]; then
    neppid=`cat $PIDFILE`
    `kill -9 $neppid`;
    rm $PIDFILE
    echo "OK"
  else
    echo "failed: no pid found"
  fi
  ;;
restart)
  shift
  $0 stop ${@}
  sleep 1
  $0 start ${0}
  ;;
*)
  echo "Usage: $0 {start|stop|restart}" >&2
  exit 1
  ;;
esac
exit 0
```

# Appendix: SURFids module for Dionaea

```
from dionaea.core import ihandler, incident, g_dionaea
from dionaea.smb import smb

import os
import logging
import random

logger = logging.getLogger('surfids')
logger.setLevel(logging.DEBUG)

AS_POSSIBLE_MALICIOUS_CONNECTION   = 0x00000
AS_DEFINITLY_MALICIOUS_CONNECTION  = 0x00001

AS_DOWNLOAD_OFFER                  = 0x00010
AS_DOWNLOAD_SUCCESS                = 0x00020


DT_PROTOCOL_NAME                  = 80
DT_EMULATION_PROFILE               = 81
DT_SHELLCODE_ACTION                = 82
DT_DCERPC_REQUEST                 = 83
DT_VULN_NAME                    = 84

class surfidshandler(ihandler):
	def __init__(self, path):
		logger.debug("%s ready!" % (self.__class__.__name__))
		ihandler.__init__(self, path)

		# mapping socket -> attackid
		self.attacks = {}

		self.dbh = None

	def handle_incident(self, icd):
		origin = icd.origin
		origin = origin.replace(".","_")
		try:
			method = getattr(self, "_handle_incident_" + origin)
		except:
			return

		while True:
			try:
				method(icd)
				return
			except ConnectionError as e:
				#logger.warn("ConnectionError %s" % e)
				time.sleep(1)

	def _handle_incident_dionaea_connection_tcp_accept(self, icd):
		con=icd.con
		logger.info("[%s:%i to %s:%i] tcp accept" %
			(con.remote.host, con.remote.port, con.local.host, con.local.port))

	def _handle_incident_dionaea_connection_tcp_connect(self, icd):
		con=icd.con
		logger.info("[%s:%i to %s:%i] tcp connect (hostname: %s)" %
			(con.remote.host, con.remote.port, con.local.host, con.local.port, con.remote.hostname))

	def _handle_incident_dionaea_connection_tcp_reject(self, icd):
		con=icd.con
		logger.info("[%s:%i to %s:%i] tcp reject" %
			(con.remote.host, con.remote.port, con.local.host, con.local.port))

	def _handle_incident_dionaea_connection_free(self, icd):
		con=icd.con
		logger.info("[%s:%i to %s:%i] tcp disconnect" % (con.remote.host, con.remote.port, con.local.host, con.local.port)
)

	def _handle_incident_dionaea_module_emu_profile(self, icd):
		con = icd.con
		profile = str(icd.profile)
```

```
                logger.info("[%s:%i to %s:%i] emuprofile %s" % (con.remote.host, con.remote.port, con.local.host, con.local.port,
profile.replace('\n', '')))

        def _handle_incident_dionaea_download_offer(self, icd):
                con=icd.con
                logger.info("[%s:%i to %s:%i] download offer %s" % (con.remote.host, con.remote.port, con.local.host,
con.local.port, icd.url))

        def _handle_incident_dionaea_download_complete_hash(self, icd):
                con=icd.con
                logger.info("[%s:%i to %s:%i] download complete url %s with hash %s" % (con.remote.host, con.remote.port,
con.local.host, con.local.port, icd.url, icd.md5hash))

        def _handle_incident_dionaea_service_shell_listen(self, icd):
                con=icd.con
                logger.info("[%s:%i to %s:%i] listen shell %s" % (con.remote.host, con.remote.port, con.local.host, con.local.port,
"bindshell://"+str(icd.port)))

        def _handle_incident_dionaea_service_shell_connect(self, icd):
                con=icd.con
                logger.info("[%s:%i to %s:%i] connect shell %s" % (con.remote.host, con.remote.port, con.local.host,
con.local.port, "connectbackshell://"+str(icd.host)+":"+str(icd.port)))

        def _handle_incident_dionaea_modules_python_smb_dcerpc_request(self, icd):
                con=icd.con
                myuuid = icd.uuid.replace('-', '')
                try:
                        vuln = smb.registered_services[myuuid].vulns[icd.opnum]
                except:
                        vuln = "SMBDialogue"

                logger.info("dcerpc request for attackid %i" % attackid)
                logger.info("[%s:%i to %s:%i] dcerpc request %s (vulernability: %s)" % (con.remote.host, con.remote.port,
con.local.host, con.local.port, icd.uuid + ":" + str(icd.opnum), vuln))
```

# Appendix: Virustotal module for Dionaea

```
#********************************************************************************
#*                        Dionaea
#*                    - catches bugs -
#*
#*
#*
#* Copyright (C) 2010  Markus Koetter
#*
#* This program is free software; you can redistribute it and/or
#* modify it under the terms of the GNU General Public License
#* as published by the Free Software Foundation; either version 2
#* of the License, or (at your option) any later version.
#*
#* This program is distributed in the hope that it will be useful,
#* but WITHOUT ANY WARRANTY; without even the implied warranty of
#* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#* GNU General Public License for more details.
#*
#* You should have received a copy of the GNU General Public License
#* along with this program; if not, write to the Free Software
#* Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.
#*
#*
#*          contact nepenthesdev@gmail.com
#*
#********************************************************************************/

from dionaea.core import ihandler, incident, g_dionaea

import logging
import json
import os
import uuid
import sqlite3
from dionaea import pyev
from dionaea.core import g_dionaea

logger = logging.getLogger('virustotal')
logger.setLevel(logging.DEBUG)

class vtreport:
	def __init__(self, backlogfile, md5hash, file, status):
		self.backlogfile = backlogfile
		self.md5hash = md5hash
		self.file = file
		self.status = status

class virustotalhandler(ihandler):
	def __init__(self, path):
		logger.debug("%s ready!" % (self.__class__.__name__))
		ihandler.__init__(self, path)
		self.apikey = g_dionaea.config()['modules']['python']['virustotal']['apikey']
		self.cookies = {}
		self.loop = pyev.default_loop()

		self.backlog_timer = pyev.Timer(0, 20, self.loop, self.__handle_backlog_timeout)
		self.backlog_timer.start()
		p = g_dionaea.config()['modules']['python']['virustotal']['file']
		self.dbh = sqlite3.connect(p)
		self.cursor = self.dbh.cursor()
		self.cursor.execute("""
			CREATE TABLE IF NOT EXISTS backlogfiles (
				backlogfile INTEGER PRIMARY KEY,
				status TEXT NOT NULL, -- new, submit, query, comment
				md5_hash TEXT NOT NULL,
				path TEXT NOT NULL,
				timestamp INTEGER NOT NULL,
				scan_id TEXT,
				lastcheck_time INTEGER,
				submit_time INTEGER
			);""")

	def __handle_backlog_timeout(self, watcher, event):
```

```
            #logger.debug("backlog_timeout")

            # try to comment on files
            # comment on files which were submitted at least 60 seconds ago
            sfs = self.cursor.execute("""SELECT backlogfile, md5_hash, path FROM backlogfiles WHERE status = 'comment' AND
submit_time < strftime("%s",'now')-1*60 LIMIT 1""")
            for sf in sfs:
                self.cursor.execute("""UPDATE backlogfiles SET status = 'comment-' WHERE backlogfile = ?""", (sf[0],))
                self.dbh.commit()
                self.make_comment(sf[0], sf[1], sf[2], 'comment')
                return

            # try to receive reports for files we submitted
            sfs = self.cursor.execute("""SELECT backlogfile, md5_hash, path FROM backlogfiles WHERE status = 'query' AND
submit_time < strftime("%s",'now')-15*60 AND lastcheck_time < strftime("%s",'now')-15*60 LIMIT 1""")
            for sf in sfs:
                self.cursor.execute("""UPDATE backlogfiles SET status = 'query-' WHERE backlogfile = ?""", (sf[0],))
                self.dbh.commit()
                self.get_file_report(sf[0], sf[1], sf[2], 'query')
                return

            # submit files not known to virustotal
            sfs = self.cursor.execute("""SELECT backlogfile, md5_hash, path FROM backlogfiles WHERE status = 'submit' LIMIT
1""")
            for sf in sfs:
                self.cursor.execute("""UPDATE backlogfiles SET status = 'submit-' WHERE backlogfile = ?""", (sf[0],))
                self.dbh.commit()
                self.scan_file(sf[0], sf[1], sf[2], 'submit')
                return

            # query new files
            sfs = self.cursor.execute("""SELECT backlogfile, md5_hash, path FROM backlogfiles WHERE status = 'new' ORDER
BY timestamp DESC LIMIT 1""")
            for sf in sfs:
                self.cursor.execute("""UPDATE backlogfiles SET status = 'new-' WHERE backlogfile = ?""", (sf[0],))
                self.dbh.commit()
                self.get_file_report(sf[0], sf[1], sf[2], 'new')
                return

        def stop(self):
            self.backlog_timer.stop()
            self.backlog_timer = None

        def handle_incident(self, icd):
            pass

        def handle_incident_dionaea_download_complete_unique(self, icd):
            self.cursor.execute("""INSERT INTO backlogfiles (md5_hash, path, status, timestamp) VALUES
(?,?,?,strftime("%s",'now')) """, (icd.md5hash, icd.file, 'new'))
            logger.debug("New file captured: {}".format(md5_hash))

        def get_file_report(self, backlogfile, md5_hash, path, status):
            cookie = str(uuid.uuid4())
            self.cookies[cookie] = vtreport(backlogfile, md5_hash, path, status)

            i = incident("dionaea.upload.request")
            i._url = "http://www.virustotal.com/api/get_file_report.json"
            i.resource = md5_hash
            i.key = self.apikey
            i._callback = "dionaea.modules.python.virustotal.get_file_report"
            i._userdata = cookie
            i.report()

        def handle_incident_dionaea_modules_python_virustotal_get_file_report(self, icd):
            f = open(icd.path, mode='r')
            j = json.load(f)

            cookie = icd._userdata
            vtr = self.cookies[cookie]

            if j['result'] == -2:
                logger.warn("Virustotal API Throttle")
                self.cursor.execute("""UPDATE backlogfiles SET status = ? WHERE backlogfile = ?""", (vtr.status,
vtr.backlogfile))
                self.dbh.commit()
            elif j['result'] == -1:
                logger.warn("Something is wrong with your Virustotal API key")
```

```
            elif j['result'] == 0: # file unknown
                    # mark for submit
                    if vtr.status == 'new':
                            self.cursor.execute("""UPDATE backlogfiles SET status = 'submit', lastcheck_time = strftime("%s",'now')
WHERE backlogfile = ?""", (vtr.backlogfile,))
                    elif vtr.status == 'query':
                            self.cursor.execute("""UPDATE backlogfiles SET lastcheck_time = strftime("%s",'now') WHERE backlogfile
= ?""", (vtr.backlogfile,))
                    self.dbh.commit()
            elif j['result'] == 1: # file known
                    self.cursor.execute("""UPDATE backlogfiles SET status = 'comment', lastcheck_time = strftime("%s",'now')
WHERE backlogfile = ?""", (vtr.backlogfile,))
                    self.cursor.execute("""DELETE FROM backlogfiles WHERE backlogfile = ?""", (vtr.backlogfile,) )
                    self.dbh.commit()

                    logger.debug("Report {}".format(j) )
                    date = j['report'][0]
                    scans = j['report'][1]
                    for av in scans:
                            logger.debug("Scanner {} reported {} for {}".format(av,scans[av],vtr.md5hash))

                    i = incident("dionaea.modules.python.virustotal.report")
                    i.md5hash = vtr.md5hash
                    i.path = icd.path
                    i.report()
            else:
                    logger.warn("Virustotal reported {}".format(j))
            del self.cookies[cookie]

    def scan_file(self, backlogfile, md5_hash, path, status):
            cookie = str(uuid.uuid4())
            self.cookies[cookie] = vtreport(backlogfile, md5_hash, path, status)

            i = incident("dionaea.upload.request")
            i._url = "http://www.virustotal.com/api/scan_file.json"
            i.key = self.apikey
            i.set('file://file', path)
            i._callback = "dionaea.modules.python.virustotal_scan_file"
            i._userdata = cookie
            i.report()


    def handle_incident_dionaea_modules_python_virustotal_scan_file(self, icd):
            f = open(icd.path, mode='r')
            j = json.load(f)
            logger.debug("scan_file {}".format(j))
            cookie = icd._userdata
            vtr = self.cookies[cookie]

            if j['result'] == -2:
                    logger.warn("Virustotal API throttle")
                    self.cursor.execute("""UPDATE backlogfiles SET status = ? WHERE backlogfile = ?""", (vtr.status,
vtr.backlogfile))
                    self.dbh.commit()
            elif j['result'] == -1:
                    logger.warn("Something is wrong with your Virustotal API key")
            elif j['result'] == 1:
                    scan_id = j['scan_id']
                    # recycle this entry for the query
                    self.cursor.execute("""UPDATE backlogfiles SET scan_id = ?, status = 'comment', submit_time =
strftime("%s",'now') WHERE backlogfile = ?""", (scan_id, vtr.backlogfile,))
                    self.dbh.commit()
            del self.cookies[cookie]

    def make_comment(self, backlogfile, md5_hash, path, status):
            cookie = str(uuid.uuid4())
            self.cookies[cookie] = vtreport(backlogfile, md5_hash, path, status)

            i = incident("dionaea.upload.request")
            i._url = "http://www.virustotal.com/api/make_comment.json"
            i.key = self.apikey
            i.file = md5_hash
            i.tags = "honeypot;malware;networkworm"
            i.comment = "This sample was captured in the wild and uploaded by the dionaea honeypot."
            i._callback = "dionaea.modules.python.virustotal_make_comment"
            i._userdata = cookie
            i.report()
```

```
    def handle_incident_dionaea_modules_python_virustotal_make_comment(self, icd):
        cookie = icd._userdata
        vtr = self.cookies[cookie]
        f = open(icd.path, mode='r')
        try:
            j = json.load(f)
            if j['result'] == -2:
                logger.warn("Virustotal API throttle")
                self.cursor.execute("""UPDATE backlogfiles SET status = ? WHERE backlogfile = ?""", (vtr.status,
vtr.backlogfile))
                self.dbh.commit()
            elif j['result'] == -1:
                logger.warn("Something is wrong with your Virustotal API key")
            elif j['result'] == 1:
                self.cursor.execute("""UPDATE backlogfiles SET status = 'query' WHERE backlogfile = ? """,
(vtr.backlogfile, ))
                logger.debug("Status for file {} changed to query".format(vtr.backlogfile))
                self.dbh.commit()

        except Exception as e:
            pass
        del self.cookies[cookie]
```

# Appendix: Init script for p0f

```
### BEGIN INIT INFO
# Provides:          p0f
# Required-Start:    $all
# Required-Stop:
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: p0f
# Description:       p0f
#                    Passive OS Fingerprinting
### END INIT INFO


# Using the lsb functions to perform the operations.
. /lib/lsb/init-functions
# Process name ( For display )
NAME=p0f
# Daemon name, where is the actual executable
DAEMON=/usr/sbin/p0f

# pid file for the daemon
PIDFILE=/var/run/p0f.pid

PARAMETERS="-u root -i eth0 -Q /tmp/p0f.sock -q -l -d -o /var/log/p0f/p0f.log"

# If the daemon is not there, then exit.
test -x $DAEMON || exit 5

case $1 in
 start)
  # Checked the PID file exists and check the actual status of process
  if [ -e $PIDFILE ]; then
   status_of_proc -p $PIDFILE $DAEMON "$NAME process" && status="0" || status="$?"
   # If the status is SUCCESS then don't need to start again.
   if [ $status = "0" ]; then
    exit # Exit
   fi
  fi
  # Start the daemon.
  log_daemon_msg "Starting the process" "$NAME"
  # Start the daemon with the help of start-stop-daemon
  # Log the message appropriately
  if start-stop-daemon --start --quiet --oknodo --pidfile $PIDFILE --exec $DAEMON -- $PARAMETERS; then
   PID=`pidof -s p0f`
   if [ $PID ] ; then
                echo $PID >$PIDFILE
   fi
   log_end_msg 0
  else
   log_end_msg 1
  fi
  ;;
 stop)
  # Stop the daemon.
  if [ -e $PIDFILE ]; then
   status_of_proc -p $PIDFILE $DAEMON "Stoppping the $NAME process" && status="0" || status="$?"
   if [ "$status" = 0 ]; then
    start-stop-daemon --stop --quiet --oknodo --pidfile $PIDFILE
    /bin/rm -rf $PIDFILE
   fi
  else
   log_daemon_msg "$NAME process is not running"
   log_end_msg 0
  fi
  ;;
 restart)
  # Restart the daemon.
  $0 stop && sleep 2 && $0 start
  ;;
 status)
  # Check the status of the process.
  if [ -e $PIDFILE ]; then
   status_of_proc -p $PIDFILE $DAEMON "$NAME process" && exit 0 || exit $?
  else
```

```
  log_daemon_msg "$NAME Process is not running"
  log_end_msg 0
 fi
 ;;
reload)
 # Reload the process. Basically sending some signal to a daemon to reload
 # it configurations.
 if [ -e $PIDFILE ]; then
  start-stop-daemon --stop --signal USR1 --quiet --pidfile $PIDFILE --name $NAME
  log_success_msg "$NAME process reloaded successfully"
 else
  log_failure_msg "$PIDFILE does not exists"
 fi
 ;;
*)
 # For invalid arguments, print the usage message.
 echo "Usage: $0 {start|stop|restart|reload|status}"
 exit 2
 ;;
esac
```